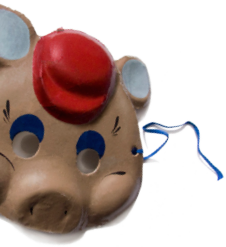
ПРОФЕССИОНАЛЬНО

Алан Купер

ОБ ИНТЕРФЕЙСЕ

основы проектирования взаимодействия



Алан КУПЕР
Роберт РЕЙМАН
Дэвид КРОНИН

ABOUT FACE 3

СИМВОЛ®

По договору между издательством «Символ-Плюс» и Интернет-магазином «Books.Ru – Книги России» единственный легальный способ получения данного файла с книгой ISBN 978-5-93286-132-5, название «Алан Купер об интерфейсе. Основы проектирования взаимодействия» – покупка в Интернет-магазине «Books.Ru – Книги России». Если Вы получили данный файл каким-либо другим образом, Вы нарушили международное законодательство и законодательство Российской Федерации об охране авторского права. Вам необходимо удалить данный файл, а также сообщить издательству «Символ-Плюс» (piracy@symbol.ru), где именно Вы получили данный файл.

About Face 3

The Essentials of Interaction Design

Alan Cooper, Robert Reimann, Dave Cronin

Wiley Publishing, Inc.

ПРОФЕССИОНАЛЬНО

Алан Купер об интерфейсе

Основы проектирования взаимодействия

Алан Купер, Роберт Рейман, Дэвид Кронин

СИМВОЛ®

Санкт-Петербург — Москва
2009

Серия «Профессионально»
Алан Купер, Роберт Рейман, Дэвид Кронин

Алан Купер об интерфейсе
Основы проектирования взаимодействия

Перевод М. Зислиса

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Научный редактор	*А. Копылов*
Редактор	*В. Подобед*
Художник	*О. Макарова*
Корректор	*С. Минин*
Верстка	*Д. Орлова*

Купер А., Рейман Р., Кронин Д.
Алан Купер об интерфейсе. Основы проектирования взаимодействия. – Пер. с англ. – СПб.: Символ-Плюс, 2009. – 688 с., ил.
ISBN 978-5-93286-132-5

Когда в 1995 году увидело свет первое издание «About Face», идея проектировать продукты исходя из целей людей казалась революционной. Благодаря работам Алана Купера и других первопроходцев, проектирование взаимодействия получило сегодня широкое признание как уникальная и крайне важная дисциплина, однако эта работа далека от завершения.

Авторы полностью обновленного издания, признанные мировые эксперты в вопросах создания интерфейсов, детально описывают разработанный в компании Cooper и примененный во множестве проектов целостный подход к проектированию взаимодействия, ориентированный на цели пользователя. Отличительной чертой книги является ее практическая направленность – значительную часть издания занимает подробный разбор принципов и шаблонов проектирования взаимодействия. Большое внимание уделено новым информационным средам: веб-приложениям, мобильным приложениям, киоскам и т. п.

Книга адресована всем специалистам, по роду деятельности соприкасающимся с процессом создания цифровых продуктов. Проектировщикам взаимодействия и дизайнерам интерфейсов она послужит настольным справочником по организации процесса и повседневным подручным инструментарием.

ISBN 978-5-93286-132-5
ISBN 978- 0-470-08411-3 (англ)

© Издательство Символ-Плюс, 2009
Authorized translation of the English edition © 2007 Wiley Publishing, Inc. This translation is published and sold by permission of Wiley Publishing, Inc., the owner of all rights to publish and sell the same.

Все права на данное издание защищены Законодательством РФ, включая право на полное или частичное воспроизведение в любой форме. Все товарные знаки или зарегистрированные товарные знаки, упоминаемые в настоящем издании, являются собственностью соответствующих фирм.

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 324-5353, www.symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Подписано в печать 06.04.2009. Формат 70x100 1/16 . Печать офсетная.
Объем 43 печ. л. Тираж 1500 экз. Заказ N
Отпечатано с готовых диапозитивов в ГУП «Типография «Наука»
199034, Санкт-Петербург, 9 линия, 12.

Сью, моей лучшей подруге во всех жизненных испытаниях.

Максвеллу Аарону Рейману.

Гретхен.

А также куперистам прошлого, настоящего и будущего
и тем умеющим мечтать практикам, чьими усилиями
проектирование взаимодействия превратилось в профессию.

Оглавление

Об авторах 12

Предисловие. Постиндустриальный мир 13

Введение к третьему изданию 19

I. Введение в целеориентированное проектирование 31

1. Проектирование, ориентированное на цели 33

Цифровым продуктам необходимы более качественные методы проектирования 33
Эволюция проектирования в промышленности 41
Планирование и проектирование поведения 43
Выявление целей пользователей 44
Целеориентированный процесс проектирования 48

2. Модели реализации и ментальные модели 59

Модели реализации 59
Пользовательские ментальные модели 60
Модели представления 61
Большинство программных продуктов следуют модели реализации 64
Модели представления механической и информационной эры 67

3. Новички, эксперты и середняки 73

Вечные середняки 73
Проектирование для пользователей с различной подготовкой 76

4. Как понять пользователей: качественные исследования 81

Качественные и количественные исследования 81
Этнографические интервью: интервьюирование и наблюдение за пользователями 90
Прочие виды исследований 102

5. Модели пользователей: персонажи и цели 109

Для чего нам модели? 110
Персонажи 111
Цели 123
Разработка персонажей 133
Прочие модели 143

6. Основы проектирования: сценарии и требования 146

Сценарии: повествование как средство проектирования 146
Требования: информационное обеспечение проектирования взаимодействия 151
Выработка требований с использованием персонажей и сценариев 153

7. От требований к пользовательскому интерфейсу: общая инфраструктура и детализация 162

Общая инфраструктура пользовательского интерфейса 162
Детализация формы и поведения 179
Проверка результата проектирования и юзабилити-тестирование 181

II. Проектирование облика и поведения 187

8. Создание качественного интерфейса: принципы и шаблоны 189

Принципы проектирования взаимодействия 189
Ценности проектирования 191
Шаблоны проектирования взаимодействия 196

9. Техническая платформа и тип интерфейса 200

Тип интерфейса 201
Проектирование настольных приложений 202
Проектирование в среде Всемирной паутины 214
Прочие платформы 223

10. Оркестровка и состояние потока 243

Состояние потока и прозрачность 243
Проектирование гармоничного взаимодействия 245

11. Оптимизация налогообложения 266

Налоги в графическом пользовательском интерфейсе 267
Прекращение работы 271
Распространенные налоговые ловушки 274
Навигация как налог 275
Улучшение навигации 281

12. Проектирование хорошего поведения 293

Проектирование тактичных продуктов 294
Проектирование интеллектуальных продуктов 304

13. Метафоры, идиомы, ожидаемое назначение 315

Парадигмы интерфейса 316
Еще об ограничениях метафор 322
Построение идиом 327
Ожидаемые физические назначения 329

14. Визуальный дизайн интерфейсов 333

Изобразительное искусство, визуальный дизайн интерфейсов и прочие дисциплины дизайна 334
Строительные блоки визуального дизайна интерфейсов 336
Принципы визуального дизайна интерфейса 339
Принципы визуального информационного дизайна 361
Единство и стандарты 365

III. Детальное проектирование взаимодействия 369

15. Совершенствуем поиск и извлечение данных 371

Системы хранения и извлечения информации 372
Хранение и извлечение в физическом мире 372
Хранение и извлечение в цифровом мире 374
Реляционные базы данных и «цифровой бульон» 379
Вывод на естественном языке: идеальный интерфейс для извлечения по атрибутам 382

16. Отмена 384

Пользователи и отмена 384
Проектирование функции отмены 386
Типы и варианты отмены 387
Прочие модели для механизмов, схожих с отменой 392
Необратимые действия 398

17. Новый взгляд на файлы и операцию сохранения 399

Что не так с сохранением файлов? 400
Проблемы модели реализации 402
Модель реализации против ментальной модели 405
Прощаемся с моделью реализации 406
Проектирование с унифицированной файловой моделью 408
Являются ли диски и файловые системы важным конструктивным элементом? 414
Время перемен 416

18. Улучшаем ввод данных 417

Целостность данных и информационный иммунитет 417
Аудит и редактирование 422

19. Указание, выделение, непосредственное манипулирование 425

Непосредственное манипулирование 425
Устройства указания 427
Указание и курсор 437
Выделение 441
Перетаскивание 449
Манипулирование элементами управления 461
Инструменты палитры 462
Манипулирование объектами 465
Связывание объектов 474

20. Поведение окон 476

PARC и Alto 476
Принципы PARC 478
Microsoft и окна плиткой 480
Полноэкранные приложения 481
Многопанельные приложения 482
Проектирование окон 483
Состояния окон 490
MDI против SDI 491

21. Элементы управления 494

Нет окнам, перегруженным элементами управления! 494
Командные элементы управления 495
Элементы управления выбором 498
Элементы ввода 514
Элементы управления отображением 527

22. Меню 532

Немного истории 532
Современные меню: средство обучения 538
Необязательные меню 543
Идиомы меню 545

23. Панели инструментов 554

Панели инструментов: наглядные, мгновенно исполняемые команды 554
Панели инструментов и меню 555

Панели инструментов и элементы управления на панелях инструментов . . . 556
Элементы управления на панели инструментов: обучение . . . 558
Эволюция панели инструментов . . . 560

24. Диалоговые окна . . . 566

Уместное применение диалоговых окон . . . 566
Основы применения диалоговых окон . . . 568
Модальные диалоговые окна . . . 570
Немодальные диалоговые окна . . . 571
Четыре назначения диалоговых окон . . . 578
Управление содержимым диалоговых окон . . . 586

25. Ошибки, уведомления, подтверждения . . . 592

Диалоги сообщений об ошибках . . . 592
Диалоговые окна уведомлений: сообщение об очевидном . . . 602
Диалоговое окно подтверждения . . . 605
Заменяем диалоговые окна обогащенной немодальной обратной связью . . . 608

26. Проектирование для различных потребностей . . . 615

Командные векторы и рабочие наборы . . . 615
Перевод новичков в середняки . . . 617
Персонализация и настройка . . . 620
Идиосинкратически модальное поведение . . . 623
Локализация и глобализация . . . 624
Коллекции и шаблоны . . . 625
Справка . . . 625

Послесловие: несколько слов о сотрудничестве . . . 630

Приложение. Принципы проектирования . . . 634

Библиография . . . 640

Словарь терминов . . . 646

Алфавитный указатель . . . 649

Об авторах

Алан Купер – новатор в области программного обеспечения, программист, проектировщик и теоретик. Его упоминают как создателя первых серьезных деловых программ для микрокомпьютеров и часто называют «отцом языка Visual Basic». Последние 15 лет компания Алана, Cooper, занималась консультированием в области проектирования взаимодействия, помогая другим компаниям создавать новые продукты и совершенствовать поведение цифровых технологий. В компании Cooper Алан возглавлял работы по созданию новой методологии проектирования успешного программного обеспечения, которую он называет целеориентированным процессом. В ходе этой работы, в частности, появились персонажи – инструмент, получивший широкое распространение после выхода в 1998 году второй книги Алана «Психбольница в руках пациентов». Купер известен также как писатель, лектор и горячий сторонник гуманизации технологий.

Роберт Рейман посвятил последние 15 лет расширению границ цифровых продуктов в роли проектировщика, писателя, лектора и консультанта. Он возглавлял десятки проектов по проектированию взаимодействия в таких областях, как электронная коммерция, порталы, персональная производительность, среды создания контента, медицинские и научные приборы, беспроводные технологии и портативные устройства, работая как с начинающими компаниями, так и с компаниями из числа Fortune 500. В качестве главы исследовательского отдела в Cooper Рейман руководил разработкой и совершенствованием многих из целеориентированных методов, описанных в данной книге. В 2005 году Рейман стал первым президентом ассоциации проектирования взаимодействия (IxDA, www.ixda.org) – глобальной некоммерческой организации, объединяющей проектировщиков взаимодействия. В настоящий момент он отвечает за проектирование опыта взаимодействия в Bose Corporation.

Дэйв Кронин – руководитель отдела проектирования взаимодействия в Cooper. Он помогает проектировать продукты, служащие нуждам таких групп людей, как хирурги, посетители музеев, маркетологи, инвестиционные аналитики, интернет-покупатели, персонал больниц, водители, стоматологи, финансовые аналитики, логистики, пожилые и немощные люди. Работая в компании Cooper, он также внес существенный вклад в процесс создания и совершенствования методов целеориентированного проектирования, описываемых в этой книге.

Предисловие. Постиндустриальный мир

Промышленная эра завершилась. Производство, бывшее основным стимулом развития экономики в последние 175 лет, сдало свои позиции. Да, объемы производства сегодня велики как никогда, однако лидерство принадлежит цифровым технологиям, а в экономике ключевую роль играет программное обеспечение. Произошел переход от атомов к битам. На дворе постиндустриальная эра.

Все больше и больше продуктов содержат в себе программный код. В моей духовке есть микросхема, управляющая подсветкой, вентиляцией и температурой нагрева. Когда мне доставляют товары на дом, я расписываюсь уже не на листке бумаги, а в компьютере. Выбирая автомобиль, в действительности я выбираю навигационную систему.

Все большее число компаний сегодня полностью зависят от программного обеспечения, и речь идет не только о таких очевидных примерах, как Amazon.com и Microsoft. Тысячи компаний от мала до велика, предлагающих продукты и предоставляющих услуги в самых различных областях бизнеса, используют программное обеспечение во всех сферах своей деятельности, будь то финансовые операции, управление, планирование или продажи. Системы бизнес-логики, поддерживающие работу крупных компаний, – это программные системы. Поиск и найм сотрудников, управление персоналом, инвестиции и арбитражные операции, закупки и организация поставок, финансовые операции и поддержка принятия решений – все эти системы сегодня основаны на программном коде. Продажи и маркетинг осуществляются преимущественно через Интернет. На «передовой» больше нет живых людей – теперь там находится программное обеспечение. Поставщики, клиенты, коллеги, сотрудники – все они общаются с компаниями посредством программ или при поддержке программ.

Организационные структуры и методы управления, хорошо зарекомендовавшие себя в прошлом в промышленном производстве, сегодня, в постиндустриальную эру, подводят нас. Подводят потому, что ориентированы на преобразования и перемещение таких вещей, которые сделаны из атомов. Нужные нам атомы существуют в ограниченных количествах и требуют значительных энергозатрат для преобразования и транспортировки. Программы сделаны из битов, а не из атомов,

и это качественное отличие. Существует бесконечное количество битов, а их транспортировка, преобразование и даже копирование практически не требуют затрат энергии.

Отличаются и люди, создающие программы. Склонности, отношение к работе, подготовка, язык, инструменты и система ценностей у среднего программиста и среднего рабочего на конвейере качественно различны. Наиболее эффективные способы управления программистами – радикально иные по сравнению с эффективными способами управления рабочим классом, применявшимися в прошлом. Чтобы добиться от программистов выполнения задач компании, требуются навыки, неведомые управленцам промышленной эпохи.

Снижение стоимости производства было главным достижением индустриализации. Поэтому лучшие и светлейшие умы этой эры бились над уменьшением расходов, связанных с созданием продукции. В постиндустриальную эру стоимость сырья, сборки и поставки продукции одинаково мала для всех игроков рынка. Единственный действенный путь снижения затрат на производство связан с автоматизацией, планированием и управлением знаниями, т. е. с программами. Другими словами, вместо того чтобы экономить доллар на создании каждой безделушки, можно сэкономить миллион, произведя наиболее желанный товар в нужном количестве.

Будучи единожды созданной, программа может копироваться бесконечное число раз практически бесплатно. Сокращение затрат на написание этой программы не приносит практически никакой выгоды и, как правило, выливается в снижение качества. Поэтому основное уравнение бизнеса промышленной эпохи сегодня оказывается перевернутым: лучшие и светлейшие головы посвящают себя повышению эффективности программ и качества взаимодействия с ними. Не стоит забывать, что все современные финансово-расчетные системы направлены на пристальное отслеживание стоимости производства и потому не способны корректно отражать состояние бизнеса, основанного на программном обеспечении. Когда руководство принимает решения на основе этих ошибочных показателей, теряется масса времени, тратятся лишние деньги и упускаются благоприятные возможности.

Неудивительно, что компании сталкиваются с огромными трудностями при создании программ. Талантливые управленцы обнаруживают, что их замыслы неуловимо, но существенно видоизменяются где-то на пути от концепции продукта к его выпуску. Хорошие с виду планы оказываются непригодными, когда дело доходит до управления созданием программного продукта. Пришла пора избавиться от устаревших методов управления и дать дорогу проектированию взаимодействия как ключевому инструменту создания программных продуктов и управления их разработкой.

Со дня первой публикации этой книги в 1995 году область проектирования взаимодействия значительно выросла и возмужала. Просуществово-

вав долгое время под гнетом метода проб и ошибок и решений, принимаемых задним числом, ныне проектирование взаимодействия вместе со своими вариациями и смежными областями превратилось в четкий, надежный, эффективный инструмент проектирования успешного поведения продуктов. С изобретением и внедрением методики персонажей, усовершенствованием текстовых сценариев взаимодействия и с появлением практики целеориентированного проектирования (Goal-Directed™ Design) в целом все компании получили в свое распоряжение средства, которые позволяют создавать программные продукты, ориентированные на высококачественное взаимодействие с пользователями.

Более того, проектирование взаимодействия превратилось в чрезвычайно мощный инструмент управления разработкой программного продукта. Предоставляя описание окончательного варианта продукта, проектирование взаимодействия снабжает разработчиков аналогом чертежей, который помогает программистам осознавать, что именно они создают, а руководителям – оценивать прогресс в работе программистов.

Проектирование взаимодействия проявило себя и как маркетинговый инструмент, предоставляющий предельно ясную и точную информацию о том, кто конкретно будет использовать продукт и для чего. Понимание источника мотивации покупателей – манна небесная для маркетологов. Качественные исследования и аналитические аспекты целеориентированного проектирования позволяют достичь глубокого понимания рынка.

После того как начался интернет-бум, породивший ощущение, что отбрасывание здравого смысла – самый верный путь к мгновенному обогащению, мне все чаще приходилось слышать от разумных и вроде бы вменяемых людей, что знать, чего хочет пользователь, попросту невозможно! Это убеждение, конечно, позволяет увильнуть от необходимости действительно понять, чего хочет пользователь, однако нет ничего более далекого от истины. В моей компании Cooper поставленные клиентами задачи заставляют проектировщиков постигать премудрости финансов, здравоохранения, фармацевтики, управления персоналом, инструментариев программирования, музейного дела, потребительского кредитования и многих других специфических областей. Наши команды, не располагающие подготовкой в интересующих клиентов областях, а иногда даже и не имеющие представления об оных, к восхищению наших клиентов всего за несколько недель становятся достаточно подкованными в теме. Как нам это удается? Секрет в том, что за отправную точку мы неизменно принимаем человека, а не технологию.

Проектирование взаимодействия – это инструмент, позволяющий «узнавать, чего хочет пользователь». Вооружившись этим знанием, вы можете создавать более качественные, более успешные цифровые продукты, приносящие больше денег. А благодаря лояльности довольных пользователей вы сумеете занять своими решениями рынок. Мы

много раз видели, как перегруженные функциями продукты, первыми успевшие на рынок, оказывались вытесненными более поздними продуктами, которые были тщательнее продуманы в плане взаимодействия с пользователем. Не лучше ли продумать все до того, как самый первый выпуск продукта выведет его на далекую от оптимальности траекторию существования?

Ничто так не вдохновляет, как успех. Успех практического применения принципов и методов, лежащих в основе этой книги, а также родственных им, отчетливо демонстрирует, что программные продукты не настолько гибки, как многим казалось, и что доскональное изучение пользователей и детальное планирование в постиндустриальном веке важны как никогда.

Если вы преданы идее улучшения мира через совершенствование поведенческих аспектов цифровых продуктов и услуг, я рад приветствовать вас во вселенной целеориентированного проектирования!

Алан Купер

Благодарности

Мы хотели бы выразить свою глубочайшую признательность следующим людям, без которых не появилось бы на свет новое издание «About Face»: Крису Уэббу (Chris Webb) из Wiley, который понял, что настало время для нового издания, Сью Купер (Sue Cooper), которая согласилась с этой мыслью, и Саре Шлейр (Sara Shlaer) из Wiley, которая терпеливо помогала нам в подготовке всех редакций этой книги.

Мы хотели бы также поблагодарить наших коллег и проектировщиков из Cooper за их вклад в эту работу и в предыдущие ее воплощения: Ким Гудвин (Kim Goodwin) за серьезный вклад в разработку и описание концепций и методов, представленных на этих страницах; Ребекку Бортман (Rebecca Bortman) и Ника Майерса (Nick Myers) за пересмотр оформления книги и обложки, а также за иллюстрации; Хью Дабберли (Hugh Dubberly) за помощь в разработке принципов, приводимых в конце главы 8, и за иллюстрации к целеориентированному процессу в главе 1; Гретхен Андерсон (Gretchen Anderson), Элейн Монтгомери (Elaine Montgomery) и Дага Лемуана (Doug LeMoine) за их вклад в материал об исследованиях пользователей и рынка в главе 4; Рика Бонда (Rick Bond) за его многочисленные идеи в связи с юзабилити-тестированием (глава 7); Эрнеста Кинсолвинга (Ernest Kinsolving) и Йорга Берингера (Joerg Beringer) из SAP за их вклад в описание типов приложений веб-порталов в главе 9; Криса Уилдрайера (Chris Weeldreyer) за идеи по проектированию встроенных систем (глава 9); Уэйна Гринвуда (Wayne Greenwood) за вклад в тему отображения элементов управления (глава 10); Нейта Фортена (Nate Fortin) и Ника Майерса (Nick Myers) за вклад в тему визуального проектирования интерфейсов и брендинга (глава 14). Мы говорим спасибо Элизабет Бейкон (Elizabeth Bacon), Стиву Калду (Steve Calde), Джону Даннингу (John Dunning), Дэвиду Фору (David Fore), Нейту Фортену (Nate Fortin), Ким Гудвин (Kim Goodwin), Уэйну Гринвуду (Wayne Greenwood), Ноа Гийо (Noah Guyot), Лейн Халли (Lane Halley), Эрнесту Кинсолвингу (Ernest Kinsolving), Дэниелу Куо (Daniel Kuo), Берму Ли (Berm Lee), Дагу Лемуану (Doug LeMoine), Тиму Маккою (Tim McCoy), Элейн Монтгомери (Elaine Montgomery), Нику Майерсу (Nick Myers), Крису Нойсселу (Chris Noessel), Райану Ольшавски (Ryan Olshavsky), Анджеле Куэйл (Angela Quail), Сьюзи Томпсон (Suzy Thompson), а также Крису Уилдрайеру (Chris Weeldreyer) за их вклад в работы компании Cooper и иллюстрации, представленные в данной книге.

Мы выражаем признательность нашим клиентам – Дэвиду Уэсту (David West) из Shared Healthcare Systems, Майку Кею (Mike Kay) и Биллу Чангу (Bill Chang) из Fujitsu Softek, Джону Чаффинсу (John Chaffins) из CrossCountry, Крису Тугуду (Chris Twogood) из Teradata и Крису Доллару (Chris Dollar) из McKesson – за разрешение использовать проекты, выполнявшиеся для них компанией Cooper, в качестве примеров в этой книге. Мы хотим также поблагодарить многих других клиентов, у которых хватило прозорливости и воображения, чтобы работать с нами и отстаивать наш подход в своих организациях.

Мы хотели бы также засвидетельствовать свое почтение следующим авторам и коллегам, чьи работы много лет служили для нас источником вдохновения и призмой для рассмотрения идей: Кристофер Александер (Christopher Alexander), Эдвард Тафт (Edward Tufte), Кевин Маллет (Kevin Mullet), Виктор Папанек (Victor Papanek), Дональд Норман (Donald Norman), Ларри Константайн (Larry Constantine), Чаллис Ходж (Challis Hodge), Шелли Ивенсон (Shelley Evenson), Клиффорд Насс (Clifford Nass), Байрон Ривз (Byron Reeves), Стивен Пинкер (Stephen Pinker) и Терри Суок (Terry Swack).

Наконец, следует отметить, что фрагменты главы 5, касающиеся когнитивных процессов обработки, изначально были опубликованы в статье Роберта Реймана на сайте UXMatters.com и использованы с разрешения владельцев авторских и имущественных прав.

Введение к третьему изданию

Эта книга посвящена **проектированию взаимодействия** – практике создания цифровых интерактивных продуктов, сред, систем и служб. Как и многие другие дисциплины проектирования, проектирование взаимодействия работает с формой. Однако в первую очередь оно сосредоточено на аспекте, который нечасто затрагивают традиционные дисциплины проектирования и дизайна, – на проектировании *поведения*.

Почти все виды проектирования и дизайна *влияют* на поведение человека: архитектура имеет дело с тем, как люди используют физическое пространство, а графический дизайн – с вопросами мотивации или получения необходимых реакций от людей. Однако сегодня, когда повсеместное распространение получили устройства с микросхемами внутри – от компьютеров до автомобилей и телефонов, – мы ежедневно создаем продукты, которые *демонстрируют* сложное поведение.

Возьмем для примера такую простую вещь, как духовка. До наступления цифровой эры управление духовкой было весьма простым – достаточно было повернуть единственный регулятор в нужное положение. Выключенной духовке соответствовало строго одно положение, каждому из возможных температурных режимов также отвечало одно положение. Всякий раз при повороте регулятора в конкретное положение *происходило одно и то же*. Это можно назвать «поведением», но это определенно очень простое, механическое поведение. Сравните его с поведением современных микроволновок с микросхемами и жидкокристаллическими экранами. Они изобилуют кнопками с надписями, не имеющими отношения к приготовлению пищи: «Начать», «Отменить», «Программировать», которые соседствуют с более привычными, такими как «Запекать» и «Жарить». Угадать, что случится, если нажать какую-либо из таких кнопок, гораздо сложнее, чем в случае со старым регулятором на газовой плите. Более того, результаты нажатия любой из этих кнопок полностью зависят от текущего состояния микроволновки и того, какие кнопки были нажаты до этого. Вот что мы имеем в виду, когда говорим о *сложном поведении*.

Появление продуктов с таким сложным поведением стало причиной рождения новой дисциплины. Проектирование взаимодействия перенимает теорию и методы из таких областей, как традиционный дизайн, юзабилити и инженерные дисциплины. Но в данном случае целое больше суммы частей и обладает собственными уникальными ме-

тодами и подходами. И, скажем сразу, это в большой степени область именно дизайна, которая сильно отличается от научных и инженерных дисциплин. При неизменной взвешенности и рациональности подхода проектирование взаимодействия не обязательно имеет дело с текущим положением вещей – оно связано с придумыванием и синтезом того, каким это положение может и должно быть.

Помимо прочего, проектирование взаимодействия по сути своей – еще и гуманистическая затея. Эта дисциплина направлена в первую очередь на удовлетворение потребностей и желаний людей, имеющих дело с продуктом или услугой. В данной книге мы описываем конкретный подход к проектированию взаимодействия, который мы назвали целеориентированным методом. Мы обнаружили, что когда проектировщик заостряет свое внимание на целях людей – в первую очередь на побудительных мотивах использования продукта, – а также на их ожиданиях, наклонностях и отношении к окружению, возникают решения, которые люди находят мощными и приятными.

Как может заметить даже самый неискушенный наблюдатель, развитие технологий способно крайне быстро приводить к серьезному усложнению интерактивных продуктов. В то время как у механического устройства ясно различимых состояний может быть десяток, цифровой продукт способен находиться в тысячах (а то и большем числе!) различных состояний. Эта сложность может превратиться в кошмар как для пользователей, так и для проектировщиков. Чтобы обуздать ее, мы полагаемся на системный рациональный подход. Это не означает, что мы не ценим и не поощряем изобретательность и творчество. Напротив, оказывается, что методичность помогает нам замечать благоприятные обстоятельства для инновационных идей и предоставляет способ оценки их эффективности.

Согласно гештальт-психологии, человек воспринимает предмет не как набор отдельных свойств и функций, а как единое целое, неразрывно связанное с окружением. Вследствие этого невозможно эффективно спроектировать интерактивный продукт, разобрав его на атомы требований и создав отдельное решение по каждому из требований. Даже относительно простой продукт следует рассматривать цельно и в контексте окружающего его мира. И здесь мы вновь обнаруживаем, что систематический подход помогает получать целостную перспективу, столь необходимую для создания продуктов, которые люди сочтут полезными и привлекательными.

Краткая история проектирования взаимодействия

В конце семидесятых – начале восьмидесятых годов группа увлеченных исследователей, инженеров и дизайнеров в Сан-Франциско занималась изучением того, как люди могли бы взаимодействовать с компь-

ютерами в будущем. В Xerox PARC SRI, а потом и в Apple Computer вовсю обсуждали, в чем состоит создание полезных и удобных **«гуманных интерфейсов»** в применении к цифровым продуктам. В середине восьмидесятых промышленные дизайнеры Билл Могридж (Bill Moggridge) и Билл Верпланк (Bill Verplank), работавшие над первым ноутбуком (GRiD Compass), предложили для описания своей работы термин **проектирование взаимодействия**, однако понадобилось еще десятилетие, чтобы другие проектировщики заново сформулировали это понятие и сделали его общепринятым.

Когда в августе 1995 года увидело свет первое издание книги «About Face», ландшафт проектирования взаимодействия своей неорганизованностью напоминал «дикие прерии». Маленький отряд храбрецов, отважившихся носить звание «Проектировщик пользовательских интерфейсов» (user interface designer), предпринимал вылазки под прикрытием разработчиков программного обеспечения, словно кучка крошечных сообразительных млекопитающих, суетящихся в тени неповоротливых динозавров. Мало кто понимал, что представляет собой и какое имеет значение то «проектирование программного обеспечения», о котором мы говорили в первом издании; если им кто-либо и занимался, то это обычно оказывались программисты. Лишь горстка обеспокоенных технических писателей, преподавателей, специалистов служб технической поддержки, а также постепенно растущая группа профессионалов из другой зарождающейся отрасли – юзабилити – осознавали необходимость перемен.

Эти перемены произошли молниеносно благодаря всплеску популярности и лавинообразному росту Интернета. Вдруг у всех на устах оказался термин **«простота использования»**. Традиционные дизайнеры, ваявшие цифровые продукты в краткую эпоху популярности мультимедиа в начале девяностых, дружно перебрались во Всемирную паутину. Как грибы после дождя стали появляться новые, как казалось, названия профессий, связанных с дизайном и проектированием: информационные дизайнеры (information designer), информационные архитекторы (information architect), специалисты по проектированию опыта взаимодействия (user experience strategist) и проектировщики взаимодействия (interaction designer). Впервые возникли руководящие должности, связанные с созданием ориентированных на пользователей продуктов и услуг, например директор по опыту взаимодействия (chief experience officer). Университеты быстро подхватили идею и начали предлагать программы подготовки специалистов по этим дисциплинам. Тем временем юзабилити-специалисты и профессионалы, имеющие дело с человеческим фактором, также получили право голоса и стали признанными борцами за качественное проектирование продуктов.

Хотя Всемирная паутина отшвырнула инструментарий проектирования взаимодействия более чем на десятилетие в прошлое, она, бесспор-

но, совершила благое дело, поместив требования пользователей в поле зрения корпораций. Со времени выхода второго издания «About Face» в 2003 году *опыт взаимодействия* при общении с цифровыми продуктами стал темой первых полос таких изданий, как журналы *Time* и *BusinessWeek*, а такие организации, как Harvard Business School и Стэнфордский университет, признали, что следующее поколение управленцев и технологов должно находить проектированию взаимодействия место в своих бизнес-планах и графиках разработок – и это следует учитывать в программах их подготовки. Люди устали от «технологии ради технологии». Потребители посылают недвусмысленное сообщение: им нужна *хорошая* технология, то есть такая, опыт взаимодействия с которой будет подкупающе простым и приятным.

В августе 2003 года, через пять месяцев после того, как во втором издании «About Face» было провозглашено существование новой дисциплины проектирования – *проектирования взаимодействия*, – Брюс «Тог» Тогнаццини (Bruce «Tog» Tognazzini) обратился к зарождающемуся сообществу со страстным призывом, предложив создать некоммерческую профессиональную организацию. Короткое время спустя Чаллис Ходж (Challis Hodge), Дэвид Хеллер (David Heller), Рик Сесил (Rick Cecil) и Джим Джаррет (Jim Jarrett) создали список рассылки и организовали руководящий комитет. В сентябре 2005 года официально родилась организация IxDA – Interaction Design Association (ассоциация проектирования взаимодействия, *www.ixda.org*). На момент подготовки этой книги в организации уже состояло более 2000 членов из 20 с лишним стран. Нам приятно заявить, что проектирование взаимодействия наконец становится самостоятельной дисциплиной и профессией.

Почему «проектирование взаимодействия»?

В первом издании книги мы описывали дисциплину дизайна программного обеспечения и отождествляли ее с другой дисциплиной – проектированием пользовательских интерфейсов. Из двух названий более живучим оказалось, конечно, «проектирование пользовательских интерфейсов». Мы по-прежнему будем использовать его здесь в тех случаях, когда это уместно, в частности для обозначения расположения элементов интерфейса на экране. Однако в этой книге говорится о дисциплине более широкой, нежели проектирование пользовательских интерфейсов. В мире цифровых технологий столь тесно переплетены форма, функции, содержание и поведение, что, отвечая на вызовы, которые бросает нам проектирование интерактивных продуктов, мы сплошь и рядом затрагиваем самую суть того, чем продукт *является* и что он *делает*.

Как мы уже говорили, проектировщики взаимодействия, заимствуя подходы из других, уже сложившихся дисциплин проектирования,

выходят за пределы этих подходов. Одно время к проектированию цифровых продуктов пытались обратиться промышленные дизайнеры, однако, как и их коллеги из области графического дизайна, они обычно фокусировались на проектировании статической формы вместо интерактивной, то есть такой, которая реагировала бы на внешние факторы и менялась со временем. В этих дисциплинах отсутствует язык, который позволил бы обсуждать проектирование динамичных, развитых моделей поведения и изменяющихся пользовательских интерфейсов.

В последние годы для этой разновидности проектирования предлагались различные названия. С распространением Всемирной паутины появилась *информационная архитектура* (*information architecture*) – дисциплина, посвященная решению вопросов, связанных с навигацией и «находимостью» содержимого – преимущественно (хотя и не исключительно) в контексте веб-сайтов. Будучи очевидно близкой к проектированию взаимодействия, информационная архитектура по большей части сохранила свой узкий, ориентированный на веб-окружение подход к организации содержимого и навигации, использующий страницы, ссылки и в минимальной степени интерактивные интерфейсные элементы. Однако свежие веяния в этой области, такие как Web 2.0 и полнофункциональные интернет-приложения, вывели веб-дизайнеров из оцепенения, подтолкнув их к поиску за пределами архаичных идиом взаимодействия с броузером. Мы считаем, что это пробуждение еще сильнее сближает информационных архитекторов с проектировщиками взаимодействия.

Еще один термин, который приобрел популярность, – *опыт взаимодействия* (*user experience*, UX). Многие люди выступают за использование этого термина в качестве «зонтика», под которым объединяется множество различных дисциплин, связанных с проектированием и удобством использования продуктов, систем и услуг. Достойная и заманчивая цель, которая, однако, слабо связана с ключевой проблемой проектирования взаимодействия, обсуждаемой в этой книге: как конкретно следует проектировать поведение сложных интерактивных систем? И хотя поиск сходств и путей взаимного обогащения подходов в проектировании опыта взаимодействия для покупателя в реальном магазине и для пользователя интерактивного продукта – полезное занятие, мы считаем, что существуют особые методы, уместные именно для проектирования в цифровом мире.

А возможно ли действительно *спроектировать* опыт взаимодействия? Проектировщики всех мастей лелеют надежду управлять опытом людей и *влиять* на него, однако в качестве инструмента выступает всего лишь аккуратное манипулирование свойствами, присущими используемой среде. Дизайнер, работающий над плакатом, определяет опыт взаимодействия при помощи сочетания текста, фотографий и иллюстраций; дизайнер мебели, работающий над креслом, создает опыт по-

средством материалов и инженерных методов; дизайнер интерьеров влияет на опыт расстановкой, освещением, подбором материалов и даже звуком.

При распространении этого подхода на цифровые продукты оказывается полезным считать, что мы воздействуем на опыт пользователей, проектируя механизмы взаимодействия с продуктом. Поэтому мы предпочли термин Могриджа **проектирование взаимодействия** (многими сокращаемое до IxD – от interaction design) для обозначения вида проектирования, описываемого в настоящей книге.

Разумеется, во многих случаях при создании продукта требуется слаженное применение целого ряда дисциплин проектирования, чтобы дать пользователю позитивный опыт (рис. 1). Мы считаем, что именно для таких ситуаций уместен термин *проектирование опыта взаимодействия*.

Рис. 1. *Можно считать, что в проектировании опыта взаимодействия (UX) для цифровых продуктов сочетаются три пересекающиеся темы: форма, поведение, наполнение. Проектирование взаимодействия сосредоточено на проектировании поведения, но помимо этого ставит вопрос о том, как это поведение связано с формой и наполнением. В свою очередь, информационная архитектура занимается структурированием наполнения, но ставит вопрос о том, какие поведенческие модели обеспечивают доступ к этому наполнению и каким образом представлено наполнение. Промышленный дизайн и графический дизайн отвечают за форму продуктов и услуг, но помимо этого должны гарантировать, что форма поддерживает модель использования, а это требует внимания к поведению и наполнению*

Сотрудничество с командой, создающей продукт

Помимо определения проектирования взаимодействия в терминах решаемых задач и отношений с другими дисциплинами проектирования, часто оказывается необходимым найти этой дисциплине подходящее место внутри организации. Мы считаем, что построение четкого процесса разработки, в котором проектирование имеет равные права с разработкой, маркетингом и управлением бизнесом, а зоны ответственности каждой из групп строго определены, значительно увеличивает те выгоды, которые бизнес может извлечь из проектирования. Описанное ниже распределение ответственности, уравновешенное соответствующим распределением полномочий, позволяет значительно повысить шансы продукта на успех и обеспечить организационную поддержку продукта как на протяжении всего цикла разработки, так и по его окончании.

- Команда **проектировщиков** отвечает за степень удовлетворенности пользователей процессом взаимодействия с продуктом. На текущий момент в большинстве организаций за это никто не отвечает. Чтобы нести такую ответственность, проектировщики должны иметь возможность определять внешний вид продукта, его поведение и создаваемое им впечатление. Кроме того, им понадобится доступ к информации: они должны наблюдать за потенциальными пользователями продукта, обсуждать с пользователями их потребности, с инженерами – технологические возможности и ограничения, с маркетологами – возможности и требования, а с руководством – что из себя должен представлять конечный продукт.
- Команда **инженеров** отвечает за реализацию и производство продукта. Чтобы проектирование принесло пользу, инженеры должны нести ответственность за воспроизведение формы и поведения продукта в том виде, как это *задано проектировщиками*, уложившись при этом в бюджет и выдержав сроки. Следовательно, инженерам требуется ясное описание формы и поведения продукта, которое будет направлять их работу и послужит точкой отсчета при оценке затрат времени и средств. Такое описание должно поступать от команды проектировщиков. Инженеры должны также участвовать в обсуждениях технических возможностей и ограничений и оценке выполнимости предложенных проектировщиками решений.
- Команда **маркетологов** несет ответственность за желание клиентов заказать продукт, а потому в их ведении должны находиться все аспекты общения с клиентами. У них должна быть также возможность повлиять на характеристики и дизайн продукта. Для этого участники маркетинговой команды должны иметь доступ к информации, включая результаты исследований проектировщиков, и располагать возможностью проводить собственные исследования. (Следует отметить, что клиенты и пользователи – зачастую совсем разные люди с разными потребностями. Мы обсудим эту тему более подробно в главах 4 и 5.)

- **Руководство** отвечает за прибыльность полученного продукта, а потому имеет полномочия принимать решения о том, над чем следует работать перечисленным выше группам. Чтобы принимать такие решения, руководство должно получать внятные сведения от других групп: результаты исследований проектировщиков и характеристики продукта, результаты маркетинговых исследований и прогнозы продаж, оценки инженеров относительно времени и затрат, необходимых для создания продукта.

Что есть и чего нет в этой книге

Мы попытались дать читателям эффективные, практичные инструменты для проектирования взаимодействия. Наш набор инструментов состоит из *принципов*, *шаблонов* и *процессов*. В *принципах* проектирования сформулированы общие идеи о практике проектирования, а также правила и советы относительно наилучшего применения тех или иных идиом взаимодействия и пользовательского интерфейса. *Шаблоны* проектирования описывают такие наборы идиом взаимодействия, которые регулярно применяются для реализации определенных пользовательских требований и решения типичных проблем проектирования. *Процессы* проектирования определяют схему, позволяющую понять и описать требования пользователей, преобразовать эти требования в общую структуру проекта и, наконец, найти лучший способ применения принципов и шаблонов проектирования в конкретных ситуациях.

Существуют книги, посвященные принципам и шаблонам проектирования; книг, посвященных процессам проектирования, гораздо меньше; и совсем мало книг, повествующих одновременно обо всех этих инструментах и их совместном применении для целей качественного проектирования. При написании этой книги нашей целью было сведение трех инструментов воедино. Помогая вам проектировать более эффективные и полезные диалоговые окна и меню, эта книга одновременно поможет понять, как пользователи воспринимают ваш цифровой продукт и взаимодействуют с ним и как использовать это понимание в качестве отправной точки проектирования.

Интеграция принципов, процессов и шаблонов проектирования – ключ к созданию эффективного взаимодействия и эффективных интерфейсов цифровых продуктов. Объективно хорошего пользовательского интерфейса попросту не существует – качество зависит от контекста: кем является пользователь, что он делает, каковы его мотивы. Подход «универсальный интерфейс для всех» *упрощает* создание пользовательских интерфейсов, однако не обязательно *улучшает* качество результата. Если вы хотите добиться качественных решений в проектировании, вам не избежать тяжелой работы по изучению людей, которым адресован продукт. И вот тогда оказывается полезным иметь в распоряжении набор принципов и шаблонов, применимых в конкретных

ситуациях. Мы надеемся, что эта книга не только побудит вас лучше понять пользователей своего продукта, но и научит создавать высококлассные продукты, опираясь на это понимание.

Эта книга *не* предназначена для того, чтобы быть руководством по стилю или стандартом по интерфейсам. Более того, в главе 14 вы узнаете, почему применимость подобных инструментов ограничена весьма узким набором ситуаций. Тем не менее мы надеемся, что процессы и принципы, описанные в этой книге, станут полезным дополнением к используемым вами руководствам по стилю. Такие руководства хорошо отвечают на вопрос *«что?»*, однако редко способны ответить на вопрос *«почему?»*. Эта книга пытается дать ответы на вопросы второго типа.

Мы рассмотрим четыре основные стадии проектирования интерактивных систем: исследование предметной области, анализ пользователей и их требований, определение структуры системы, детализация интерфейсных решений. Многие практики добавили бы пятую стадию – *проверку* эффективности решений на реальных пользователях. Это часть дисциплины, широко известной как *юзабилити*. Будучи важной и результативной составляющей многих начинаний в области проектирования взаимодействия, эта дисциплина, однако, вполне самостоятельна. Проверку проектирования и юзабилити-тестирование мы бегло обсудим в главе 7, однако настоятельно рекомендуем читателям обратиться к постоянно растущему списку книг по юзабилити за более подробной информацией о проведении юзабилити-тестов и анализе их результатов.

Новое в третьем издании

Многое изменилось в мире проектирования интерфейсов с тех пор, как в 1995 году увидело свет первое издание этой книги. В то же время многое осталось прежним. В третьем издании мы сохранили все, что не потеряло актуальности, обновили то, что с тех пор изменилось, и включили новые материалы, которые отражают не только перемены в отрасли за последние одиннадцать лет, но и появление новых концепций, созданных авторами с учетом меняющихся требований времени. Вот некоторые серьезные изменения в «About Face 3.0»:

- Структура книги полностью переработана с тем, чтобы подать идеи в более удобной справочной форме. Книга состоит из трех разделов: первый посвящен процессу и общим идеям о пользователях и проектировании, второй – общим принципам проектирования взаимодействия, третий описывает низкоуровневые принципы проектирования.
- Первая часть гораздо более подробно, чем во втором издании, описывает процесс целеориентированного проектирования, и теперь книга более точно отражает сложившуюся в компании Cooper прак-

тику, включая методики исследований, создание персонажей и применение персонажей и сценариев для синтезирования решений в области проектирования взаимодействия.

- На протяжении всей книги мы старались более явно и наглядно рассказывать о концепциях, методах и проблемах проектирования визуальной части пользовательских интерфейсов, а также о задачах, возникающих за пределами настольных компьютеров.
- Мы обновили терминологию и примеры сообразно состоянию дел в отрасли, а текст в целом подвергся значительным правкам, стал более понятным и легкочитаемым.

Надеемся, что благодаря этим дополнениям и изменениям читатели смогут по-новому взглянуть на рассмотренные темы.

О примерах

Вы держите в руках книгу о проектировании самых разнообразных цифровых интерактивных продуктов. Но поскольку проектирование взаимодействия уходит корнями в программное обеспечение для настольных компьютеров, а подавляющее большинство современных персональных компьютеров работают под управлением Microsoft Windows, наш текст определенно имеет перекос в сторону платформы, на которой потребность в эффективных целеориентированных пользовательских интерфейсах особенно высока.

Сделав эту оговорку, отметим, что основной материал книги не зависит от используемой платформы и в равной степени полезен на всех платформах для персональных компьютеров – Mac OS, Linux и прочих, а большая его часть остается применимой и на таких специфических платформах, как киоски, наладонные устройства, встроенные системы и т. п.

Большинство примеров в этой книге взято из программ пакета Microsoft Office – Word, Excel, PowerPoint, Outlook, а также из Internet Explorer, Adobe Photoshop и Illustrator. Есть две причины, по которым мы старались использовать в качестве источника примеров эти популярные программы. Во-первых, в таком случае примеры будут, вероятно, хотя бы в минимальной степени знакомы читателям. Во-вторых, важно было продемонстрировать, что целеориентированное проектирование позволяет улучшить дизайн пользовательского интерфейса даже самых отточенных продуктов. Мы включили в книгу также некоторое количество примеров из менее распространенных приложений – там, где они были особенно показательны.

Ряд примеров позаимствован из ныне устаревших программ или операционных систем – мы решили сохранить их в новой версии книги, поскольку они хорошо иллюстрируют определенные особенности проектирования. Однако подавляющее большинство примеров взяты из современных программных продуктов и операционных систем.

Для кого эта книга

Хотя содержание книги в основном ориентировано на тех, кто занимается проектированием взаимодействия на практике либо обучается этой дисциплине, пользу от чтения книги получат все, кого заботит взаимодействие пользователей с цифровыми технологиями. Программисты, дизайнеры и проектировщики, работающие над созданием цифровых продуктов, юзабилити-специалисты, а также руководители проектов – все они найдут что-либо полезное для себя. Читатели предыдущих изданий книги «About Face» и «The Inmates Are Running the Asylum»[1] найдут свежую и обновленную информацию о методах и принципах проектирования.

Надеемся, эта книга заинтригует вас и обогатит знаниями, но самое главное – надеемся, что она заставит вас по-новому думать о проектировании цифровых продуктов. Практика проектирования взаимодействия активно развивается, а сама дисциплина настолько молода и изменчива, что порождает огромное разнообразие мнений и мыслей. Если у вас есть интересное мнение или вы просто хотите пообщаться с нами, мы будем рады получить ваши письма по адресам *alan@cooper.com*, *rmreimann@gmail.com* и *dave@cooper.com*.

От редакторов перевода

Вы держите в руках особенную книгу. Для проектировщиков интерфейсов она является тем же, чем для венецианских кораблестроителей XIV века была «Arte de far vasselli» («Искусство судостроения»), с той лишь разницей, что Алан Купер и его команда не делают из своих открытий государственного секрета, и поэтому вы можете спокойно листать эти страницы при дневном свете, не пугаясь собственной тени.

Проектирование взаимодействия – бурно развивающаяся дисциплина. Перемены в ней происходят порой быстрее, чем в земной атмосфере. Терминология (особенно русская) не только не устоялась, но иногда просто-напросто отсутствует. При работе над переводом мы особенно остро ощущали и ураганную скорость перемен, и неизбежную на старте любой науки бедность лексикона, и колоссальную значимость этой книги. Все это привело нас к идее параллельно с подготовкой бумажного издания создать интернет-ресурс, где можно было бы разме-

[1] А. Купер «Психбольница в руках пациентов. Почему высокие технологии сводят нас с ума и как восстановить душевное равновесие», дополненное издание. – Пер. с англ. – СПб: Символ-Плюс, 2009.

щать уточнения к книге и новости мира проектировщиков, обсуждать термины, до хрипоты спорить о принципах... Мы сделали это и приглашаем вас в гости. Наш адрес *http://aboutface.gui.ru*.

Вселенная проектирования взаимодействия еще так молода, что вы можете успеть в ряды ее демиургов!

I

Введение в целеориентированное проектирование

Глава 1. Проектирование, ориентированное на цели
Глава 2. Модели реализации и ментальные модели
Глава 3. Новички, эксперты и середняки
Глава 4. Как понять пользователей: качественные исследования
Глава 5. Модели пользователей: персонажи и цели
Глава 6. Основы проектирования: сценарии и требования
Глава 7. От требований к пользовательскому интерфейсу: общая инфраструктура и детализация

1

Проектирование, ориентированное на цели

У нашей книги простая предпосылка: если мы проектируем и конструируем продукт таким образом, чтобы использующие его люди достигали своих целей, эти люди будут продуктивны, довольны и счастливы – и тогда они с радостью заплатят за наш продукт и посоветуют его другим. Если при этом мы сможем окупить свои затраты, наш бизнес станет успешным.

На первый взгляд эта предпосылка довольно проста и очевидна: осчастливьте людей – и ваши продукты станут успешными. Тогда почему столь многие цифровые продукты вызывают неприятие и сложности в использовании? Где всеобщее счастье и повальный успех?

Цифровым продуктам необходимы более качественные методы проектирования

В большинстве своем цифровые продукты *возникают* в процессе разработки – как монстр из булькающей протоплазмы в резервуаре. Разработчики, вместо того чтобы планировать и действовать исходя из нужд тех пользователей, которые приобретут и будут использовать созданные продукты, сплошь и рядом сосредотачиваются на технологии и в результате порождают решения, слабоуправляемые и неудобные в применении. Словно безумные ученые они терпят поражение, потому что не наделяют свои создания человечностью.

Проектирование, по мнению промышленного дизайнера Виктора Папанека (Victor Papanek), есть *сознательные и интуитивные усилия*

по созданию значимого порядка.[1] Мы предлагаем несколько более подробное определение проектирования, ориентированного на людей:

- понимание желаний, потребностей, мотивации пользователей и контекста, в котором эти пользователи находятся;
- понимание возможностей, требований и ограничений бизнеса, технологии и предметной области;
- использование этих знаний в качестве основы всех планов по созданию продуктов, форма, содержание и поведение которых делают их полезными, удобными и желанными, а также экономически жизнеспособными и технически осуществимыми.

Это определение хорошо подходит для многих дисциплин проектирования, хотя конкретные акценты на форме, содержании или поведении будут зависеть от того, что именно мы проектируем. Так, в случае информационного веб-сайта может потребоваться особенное внимание к *содержанию*, тогда как при проектировании кресла важна главным образом *форма*. Как мы уже говорили во «Введении», уникальность интерактивных цифровых продуктов – в их сложном *поведении*.

Если проектирование осуществляется подходящими методами, оно способно восстановить отсутствующую связь человека с технологическими продуктами. Очевидно, однако, что большинство существующих подходов к проектированию цифровых продуктов не дают обещанного эффекта.

Современное проектирование цифровых продуктов

Цифровые продукты рождаются в результате перетягивания каната двумя нередко враждующими силами – разработчиками и маркетологами. Маркетологи, конечно же, хорошо замечают и оценивают возможности по выходу на рынок, способны представить и позиционировать продукт, однако их вклад в процесс проектирования продукта часто ограничивается списком требований, который передается разработчикам. Эти требования зачастую далеки от реальных *потребностей* и *желаний* пользователей и имеют гораздо большее отношение к борьбе с конкурентами, к управлению ИТ-ресурсами посредством перечня задач и к предположениям, которые построены на основе исследований рынка, представляющих информацию о том, что люди *обещают* покупать. (Вопреки интуитивному мнению, мало кто способен четко выразить свои потребности. Столкнувшись с прямыми вопросами относительно используемых продуктов, большинство людей сосредотачиваются на незначительных задачах или способах борьбы с изъянами продуктов.) К сожалению, перечень из сотен функций, к которому сводится интерактивный продукт, слабо пригоден для создания той

[1] В. Папанек «Дизайн для реального мира». – Пер. с англ. – М.: Д. Аронов, 2008.

тонкой гармонии, которая делает сложную технологию полезной. Добавление в перечень требований фразы «должен быть простым в использовании» никоим образом не улучшает ситуацию.

С другой стороны, участие разработчиков в определении окончательной формы и поведения продукта обычно ничем не ограничено. Поскольку за строительство отвечают они, они же решают, что именно строить. И их набор представлений также отличается от набора представлений конечных пользователей продукта. Хорошие разработчики концентрируются на том, чтобы найти решение непростых технических проблем, применить подходящие инженерные методы и сдать продукт вовремя. Часто они получают неполные, беспорядочные, а порой даже противоречивые инструкции и вынуждены в условиях недостатка времени или информации принимать важные решения относительно того, каким будет опыт пользователей.

Вот как случается, что люди, обычно ответственные за создание наших цифровых продуктов, редко принимают во внимание *цели*, потребности или же мотивы пользователей – и в то же время крайне восприимчивы к новым рыночным тенденциям и техническим возможностям. Этого трудно избежать, но в результате на выходе появляется продукция, лишающая пользователей целостного опыта. Вскоре мы увидим, почему цели столь важны для решения этой проблемы.

К сожалению, результатом такого подхода становятся цифровые продукты, которые раздражают пользователя, снижают его производительность и не отвечают его потребностям. На рис. 1.1 представлена эволюция процесса разработки программного обеспечения и указано то место, которое в этом процессе отводится (когда отводится вообще) проектированию. Разработка большинства цифровых продуктов застыла на первом, втором или третьем шаге этой эволюции, где проектирование либо не играет роли, либо становится косметической заплаткой поверх некачественного взаимодействия – «макияжем для свиньи»[1], как выразился один из наших клиентов. Процесс проектирования, как мы скоро увидим, должен *предшествовать* созданию кода и тестированию, иначе нельзя гарантировать, что продукт действительно будет соответствовать потребностям пользователя.

За десятилетие, прошедшее с момента выхода первого издания этой книги, качество программ и интерактивных продуктов несомненно улучшилось. Многие компании начали уделять пристальное внимание тому, чтобы их продукция удовлетворяла нужды людей, и стали тратить время и деньги на раннее проектирование. Однако гораздо большему количеству компаний все еще не удалось включиться в эту игру, и до тех пор, пока они сосредоточены на технологии и рыночных

[1] Имеется в виду английская пословица You can put lipstick on a pig, but it's still a pig (Свинья есть свинья, сколько ее ни крась). – *Примеч. перев.*

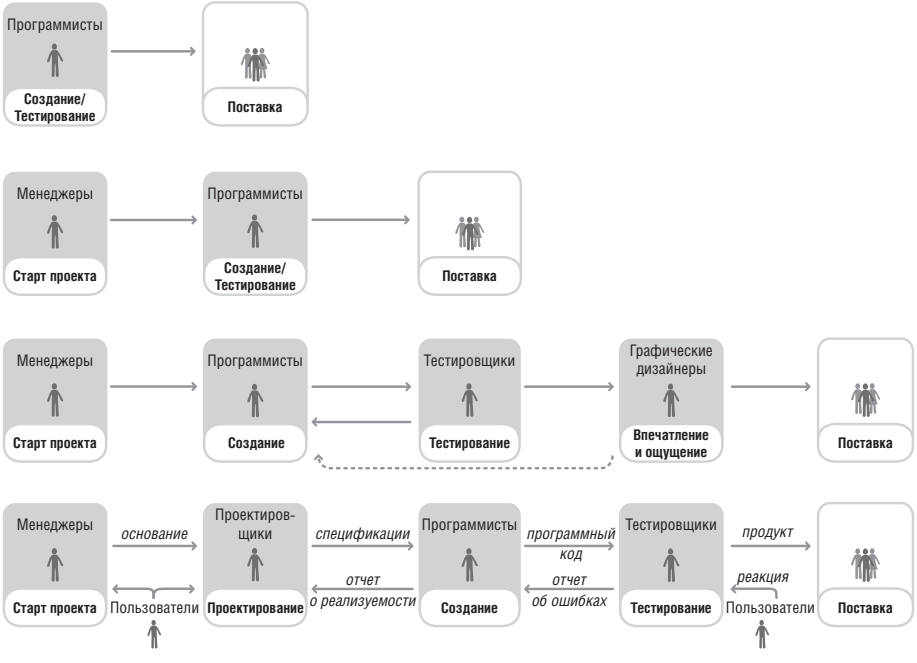


Рис. 1.1. *Эволюция процесса разработки программного обеспечения. На первой диаграмме отражены ранние дни индустрии программного обеспечения, когда умные программисты вынашивали идею продукта, а затем создавали и самостоятельно тестировали его. Разумеется, на каком-то этапе в процесс встроились профессиональные управленцы, которые помогали переводить благоприятные рыночные возможности на язык требований к продукту. Как видно из третьей диаграммы, индустрия повзрослела, и тестирование превратилось в самостоятельную дисциплину, а с распространением графических пользовательских интерфейсов (graphical user interface, GUI) к процессу подключились графические дизайнеры, которые создавали пиктограммы и прочие визуальные элементы. На последней диаграмме отражен целеориентированный подход к разработке программного обеспечения, когда решения о возможностях продукта, его форме и поведении принимаются до начала дорогостоящей и сложной фазы создания продукта*

данных, они будут продолжать создавать цифровые продукты, вызывающие у нас презрение. Вот некоторые симптомы этого недуга.

Цифровые продукты грубы

Программы часто обвиняют пользователя в ошибках, в которых нет (или не должно быть) его вины. Сообщения об ошибках вроде приведенного на рис. 1.2 выскакивают, словно черт из табакерки, чтобы еще и еще раз возвестить пользователю о его промахе. Вдобавок эти сообщения требуют, чтобы пользователь непременно согласился со своим провалом, щелкнув по кнопке OK.

Внимание! Не удалось уведомить библиотеку.

Рис. 1.2. *Замечательно, спасибо за откровенность. Почему программа не уведомила библиотеку? О чем она хотела уведомить эту библиотеку? Почему она говорит об этом нам? С чем мы вообще соглашаемся? С какой стати сбой в программе – это «OK»?*

Программы часто допрашивают пользователя, засыпая его маловразумительными вопросами, на которые пользователь не готов или не склонен отвечать: «Куда ты подевал этот файл?» Снисходительные вопросы, вроде «Вы уверены?» и «Вы действительно хотите удалить этот файл или нажали на клавишу Delete по другой причине?», равно унизительны и неприятны.

Кроме этого, наши содержащие программный код продукты не способны продемонстрировать достаточную степень услужливости. Они забывают информацию, которую мы им сообщаем, и плохо предугадывают наши потребности. К примеру, богатый функциями смартфон Palm Treo не способен предвидеть, что у пользователя может появиться желание добавить телефонный номер того, кто только что позвонил, в *существующую* запись телефонной книжки. Чтобы догадаться, что такая потребность возникнет у многих пользователей, не нужны ни обширные исследования, ни богатое воображение; однако нам приходится пробираться через сложнейший лабиринт – копировать телефонный номер, открывать нужную запись телефонной книжки, а затем вставлять номер в соответствующее поле.

Цифровые продукты навязывают людям компьютерный стиль мышления

Цифровые продукты сплошь и рядом подразумевают, что пользователь технически грамотен. К примеру, пользователь Microsoft Word, желающий переименовать текущий документ, должен знать, что потребуется либо закрыть файл, либо воспользоваться командой меню Сохранить как... (не забыв при этом удалить файл с прежним именем). Такое поведение не согласуется с тем, как обычный человек представляет себе переименование чего бы то ни было; напротив, подразумевается, что человек должен начать мыслить в логике, соответствующей принципам работы компьютера.

Программы часто невразумительны, они скрывают от пользователя смысл происходящего, свои намерения и действия. Программы нередко изъясняются на жаргоне, непостижимом для нормальных пользо-

вателей («Сколько стоповых битов?»), а иногда и для специалистов («Пожалуйста, укажите IRQ»).

Цифровые продукты ведут себя неподобающим образом

Если бы десятилетний ребенок вел себя так же, как некоторые программы или устройства, его бы заперли в детской и оставили без ужина. Программы забывают закрывать за собой дверь холодильника, оставляют ботинки на ковре посреди комнаты и не способны вспомнить то, что вы говорили им каких-то пять минут назад. К примеру, если вы сохраните документ Microsoft Word, распечатаете его, а затем попытаетесь закрыть, программа снова спросит, желаете ли вы сохранить документ! Очевидно, печать документа заставила программу думать, что документ изменился, хотя этого не произошло. «Не, мам, я тебя не слышал!»

Программы часто заставляют нас отступать от основной последовательности задач, чтобы «дотянуться» до функции, которая должна быть под рукой. В то же время опасные команды часто оказываются на самом видном месте – чтобы ничего не подозревающему пользователю было удобно случайно ими воспользоваться. В целом многие программы выглядят чрезмерно сложными и запутанными, что осложняет навигацию и затрудняет понимание происходящего.

Цифровые продукты заставляют человека делать рутинную работу

Компьютеры и их сородичи на основе микрочипов призваны облегчать человеческий труд. Однако наблюдения за реальным людьми, выполняющими свою работу при помощи технологии, всякий раз вызывают шок и трепет: эти люди вынуждены делать колоссальную работу только лишь для того, чтобы справляться с программами. Это может быть любая работа – от ручного перепечатывания значений из одного окна в другое до копирования данных между приложениями, не знающими иных способов общения друг с другом, или встречающейся буквально на каждом шагу перетасовки окон с целью добраться до постоянно нужных в работе функций.

Почему эти продукты столь плохи?

Так в чем же реальная проблема? Почему индустрия технологий оказывается в целом недееспособной, когда требуется продумать интерактивную составляющую цифровых продуктов? Тому есть три основных причины: отсутствие представления о пользователях, конфликт между потребностями людей и приоритетами разработки, а также отсутствие процесса, позволяющего понимать потребности человека и помогающего в разработке удобной формы и качественного поведения продукта.

Отсутствие представления о пользователях

Печальная истина состоит в том, что индустрия цифровых технологий не очень хорошо понимает, как сделать пользователей счастливыми. Технологичные продукты в большинстве своем создаются без достаточного представления о пользователях. Мы можем знать, в каком *сегменте рынка* находятся наши пользователи, сколько денег они зарабатывают, сколько позволяют себе тратить по выходным и какого рода автомобили приобретают. Возможно, мы даже имеем смутное представление о работе, которую они делают, и о некоторых основных часто выполняемых задачах. Но позволяет ли нам что-либо из вышеперечисленного понять, как осчастливить пользователей? Позволяет ли узнать, *как* они в действительности будут применять продукт, который мы создаем? Позволяет ли выяснить, *почему* они занимаются той деятельностью, в которой им может помочь наш продукт, *почему* они могут отдать предпочтение именно нашему продукту, а не продукту конкурента, и *как* нам добиться этого? К сожалению, ответ отрицательный.

Скоро мы увидим, как можно достичь понимания пользователей и их поведения в контексте создания продуктов.

Конфликт интересов

Вот вторая проблема, влияющая на способность продавцов и производителей делать пользователей счастливыми. В мире разработки цифровых продуктов существует серьезный конфликт интересов: проектируют продукты, как правило, те же люди, которые эти продукты разрабатывают, – программисты. Программистам часто приходится выбирать между простотой создания кода и простотой использования продукта. Поскольку о производительности программистов обычно судят по их способности эффективно писать код и сдавать его в невероятно сжатые сроки, несложно понять, в какую сторону склоняются весы для большинства цифровых продуктов. В судебном процессе мы никогда не позволим обвинителю выносить приговор – и точно так же должны убедиться, что проектируют продукт не те же люди, которые занимаются его разработкой. Даже обладая необходимыми навыками и демонстрируя лучшие намерения, программист попросту не в силах выступать на стороне пользователя, бизнеса и технологии одновременно.

Отсутствие процесса

Третья причина того, что индустрия цифровых технологий не выпекает успешные продукты словно пирожки, – у нее нет подходящего надежного *процесса*. Или, говоря точнее, нет *полноценного процесса*. Инженерные команды следуют – или должны следовать – строгим инженерным методам, обеспечивающим *осуществимость* и технологическое качество. Точно так же маркетологи, отделы продаж и прочие бизнес-подразделения следуют собственным устоявшимся методам обеспечения

коммерческой *жизнеспособности* новых продуктов. Чего не наблюдается, так это воспроизводимого предсказуемого аналитического процесса для *преобразования представления о пользователях в продукты, одновременно удовлетворяющие потребности пользователей и вызывающие живой отклик*.

Если речь идет о сложных механических устройствах, мы принимаем как должное, что эти устройства были тщательно продуманы с позиций практического применения, а не просто сконструированы с применением инженерных методов. Объекты промышленного производства, как правило, достаточно просты, и даже сложные механические продукты оказываются простыми в сравнении с большинством программ и продуктов, содержащих программный код. Цифровые продукты могут содержать миллионы строк кода (сравните с подавляюще сложным механическим артефактом – космическим челноком, содержащим 250 000 деталей, лишь малый процент которых составляют детали *подвижные*). И при этом большинство программ начинают жизнь, не пройдя скрупулезного проектирования, ставящего пользователя во главу угла.

В худшем случае решения о том, что будет делать цифровой продукт и как он будет общаться с пользователем, становятся побочным результатом процесса разработки этого продукта. Программисты, погруженные в мысли о коде и алгоритмах, «проектируют» пользовательские интерфейсы таким же образом, каким шахтеры «проектируют» ландшафт, пронизывая его шахтами и загромождая отработанной породой. У пребывающих в неведении компаний-разработчиков выбор процесса проектирования взаимодействия с цифровыми продуктами небогат: между случайным и отсутствующим.

Некоторые организации осознают необходимость процесса проектирования, но принятый ими на вооружение процесс оказывается неспособен решать поставленную задачу. Многие программисты сегодня с готовностью воспринимают идею, что интеграция представителей заказчика в процесс разработки на регулярной основе – еженедельной или даже ежедневной – способна решать проблемы проектирования человеческих интерфейсов. Хотя у такого подхода есть благотворный эффект – ответственность за проектирование частично возлагается на пользователя, – однако есть и серьезный методологический изъян: владение предметной областью путается со знаниями о проектировании. Заказчики, даже если они способны четко сформулировать проблемы взаимодействия, нечасто способны представить способы решения этих проблем. Проектирование, как и программирование, – это специальный навык. Программисты никогда не просят у пользователей помощи в написании *кода*; к проблемам проектирования следует относиться так же. Кроме того, постоянными *пользователями* продукта не всегда являются *заказчики*, приобретающие его, – это тонкое, но важное различие.

Это отнюдь не означает, что проектировщики не должны интересоваться мнением других о предложенных решениях. Однако каждый участник команды, работающей над продуктом, должен уважать специализацию других участников процесса. Представьте, что больной приходит к врачу с ужасной болью в животе. «Доктор, – говорит он, – *очень* болит. Подозреваю, это аппендикс. Его нужно срочно удалить!» Разумеется, ответственный врач не станет делать такую операцию по просьбе больного. Пациент может описать симптомы, однако для постановки верного диагноза требуются профессиональные знания врача.

Чтобы понять, как строится работоспособный процесс, благодаря которому проектирование цифровых продуктов становится ориентированным на пользователей, полезно несколько углубиться в историю промышленного дизайна и посмотреть, как сложности, связанные с интерактивными продуктами, радикально изменили требования к проектированию.

Эволюция проектирования в промышленности

На заре промышленного производства инженерный и маркетинговый процессы вполне справлялись с созданием *желанных* продуктов: для разработки и производства хорошо продаваемых молотков, дизельных двигателей и тюбиков с зубной пастой нужны были лишь профессиональный инженерный подход и разумная ценовая политика. Время шло, производители потребительской продукции осознали, что необходимо как-то выделять свои продукты на фоне функционально идентичных продуктов конкурентов, – и как средство повышения привлекательности продукта для конечного пользователя возник дизайн. Графические дизайнеры привлекались для создания более эффектной упаковки и рекламы, а промышленные дизайнеры – для создания более удобных, полезных, приятных форм.

Сознательное включение проектирования в процесс создания продуктов возвестило о восхождении современной триады задач разработки, озвученных Ларри Кили (Larry Keeley) из Дублинской группы: осуществимость, жизнеспособность, желанность (рис. 1.3). Если один из этих трех столпов значительно слабее двух других, продукт вряд ли выдержит испытание временем.

Наконец, появился компьютер – первая из созданных человеком машина, способная на практически бесконечные вариации *поведения*, задаваемые соответствующим программным кодом. Что любопытно в этом сложном поведении, называемом интерактивностью, – оно полностью изменяет природу продуктов, к которым имеет отношение. Интерактивность привлекает человека – привлекает так сильно, что прочие аспекты интерактивного продукта оказываются малозначимыми. Никто не обращает внимания на черный ящик под столом – внимание пользователей приковано к интерактивному экрану, клавиатуре и мыши.

Попробуем изучить эту схему на примере компаний, пытавшихся найти баланс:

Рис. 1.3. *Создание успешных цифровых продуктов. Диаграмма отражает три основных процесса, которые должны быть тесно увязаны друг с другом, чтобы стало возможным создание успешных технологических продуктов. Эта книга посвящена первому из перечисленных и первоочередному по значению вопросу: как создать продукт, который будет желанным для пользователя*

И при этом интерактивному поведению программ и прочих цифровых продуктов, заслуживающему львиной доли внимания при проектировании, слишком часто не уделяют внимания вообще.

Традиции дизайна, которым следуют корпорации в целях обеспечения желанности создаваемых ими продуктов, не слишком помогают в мире интерактивности. Проектирование поведения – проблема иного рода, требующая более глубокого понимания *контекста*, а не просто следования правилам визуальной композиции и брендинга. Проектирование поведения требует понимания отношений пользователя с продуктом – с момента осознания потребности в продукте до момента, когда продукт становится больше не нужен. И важнее всего понимание того, каким образом пользователь желает применять продукт, какими способами и с какими целями.

Планирование и проектирование поведения

Планирование сложных, а особенно – вступающих в прямое взаимодействие с человеком цифровых продуктов требует значительных предварительных усилий со стороны профессиональных проектировщиков – так же, как планирование сложных строений в реальном мире, с которыми приходится иметь дело человеку, требует значительных предварительных усилий профессиональных архитекторов. Если говорить об архитектуре, для этого планирования необходимо понять, как живут и работают люди, использующие здание, и спроектировать пространство таким образом, чтобы поддержать и упростить реализацию их поведения. В случае цифровых продуктов для такого планирования нужно понять, как живут и работают пользователи продукта, и спроектировать поведение и форму продукта таким образом, чтобы поддержать и упростить реализацию человеческого поведения. Архитектура – почтенная, устоявшаяся область. Проектирование поведения продуктов и систем – **проектирование взаимодействия** – область довольно новая, которая лишь в последние годы начала обретать зрелость в качестве самостоятельной дисциплины.

Проектирование взаимодействия касается не столько эстетических вопросов, сколько понимания пользователей и принципов их познавательной деятельности. Это хорошая новость, поскольку она означает, что проектирование поведения вполне подвластно воспроизводимым процессам анализа и синтеза. Из этого не вытекает, что проектирование поведения поддается автоматизации (по крайней мере, не в большей степени, чем проектирование формы или содержания), – но вытекает, что систематический подход *возможен*. Разумеется, соображениями формы и эстетической привлекательности пренебрегать не следует, но они должны работать в гармоничной связке с общей заботой о достижении целей пользователя посредством правильно спроектированного поведения.

В настоящей книге представлен ряд методов для этой новой разновидности ориентированного на поведение проектирования, направленного на реализацию *целей* (Rudolf, 1998) и мотивов пользователей, – **целеориентированного проектирования.** Чтобы понять суть целеориентированного проектирования, мы должны прежде всего лучше понять цели пользователей и осознать их ключевую роль в проектировании соответствующего интерактивного поведения.

Выявление целей пользователей

Что же такое цели пользователей? Как мы можем их выявить? Как мы поймем, что это действительные цели, а не задачи, к выполнению которых людей вынуждают некачественно спроектированные инструменты или бизнес-процессы? Одинаковы ли эти цели для всех пользователей? Меняются ли они со временем? На оставшихся страницах этой главы мы попытаемся дать ответы на перечисленные вопросы.

Цели пользователей часто существенно отличаются от наших предположений о них. К примеру, мы можем считать, что цель рядового бухгалтера – более эффективная обработка накладных. По всей вероятности, это не так. Эффективная обработка накладных – скорее цель нанимателя этого рядового бухгалтера. Сам клерк, что более вероятно, сосредоточен на иных целях – он хочет выглядеть компетентным на своем месте и сохранять интерес к работе, невзирая на необходимость выполнять рутинные и однообразные задания, хотя, возможно, и не произносит этого вслух (а может, и просто не осознает).

Независимо от профессии и поставленных перед нами задач мы в большинстве своем разделяем эти простые личные цели. Даже наши более высокие устремления связаны скорее с личным, чем с работой: получить повышение, узнать больше о своей области или же стать хорошим примером для других.

Продукты, при проектировании и создании которых преследовались только цели бизнеса, ожидает провал; личные цели пользователей требуют своей доли внимания. По причинам, которые мы более подробно рассмотрим в последующих главах, учет личных целей пользователей при проектировании продукта позволяет гораздо более эффективно достигать целей бизнеса.

Присмотревшись к большинству существующих коммерческих программ, веб-сайтов и цифровых продуктов, вы обнаружите, что их пользовательские интерфейсы пугающе далеки от целей пользователей. Эти интерфейсы раз за разом:

- заставляют пользователей чувствовать себя идиотами;
- заставляют пользователей совершать серьезные ошибки;
- требуют слишком больших трудозатрат для эффективной работы;
- не делают опыт пользователя интересным и приятным.

В большинстве своем эти программы столь же неэффективны и в достижении целей бизнеса. Не все накладные обрабатываются должным образом; клиенты обслуживаются с задержками; отсутствует толковая поддержка принятия решений... И это не случайно.

Компании, создающие такие продукты, неверно расставляют приоритеты. В основном они слишком сильно сосредотачиваются на вопросах реализации – и это уводит их в сторону от потребностей пользователей.

Даже если компании проявляют чуткость к своим пользователям, они часто бессильны изменить собственные продукты, поскольку сложившийся процесс разработки предполагает, что интерфейсом следует заниматься после начала работы над программным кодом, а иногда – даже после окончания. Но как невозможно эффективно спроектировать здание после начала строительства, так же невозможно легко заставить программу служить целям пользователя, когда уже написан значительный объем базового кода.

Наконец, даже когда компании сосредотачиваются на пользователях, они уделяют слишком пристальное внимание *задачам*, которые выполняют пользователи, и упускают из виду *цели*, ради которых выполняются эти задачи. Программа может быть технологически превосходной, прилежно решать все задачи бизнеса – и при этом являть собой серьезный коммерческий провал. Нельзя игнорировать технологию и задачи, но они представляют собой лишь часть более глобального плана, который включает в себя проектирование с учетом целей пользователей.

Цели, задачи, деятельность

Цели – не то же самое, что задачи или деятельность. Цель – это предвосхищение конечного состояния, тогда как задачи и деятельность являются лишь промежуточными этапами (на различных уровнях организации), необходимыми для достижения целей.

В иерархии, описанной Дональдом Норманом (Donald Norman), деятельность включает задачи, которые состоят из действий, в свою очередь составленных из операций. При помощи этой схемы Норман пропагандирует проектирование, ориентированное на деятельность (Activity-Centered Design, ACD), – подход, в котором внимание уделяется прежде всего пониманию деятельности. Норман утверждает, что человек приспосабливается к имеющимся инструментам и что понимание деятельности, выполняемой человеком при помощи инструментов, может положительно сказываться на дизайне этих инструментов. В основе рассуждений Нормана лежит теория деятельности – советская психологическая теория[1], рассматривающая человека через призму того,

[1] Теория деятельности выдвинута советским психологом А. Н. Леонтьевым. – *Примеч. ред.*

как он взаимодействует с окружающим миром, и в последние годы получившая применение в области изучения взаимодействия людей и компьютеров – в основном благодаря Бонни Нарди (Bonnie Nardi).

Норман делает верный вывод о том, что традиционный подход, сосредоточенный на задачах, при проектировании цифровых продуктов дает неадекватные результаты. Многие разработчики и специалисты по юзабилити по-прежнему начинают проектирование с вопроса: «Каковы задачи?» И хотя работу таким образом сделать можно, результат улучшится максимум на один балл, не станет тем решением, которое выделяет ваш продукт на рынке, и зачастую не будет по-настоящему удовлетворять пользователей.

Созданная Норманом схема ACD совершает ряд важных шагов в нужном направлении, подчеркивая важность контекста пользователя, но мы считаем, что этих шагов недостаточно. Метод вроде ACD может быть полезен при разделении на составные части того, что делает пользователь, но не отвечает на вопрос, который первым должен приходить в голову любому проектировщику: *почему* пользователь приступает к этой активности, задаче, действию или операции? Цели побуждают людей вести некую деятельность; понимание целей позволяет понять ожидания и устремления пользователей, что, в свою очередь, может помочь в определении видов деятельности, имеющих реальное отношение к дизайну вашего продукта. На уровне глубокой детализации анализ задач и деятельности полезен – но лишь после того, как будут проанализированы цели. Вопрос: «Каковы цели пользователя?» – позволяет понять *смысл* деятельности для пользователя и таким образом создавать более уместные и качественные продукты.

На тот случай, если вы все еще не чувствуете разницу между целями с одной стороны и деятельностью и задачами – с другой, есть простой способ их различать. Цели определяются человеческими мотивами и потому со временем не меняются или меняются весьма незначительно. Деятельность и задачи преходящи, поскольку почти целиком основаны на имеющейся под рукой технологии. К примеру, в поездке из Сент-Луиса в Сан-Франциско вероятные *цели* человека – скорость, удобство, безопасность. В 1850 году поселенец, желающий скорости и комфорта, путешествовал бы на крытом фургоне и держал бы при себе верное ружье. Сегодня деловой человек, направляющийся из Сент-Луиса в Сан-Франциско, путешествует на реактивном лайнере, а огнестрельное оружие в интересах безопасности ему рекомендуется оставить дома. *Цели* остались неизменными, однако деятельность и задачи изменились следом за технологиями настолько, что стали в некоторых отношениях прямо противоположными.

Строя проектирование исключительно на основе анализа деятельности и задач, мы рискуем попасть в ловушку устаревших технологий или применить модель, соответствующую целям корпорации, но не отвечающую целям пользователей. Взгляд через призму целей позволяет

пользоваться преимуществами современной технологии для исключения лишних задач и радикального упорядочения структуры деятельности. Понимание целей пользователя помогает проектировщикам избавляться от деятельности и задач, которые технология способна выполнять за человека.

Проектирование с учетом целей в определенном контексте

Многие проектировщики предполагают, что упрощение освоения интерфейсов всегда должно быть одной из задач проектирования. Простота освоения – важный принцип, однако на практике, как отмечает Бренда Лорел (Brenda Laurel), цели проектирования зависят от контекста – от того, кто наши пользователи, чем они занимаются, каковы их цели. Вы просто не сможете создать хороший дизайн, если будете следовать правилам в отрыве от целей и потребностей пользователей вашего продукта.

Возьмем для примера автоматизированную систему обработки звонков. Заработная плата пользователей этого продукта зависит от количества принятых звонков. Главная забота пользователей – не простота освоения, а эффективность – скорость, с которой можно производить переключение звонков и завершать разговоры. Легкость освоения остается важной, поскольку влияет на настроение сотрудников и, в конечном итоге, на текучесть кадров, поэтому при проектировании такой системы следует учитывать как простоту освоения, так и пропускную способность. Однако нет никаких сомнений, что пропускная способность системы – преобладающее требование пользователей к системе, а простота освоения при необходимости может отступить на задний план. После того как пользователь освоил принципы работы, его будет только раздражать программа, предлагающая проходить по шагам все этапы переключения для каждого звонка.

С другой стороны, если речь идет о справочном терминале в вестибюле корпорации, помогающем посетителям сориентироваться, простота использования для пользователей, впервые работающих на таком терминале, определенно является ведущей целью проектирования.

Один из общих принципов проектирования взаимодействия, который, похоже, особенно хорошо подходит для инструментов, призванных повышать производительность, звучит так: *результат качественного проектирования делает пользователей более эффективными*. Этот принцип принимает во внимание общечеловеческую цель не выглядеть глупо наряду с более частными целями повышения производительности бизнеса и простоты применения, которые актуальны для большинства ситуаций в деловом мире.

Выяснить, каким образом можно сделать работу пользователей продукта более эффективной, – это ваша задача как проектировщика.

Программы, дающие пользователям возможность решать стоящие перед ними задачи, но *не обращающие внимания на цели*, редко позволяют выполнять работу действительно эффективно. Если перед пользователем поставлена задача ввести в базу данных 5000 имен и адресов, самое идеальное приложение для ввода данных не даст ему ничего близко похожего на то удовлетворение, которое принесет инструмент, автоматически извлекающий имена из системы хранения счетов.

Работа пользователя состоит в концентрации на своих задачах, а работа проектировщика – в выходе за пределы задач, чтобы выяснить, *кем* являются самые важные пользователи, а затем определить, *каковы* могут быть их цели и *почему*. Процесс проектирования, очерченный на оставшихся страницах главы и подробно описанный в последующих главах первой части книги, предоставляет инструмент, который помогает находить ответы на такие вопросы и на системной основе создавать решения, опирающиеся на эту информацию.

Целеориентированный процесс проектирования

Большинство компаний, сфокусированных на технологии, не располагают адекватным процессом проектирования, ориентированного на пользователей, а то и не имеют никакого процесса вообще. Но даже более прогрессивные организации, которые могут похвастаться налаженными процессами, сталкиваются с некоторыми существенными проблемами, проистекающими из традиционных подходов к вопросам исследования и проектирования.

В последние годы бизнес-сообщество постепенно признало, что создание качественных продуктов требует исследований пользовательской аудитории, однако суть этих исследований для многих организаций по-прежнему остается загадкой. Количественные рыночные исследования и сегментация рынка вполне полезны для *продажи* продуктов, но не способны дать критически важную информацию о том, *как люди в действительности используют продукты,* в особенности если речь идет о продуктах со сложным поведением (в главе 4 мы обсудим эту тему подробнее). Вторая проблема возникает непосредственно после анализа результатов: большинство традиционных подходов ничего не говорят о том, как *преобразовать результаты исследований в проектные решения*. Сотню страниц с результатами анкетирования пользователей не так-то легко превратить в набор требований к продукту, а извлечь из них понимание того, как требования к продукту должны быть выражены в терминах ясной и осмысленной инфраструктуры интерфейса, – еще сложнее. Проектирование остается черным ящиком: «А вот здесь происходит чудо». Эта пропасть между результатами исследований и конечными проектными решениями есть порождение процесса, неспособного протянуть связи от пользователя к тому, чем продукт становится в финале. Вскоре мы увидим, как эта проблема преодолевается при помощи целеориентированного подхода.

Мост через пропасть

Как было вкратце сказано ранее, роль, которую в общем процессе разработки играет проектирование, нуждается в изменениях. Мы должны начать по-новому думать о проектировании и научиться иначе относиться к принятию решений, касающихся продукта.

Проектирование как процесс определения продукта

В промышленности значение термина **«проектирование»** (design), к несчастью, стало слишком узким. Для многих разработчиков и руководителей оно означает то, что происходит на третьей диаграмме процессов с рис. 1.1, – косметическую операцию на **модели реализации** (подробнее в главе 2). Однако при правильном применении (как показано на четвертой диаграмме процессов на рис. 1.1) проектирование выявляет требования пользователя к продукту и формирует подробное представление о поведении и внешнем виде продукта. Иначе говоря, проектирование дает настоящее **определение продукта**, основанное на целях пользователей, потребностях бизнеса и ограничениях технологии.

Проектировщики как исследователи

Если проектирование и есть определение продукта, проектировщикам необходимо смотреть на вещи более широко, нежели принято в традиционном дизайне, особенно когда предметом проектирования являются сложные интерактивные системы.

Одна из проблем существующего процесса разработки состоит в том, что роли участников чрезмерно узки: исследователи исследуют, а проектировщики проектируют (рис. 1.4). Результаты исследований поль-

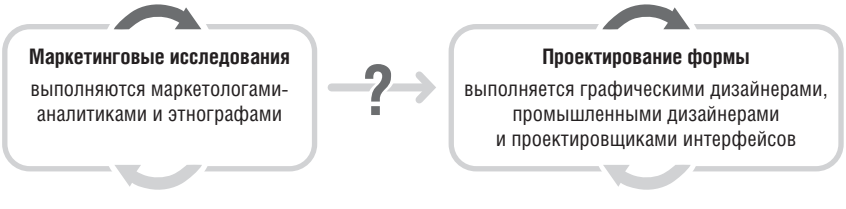


Рис. 1.4. *Проблематичный процесс проектирования.*
Исторически исследования и проектирование существовали раздельно, каждым направлением занимались свои специалисты. Под ***исследованиями*** *до недавнего времени понимали преимущественно маркетинговые исследования, а проектирование слишком часто ограничивали визуальным дизайном или поверхностным промышленным дизайном. В последнее время рамки* ***исследований целевой аудитории*** *расширились и включают в себя получение качественных этнографических данных. Однако без подключения проектировщиков к процессу исследований связь между данными исследований и проектными решениями остается в лучшем случае призрачной*

зовательской аудитории и рынка анализируются маркетологами и специалистами по юзабилити, а затем сплавляются проектировщикам или программистам. В этой модели не хватает системных средств для перевода и синтеза результатов исследований в интерфейсные решения. Один из способов решения проблемы для проектировщиков – научиться быть исследователями.

Существует неопровержимый довод в пользу участия проектировщиков в исследованиях. Эмпатия, или способность чувствовать то, что чувствуют другие люди, является одним из самых мощных инструментов проектировщиков. Прямые и обширные контакты с пользователями, без которых не обходится серьезное исследование, погружают проектировщиков в мир пользователей и заставляют думать о пользователях задолго до того, как речь пойдет о выработке решений. Одна из наиболее опасных практик при создании продукта – изоляция проектировщиков от пользователей, поскольку это не дает появиться эмпатическому знанию.

Кроме того, обычному исследователю часто сложно понять, какая информация о пользователях существенна с точки зрения проектирования. Вовлечение проектировщиков в исследование снимает оба этих вопроса.

В практике авторов проектировщики получают навыки использования методов, описанных в главе 4, и в дальнейшем самостоятельно проводят исследования без дополнительной поддержки. Описанное решение жизнеспособно при условии, что у вашей команды есть время и ресурсы, чтобы полноценно подготовить проектировщиков для использования этих методов. В противном случае более уместна будет смешанная команда, состоящая из проектировщиков и специалистов по исследованию пользовательской аудитории.

Хотя проведение исследований проектировщиками позволяет нам сделать несколько шагов навстречу целеориентированным решениям, между результатами исследований и детальными решениями по-прежнему остается разрыв. Как мы сейчас увидим, в картине не хватает нескольких фрагментов.

От исследований к проектированию: модели, требования, инфраструктура

Весьма немногие из распространенных методов проектирования включают в себя средства эффективного и систематического преобразования знаний, собранных в ходе исследований, в детальную спецификацию интерфейса. Мы уже указали на одну из причин этого: исторически сложилось так, что проектировщики выключены из цикла исследований, а потому им приходится полагаться на чужие представления о поведении и желаниях пользователей.

Другая же причина состоит в том, что очень немногие подходы фиксируют поведение пользователей в форме, пригодной для формирования

определения продукта. Вместо того чтобы давать информацию о целях пользователей, большинство методов предоставляют информацию на уровне задач. Информация такого типа хорошо подходит для создания компоновочных схем, моделирования рабочего процесса и преобразования функций в элементы пользовательского интерфейса, но далеко не столь полезна для определения общей инфраструктуры, отражающей то, чем продукт *является*, что он *делает* и как это соответствует различным потребностям пользователей.

Для преодоления разрыва нам требуется строгий систематический процесс создания моделей пользователей, определения требований к пользовательскому интерфейсу и преобразования их в общую концепцию взаимодействия (рис. 1.5). Задача целеориентированного проектирования – устранить существующий в процессе разработки цифровых продуктов разрыв между исследованиями пользовательской аудитории и проектированием эффективно сочетая новые и уже известные подходы.

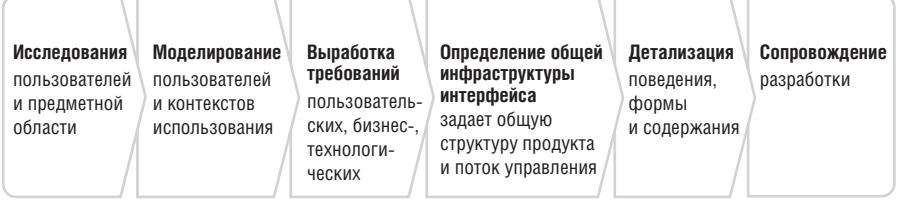


Рис. 1.5. *Процесс целеориентированного проектирования*

Обзор процесса

Целеориентированное проектирование сочетает в себе методы этнографии, интервью с заинтересованными в проекте лицами, маркетинговые исследования, моделирование пользователей, проектирование на основе сценариев, а также базовый набор принципов и шаблонов проектирования взаимодействия. Оно позволяет создавать решения, соответствующие потребностям и целям пользователей с одной стороны, а также бизнес-требованиям и технологическим ограничениям – с другой. Процесс можно грубо разделить на шесть стадий: *исследования*, *моделирование*, *выработка требований*, *определение общей инфраструктуры*, *детализация* и *сопровождение* (рис. 1.5). Эти стадии соответствуют пяти видам деятельности, составляющим процесс проектирования взаимодействия согласно Джиллиан Крэмптон Смит (Gillian Crampton Smith) и Филипу Тэйбору (Philip Tabor), – понимание, абстрагирование, структурирование, отображение, детализация, – но с более выраженным акцентом на моделировании поведения пользователей и определении поведения систем.

Оставшаяся часть главы содержит высокоуровневый обзор пяти стадий целеориентированного проектирования. В главах с 4 по 7 приводится более подробное описание процессов для каждой из стадий. На рис. 1.6 представлена развернутая диаграмма процесса, включающая основные проблемы проектирования и точки взаимодействия.

Исследования

На стадии исследований для сбора качественных данных о существующих и/или потенциальных пользователях продукта применяются такие этнографические методы, как полевые наблюдения и полевые интервью. Помимо этого, если того требует предметная область, может проводиться конкурентный анализ, обзор маркетинговых исследований, обзор статей о технологиях, материалов по стратегии брендинга, а также индивидуальное интервьюирование лиц, принимающих решения, разработчиков, специалистов в предметной области и экспертов по технологии.

Одним из основных результатов полевых исследований и интервью с пользователями является набор **поведенческих шаблонов** – характерных поведенческих моделей, помогающих классифицировать варианты использования будущего или существующего продукта. Анализ этих поведенческих шаблонов позволяет определять цели и мотивы (частные и общие желаемые результаты применения продукта). В деловой и технической областях такие поведенческие шаблоны часто совпадают с бизнес-ролями пользователей; в случае потребительской продукции – соответствуют стилю жизни пользователей. Поведенческие модели и связанные с ними цели пользователей являются основой **персонажей**, которые создаются на стадии моделирования. Маркетинговые исследования помогают находить и проводить отбор персонажей, укладывающихся в бизнес-модели. Интервью с лицами, принимающими решения, обзоры литературы и аудит пользовательского интерфейса продуктов дают проектировщикам возможность глубже вникнуть в предметную область и проливают свет на цели бизнеса, атрибуты бренда и технические ограничения, которые следует учесть при проектировании.

В главе 4 приводится более подробная информация о целеориентированных методах исследований.

Моделирование

На стадии моделирования поведенческие шаблоны и шаблоны рабочих процессов, выявленные путем анализа результатов полевых исследований и интервью, собираются вместе в виде моделей предметной области и моделей пользователей. Модели предметной области могут включать информационные потоки и диаграммы рабочих процессов. Пользовательские модели, или **персонажи**, – это подробные и структурированные **архетипы пользователей**, которые представляют собой различные

Старт проекта → Проектирование ⇄ Разработка ⇄ Тестирование → Поставка

Целеориентированное проектирование

	Активности	Объекты внимания	Участие лиц, принимающих решения	Сдаваемый результат
Исследования	**Охват** *Определение целей и графика проекта*	Задачи, сроки, финансовые ограничения, процесс, контрольные точки	**Совещания** Оценка технических возможностей и затрат	**Документ** Отчет о проделанной работе
	Аудит *Изучение доступных документов и существующих продуктов*	Бизнес-планы и маркетинговые планы, стратегия брендинга, маркетинговые исследования, планы развития продуктовой линейки, конкуренты, технологии в соответствующей области		
	Интервью с лицами, принимающими решения *Прояснение образа продукта и имеющихся ограничений*	Образ продукта, риски, благоприятные возможности, ограничения, логистика, пользователи	**Интервью** С лицами, принимающими решения, и пользователями	
	Интервью и наблюдения за пользователями *Прояснение потребностей пользователей и изучение их поведения*	Пользователи, потенциальные пользователи, модели поведения, взгляды, способности, мотивы, характеристики окружения, инструменты, проблемы	**Приемка** Предварительных результатов исследований	
Моделирование	**Персонажи** *Создание архетипов пользователей и клиентов*	Шаблоны поведения пользователей и клиентов, взгляды, способности, цели, характеристики окружения, инструменты, проблемы	**Приемка** Персонажей	
	Прочие модели *Представление особенностей предметной области, не связанных с отдельными пользователями и клиентами*	Рабочие процессы, затрагивающие группы людей, характеристики окружения, артефакты		
Выработка требований	**Контекстные сценарии** *Сочинение историй об идеальном опыте пользователей*	Как продукт соответствует жизни и среде персонажей и помогает им в достижении целей	**Приемка** Сценариев и требований	
	Требования *Определение критичных возможностей продукта*	Функциональные и информационные потребности, ментальные модели пользователей, императивы проектирования, образ продукта, бизнес-требования, технология	**Презентация** Анализ пользователей и предметной области	**Документ** Анализ пользователей и предметной области
Проектирование инфраструктуры	**Элементы** *Описание информационной и функциональной модели*	Информация, функции, механизмы, действия, объектные модели предметной области	**Приемка** Общей структуры проекта	
	Инфраструктура *Проектирование общей структуры опыта взаимодействия*	Связи между объектами, концептуальные группы, навигационные последовательности, принципы и шаблоны, поток управления, эскизы, раскадровки		
	Ключевые и проверочные сценарии *Описание взаимодействия персонажа с продуктом*	Как дизайн продукта соответствует идеальной последовательности действий пользователя и учитывает разнообразие вероятных условий	**Презентация** Концепция пользовательского интерфейса	
Детализация	**Детальное проектирование** *Доработка и уточнение деталей*	Внешний вид, идиомы, интерфейс, виджеты, поведение, информация, визуализация, бренд, опыт, язык, раскадровки	**Приемка** Детального проекта	**Документ** Спецификация формы и поведения
Сопровождение проектирования	**Корректировка спецификации** *Учет новых ограничений и сроков*	Поддержание концептуальной целостности дизайна продукта в условиях меняющихся технологических ограничений	**Совместное проектирование**	**Пересмотренный документ** Спецификация формы и поведения

Рис. 1.6. *Процесс целеориентированного проектирования в деталях*

устойчивые комбинации поведенческих моделей, склонностей, взглядов, целей, мотивов, обнаруженных на стадии исследований.

Персонажи становятся главными действующими лицами описательной методики проектирования, основанной на сценариях. На стадии определения инфраструктуры персонажи способствуют генерации концепций взаимодействия, на стадии детализации обеспечивают обратную связь, позволяющую улучшить внутреннюю согласованность дизайна и его соответствие целям, а также служат мощным инструментом коммуникации, помогающим разработчикам и руководителям видеть, на чем основывается дизайн, и ранжировать функции продукта исходя из потребностей пользователей. На стадии моделирования проектировщики применяют разнообразные методологические инструменты для синтеза, дифференциации и ранжирования персонажей. Проектировщики выявляют различные *типы* целей и связывают типы возможных моделей поведения с персонажами таким образом, чтобы не оставалось белых пятен и не возникало повторений.

Конкретное направление проектирования выбирается путем сопоставления целей персонажей и создания иерархии приоритетов, основанной на том, насколько широко цели того или иного персонажа покрывают цели других персонажей. Процесс присвоения персонажам типов определяет, насколько серьезное влияние каждый персонаж окажет на окончательную форму и поведение продукта.

Подробно о персонажах и разработке целей мы поговорим в главе 5.

Выработка требований

Методы проектирования, применяемые проектными командами на стадии выработки требований, обеспечивают столь нужную связь между пользовательскими и всеми прочими моделями и инфраструктурой проекта. Здесь используются сценарные методы проектирования с одним важным нововведением: сценарии концентрируются не на абстрактных задачах пользователей, но прежде всего на достижении целей и удовлетворении потребностей конкретных персонажей. Персонажи дают понимание того, какие задачи действительно важны и почему, что приводит к созданию интерфейса, минимизирующего число задач (усилий), но при этом увеличивающего отдачу от них. Персонажи становятся главными участниками этих сценариев, и проектировщики изучают пространство возможных решений посредством своего рода ролевой игры.

Для каждого интерфейса/ключевого персонажа процесс проектирования на данном этапе включает в себя анализ данных, связанных с персонажем, и анализ функциональных потребностей (выраженный в терминах объектов, действий и контекстов), сформированных и ранжированных с помощью целей персонажей, их моделей поведения, а также особенностей взаимодействия с другими персонажами в различных контекстах.

Такой анализ выполняется посредством последовательного уточнения **контекстного сценария**. Отправной точкой служит описание «одного дня жизни» персонажа, применяющего продукт, которое намечает высокоуровневые точки соприкосновения с продуктом, после чего происходит пошаговая детализация. Помимо требований, определяемых сценарием, проектировщики рассматривают навыки персонажа и его физические возможности, равно как и вопросы, связанные со средой применения продукта. Происходит также учет и балансирование целей бизнеса, желаемых атрибутов бренда и технических ограничений с целями и потребностями персонажа. На выходе этого процесса возникает сбалансированный **перечень требований**, включающий в себя пользовательские требования, требования бизнеса и технические ограничения, которым продукт должен удовлетворять.

Определение инфраструктуры

На стадии определения инфраструктуры команда проектировщиков создает общую концепцию продукта, определяя концепцию поведения, графического оформления и, если требуется, физической формы. Проектировщики взаимодействия синтезируют **инфраструктуру взаимодействия** при помощи контекстных сценариев в сочетании с еще двумя важнейшими методологическими инструментами. Первый инструмент – набор общих **принципов проектирования взаимодействия**, которые способствуют определению уместного в контексте различных ситуаций поведения системы. Главы 2 и 3, а также вся вторая часть книги посвящены высокоуровневым принципам проектирования взаимодействия, работающим на стадии определения инфраструктуры.

Второй важнейший методологический инструмент – это набор **шаблонов проектирования взаимодействия**, являющихся решением (вариации здесь зависят от контекста) для соответствующих типов когда-то проанализированных проблем. Принципиально эти шаблоны очень похожи на шаблоны архитектурного проектирования, созданные Кристофером Александером (Christopher Alexander). Позже Эрих Гамма (Erich Gamma) и его коллеги познакомили с шаблонами проектирования и отрасль программирования. Шаблоны проектирования взаимодействия выстроены в иерархию и эволюционируют с появлением новых контекстов. Их функция – не загнать творчество проектировщика в рамки, а дать ему точку опоры для решения сложных задач, снабдив проверенными знаниями о проектировании.

Когда информационные и функциональные потребности описаны на таком высоком уровне, они преобразуются в элементы дизайна в соответствии с принципами взаимодействия, а затем структурируются при помощи шаблонов и принципов в эскизы дизайна и описание поведения. Результатом этого процесса является **определение инфраструктуры взаимодействия** – устоявшаяся концепция проекта, задающая логическую и примерную формальную структуру для последующей детализации. Эта детализация выполняется на следующей стадии при

помощи последовательных итераций более узко сфокусированных сценариев. Такой подход часто представляет собой баланс проектирования «сверху вниз» (опирающегося на шаблоны) и проектирования «снизу вверх» (опирающегося на принципы).

Когда продукт обретает физическую форму, проектировщики взаимодействия и промышленные дизайнеры в тесном сотрудничестве прорабатывают различные **векторы ввода** и возможные **форм-факторы** продукта, используя сценарии для выявления всех «за» и «против» по каждому варианту. Как только множество вариантов сокращается до нескольких многообещающих, промышленные дизайнеры начинают создавать первые физические прототипы, чтобы проверить принципиальную работоспособность концепции взаимодействия в целом. На этой ранней стадии крайне важно, чтобы промышленные дизайнеры не уходили в свободный полет и не порождали концепции, оторванные от поведения продукта.

Как только начинается формирование инфраструктуры взаимодействия, проектировщики интерфейсов создают несколько вариантов **визуальной инфраструктуры**, которую иногда еще называют **стратегией визуального языка**. Для разработки вариантов типографики, цветовых решений и визуального стиля они задействуют атрибуты бренда, а также представление об общей структуре интерфейса.

Детализация

Стадия детализации схожа со стадией определения инфраструктуры, но в большей степени сосредоточена на подробностях реализации. Проектировщики взаимодействия фокусируются на согласованности задач, используя **ключевые (пошаговые) маршруты**, а также **проверочные сценарии**, дающие максимально подробные пути прохождения по пользовательскому интерфейсу. Графические дизайнеры определяют наборы начертаний и размеров шрифтов, пиктограмм и других визуальных элементов с очевидным ожидаемым назначением[1] и четкой визуальной иерархией, чтобы в итоге обеспечить пользователю приятный опыт взаимодействия с продуктом. Промышленные дизайнеры (если их участие требуется) принимают окончательное решение по материалам и совместно с инженерами прорабатывают схемы сборки и другие технические аспекты. Завершением стадии детализации становится подробная проектная документация – **спецификация формы и поведения** в бумажном или интерактивном формате (в зависимости от ситуации). В главе 6 применение персонажей, сценариев, принципов и шаблонов на стадиях выработки требований, определения инфраструктуры и детализации описано более подробно.

---

[1] Ожидаемое назначение (англ. affordance) – воспринимаемые и фактические качества объекта, преимущественно фундаментальные, которые определяют возможные способы обращения с этим объектом. Подробнее см. главу 13. – *Примеч. науч. ред.*

Сопровождение разработки

Даже очень продуманное и проверенное проектное решение не позволяет предусмотреть все препятствия и технические осложнения на пути разработчиков. Мы на своем опыте знаем, насколько важно оставаться в контакте с разработчиками, чтобы отвечать на их вопросы, возникающие в процессе создания продукта. Часто требуется корректировать проектные решения, упрощать их по мере того, как команда разработчиков назначает приоритеты отдельным фрагментам своей работы и урезает проект, чтобы уложиться в сроки. Если в тот момент, когда возникла нужда в таких решениях, команда проектировщиков недоступна, разработчикам приходится самостоятельно искать выход в условиях дефицита времени, что в конечном итоге может не лучшим образом сказаться на целостности продукта.

Ключ к успеху продукта – цели, а не возможности

И разработчики, и маркетологи часто говорят о продуктах в терминах функций и возможностей. Это вполне естественно, поскольку именно так разработчики создают программы – функция за функцией. Список функций – *один из способов* выразить ценность продукта для потенциального покупателя (хотя, конечно, довольно ограниченный). Проблема в том, что подобные списки содержат абстрактные концепции, дающие лишь скудное представление о том, каким образом пользователи могут повысить свою эффективность и обрести счастье, используя предлагаемые технологии.

Сузив определение продукта до списка функций и возможностей, мы проходим мимо замечательного шанса поставить возможности технологий на службу человеческим потребностям и целям. Слишком часто функции наших продуктов представляют собой лишь мозаику модных технологических новшеств, построенных на основе требований маркетологов или домыслов разработчиков, которые уделяют прискорбно мало внимания опыту взаимодействия пользователей с продуктом.

Успешный проектировщик взаимодействия обязан держать цели пользователей в поле зрения даже в хаосе и под давлением цикла разработки продукта. Хотя в этой книге описаны и многие другие техники и инструменты, к целям пользователей мы будем возвращаться постоянно. Они – та основа, на которую должно опираться проектирование взаимодействия.

Целеориентированный процесс с его четкими логическими обоснованиями проектных решений облегчает сотрудничество с инженерами и деловыми людьми, а также гарантирует, что проектирование происходит не по наитию и не является капризом творческой мысли либо отражением личных предпочтений участников разработки.

ПРИНЦИП проектирования

Проектирование взаимодействия – не гадание на кофейной гуще.

Целеориентированное проектирование – мощный инструмент, отвечающий на самые важные вопросы, которые возникают при описании и проектировании цифрового продукта:

- Кем являются мои пользователи?
- Чего пытаются достичь мои пользователи?
- Что мои пользователи думают о своих целях сами?
- Какого рода опыт будет для моих пользователей привлекательным и полезным?
- Как должен себя вести мой продукт?
- Как должен выглядеть мой продукт?
- Как пользователи будут взаимодействовать с моим продуктом?
- Как наиболее эффективно реализовать функции моего продукта?
- Как начинающие пользователи будут знакомиться с моим продуктом?
- Каким образом мой продукт сможет придать технологии привлекательный облик и сделать ее понятной и управляемой?
- Как мой продукт может решить проблемы пользователей?
- Как мой продукт поможет в достижении целей тем пользователям, которые редко работают с продуктом или имеют мало опыта?
- Каким образом мой продукт сможет удовлетворить запросы опытных пользователей, которым нужна функциональная мощь и глубина проработки?

Остальная часть этой книги посвящена ответам на перечисленные вопросы. Мы предложим вам инструменты, проверенные годами работы над сотнями продуктов, – инструменты, которые помогут вам выявить ключевых пользователей, понять их цели и транслировать это понимание в успешные проектные решения.

2

Модели реализации и ментальные модели

В компьютерной отрасли часто используют термин **компьютерная грамотность**. Ученые мужи любят разглагольствовать о том, что у некоторых она есть, а у других нет, и о том, как обладающие ею преуспеют в информационном обществе, а остальным суждено сгинуть в социоэкономических разломах. Однако компьютерная грамотность – это просто эвфемизм, с помощью которого пользователей заставляют подстраиваться под чуждую им машинную логику, вместо того чтобы приблизить программные продукты к образу мышления пользователей. В этой главе мы поговорим о том, как недостаточное понимание пользователей и их подхода к программным продуктам подпитывает разделение людей на «компьютерно грамотных» и «компьютерно неграмотных» и как программы, более соответствующие особенностям мышления и деятельности людей, способны устранить эту проблему.

Модели реализации

В каждой машине есть механизм, благодаря которому она выполняет свое предназначение. Так, кинопроектор создает волшебную движущуюся картинку за счет хитроумной цепи движущихся деталей. Он испускает очень яркий свет, в течение доли секунды проходящий через маленькую прозрачную фотографию. Затем он на долю секунды перекрывает свет, чтобы подставить под луч света следующую маленькую фотографию. Тогда он снова на мгновение открывает лампу. Эта последовательность действий выполняется двадцать четыре раза в секунду. Программные продукты не имеют механизмов в обычном понимании – на место узлов из движущихся частей пришли алгоритмы и модули кода, сообщающиеся друг с другом. Представление о том, как в действительности работает машина или программа, Дональд

Норман (Norman, 1989) и другие называют **системной моделью**; мы предпочитаем термин **модель реализации**, поскольку такая модель описывает подробности реализации программы в коде.

Пользовательские ментальные модели

Кинозрителю, пришедшему на захватывающую драму, легко забыть про тонкости, связанные с перфорацией кинопленки и работой обтюратора. В действительности многие посетители кинотеатров очень слабо представляют себе, как работает проектор и чем он отличается от телевизора. В представлении зрителя проектор просто выдает движущуюся картинку на большой экран. Это и называется пользовательской **ментальной моделью**, или **концептуальной моделью**.

Чтобы пользоваться сложным устройством, человеку вовсе не нужно знать в деталях, как оно функционирует на самом деле, а потому он создает упрощенную мысленную схему, достаточно мощную для описания взаимодействия с устройством, но не всегда отражающую его реальную внутреннюю механику. К примеру, многие люди считают, что, когда они включают пылесос или блендер в розетку на стене, электричество начинает течь из стены в прибор, словно вода, по маленькой черной трубке электрошнура. Эта ментальная модель абсолютно адекватна с точки зрения использования бытовой электротехники. Тот факт, что модель реализации электрической сети не содержит ничего похожего на жидкость, путешествующую по проводам, и что электрический потенциал меняет знак 100 раз в секунду, для пользователя значения не имеет, хотя энергокомпании приходится знать все эти тонкости.

В цифровом мире различия между пользовательской ментальной моделью и моделью реализации зачастую весьма ощутимы. Мы склонны игнорировать то обстоятельство, что сотовый телефон работает не так, как стационарный; он представляет собой приемопередатчик, способный в ходе двухминутного разговора сменить с пяток базовых станций сотового оператора. Однако это знание не помогает нам понять, как *пользоваться* телефоном.

Расхождения между моделью реализации и ментальной моделью становятся особенно сильными в случае с программными приложениями, где сложность реализации зачастую так высока, что пользователь почти теряет возможность устанавливать простые механистические связи между своими действиями и реакциями программы. Занимаясь на компьютере цифровым монтажом звука или созданием визуальных спецэффектов вроде морфинга, мы лишены аналогий с механическим миром, так что наши ментальные модели неизбежно отличаются от модели реализации. Даже если связи будут видны, для большинства людей они останутся непонятными.

Модели представления

Лицом программного или цифрового продукта, как его видят люди, является его поведение, создаваемое программистом или проектировщиком. Это представление вовсе не должно быть точным описанием того, что происходит внутри компьютера, но, к несчастью, зачастую именно им и является. Возможность *демонстрировать* функционирование компьютера в отрыве от производимых им в реальности действий проявляется в программном обеспечении сильнее, чем в любой другой среде. Она позволяет сообразительному проектировщику скрывать некоторые из наиболее малопривлекательных особенностей того, как программа выполняет свою работу. Этот разрыв между реализацией и предлагаемым объяснением служит источником *третьей* модели, возникающей в цифровом мире, – **модели представления**. Модель представления – это избранный проектировщиком способ предъявления пользователю функционирования программы. Дональд Норман (Norman, 1989) называет это просто **моделью проектирования**.

В мире программного обеспечения модель представления программы может (и часто должна) достаточно сильно отличаться от внутренней архитектуры программы. Например, операционная система может изображать сетевой файловый сервер так же, как простой локальный диск. Эта модель не отражает того, что физически диск может быть удален на многие километры. У понятия «модель представления» нет широко распространенного аналога в механическом мире. На рис. 2.1 представлены отношения, связывающие все три описанные модели.

Чем ближе к пользовательской ментальной модели окажется модель представления, тем легче будет пользователю работать с программой и понимать ее. Наоборот, модель представления, слишком сильно приближенная к модели реализации, обычно значительно затрудняет освоение и применение программы при условии, что ментальные модели, которые пользователь строит для своих задач, отличаются от модели реализации программы (а это условие практически всегда выполняется).

Все мы склонны создавать ментальные модели, упрощающие реальность. Поэтому, конструируя модели представления, которые проще моделей реализации, мы помогаем пользователю лучше понять продукт. К примеру, нажатие на педаль тормоза в автомобиле может вызывать в мыслях образ рычага, который упирается в колесо и замедляет его за счет трения. В реальном же механизме присутствуют гидравлические цилиндры и шланги, а также металлические колодки, сжимающие перфорированный диск; однако мы выбрасываем все это из головы, создавая более эффективную, хотя и менее точную ментальную модель. В случае с программами мы, например, представляем себе, что лист электронной таблицы прокручивается и являет нам новые ячейки, когда мы нажимаем на полосу прокрутки. Ничего такого в дейст-

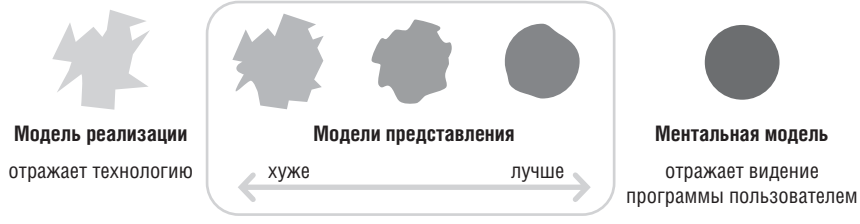


Рис. 2.1. *Внутреннее устройство программного обеспечения часто оказывается данностью и продиктовано различными ограничениями технического и делового характера. Модель того, как в действительности работает программа, называется* ***моделью реализации****. То, как пользователи воспринимают свои задачи и как программа помогает в решении этих задач, есть* ***ментальная модель*** *взаимодействия с программой. Она основана на представлениях пользователей о том, как они решают свои задачи, и о том, как может работать компьютер. То, как проектировщики преподносят работу программы пользователю, называется* ***моделью представления****. В отличие от двух предшествующих моделей, она является тем аспектом программного обеспечения, который в значительной степени поддается контролю со стороны проектировщиков. Одна из важнейших целей проектировщика – максимально приблизить модель представления к ментальной модели пользователей. Таким образом, проектировщикам очень важно детально знать, как целевые пользователи видят работу, выполняемую с помощью программы*

вительности не происходит: нет никакого листа с ячейками, вместо него – плотно упакованная структура данных со значениями, связанными друг с другом посредством указателей, на основе которых программа синтезирует в реальном времени новое изображение на экране.

Понимание того, как в действительности работает программа, помогает ее использовать, однако это понимание обходится весьма дорого. Упрощение внешних проявлений процессов и ситуаций – один из наиболее эффективных способов, которыми компьютер может помочь человеку. Как следствие, пользовательские интерфейсы, построенные на основе ментальных моделей пользователей, существенно выигрывают по сравнению с интерфейсами, которые всего лишь отражают модель реализации.

Пользовательский интерфейс должен следовать пользовательской ментальной модели, а не модели реализации.

В программе Adobe Photoshop пользователь имеет возможность регулировать цветовой баланс и яркость изображения при помощи функции Варианты (Variations). В интерфейсе этой функции вместо полей ввода численных значений цветов (модель реализации) предлагается набор миниатюр, различающихся настройками цветового баланса (рис. 2.2). Пользователю достаточно щелкнуть по миниатюре, наиболее точно

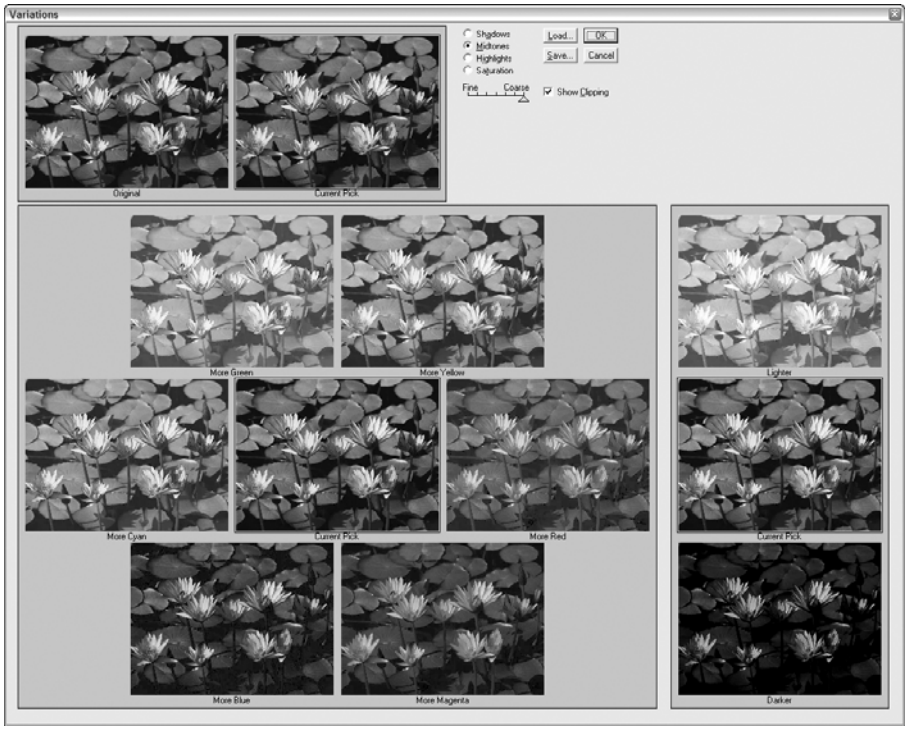


Рис. 2.2. *Adobe Photoshop дает замечательный пример интерфейса, основанного на пользовательской ментальной модели. Диалоговое окно Варианты (Variations) предлагает набор миниатюр, варьирующихся по цветовому балансу и яркости с регулируемым шагом. Пользователю достаточно щелкнуть по миниатюре, наиболее точно представляющей желаемую коррекцию цвета. Выбранное изображение становится точкой отсчета для новых вариантов. Этот интерфейс следует ментальной модели художника, которому требуется получить определенный конечный вид изображения, а не набор абстрактных чисел*

представляющей желаемую коррекцию цвета. Этот интерфейс приближен к ментальной модели действия, ибо пользователь – вероятно, художник или дизайнер – думает о том, как должно выглядеть изображение, а не об абстрактных числах.

Когда модель представления программы приближена к пользовательской ментальной модели, из интерфейса уходит избыточная сложность, поскольку пользователь получает когнитивную инфраструктуру, которая ясно указывает, как могут быть достигнуты его цели и удовлетворены его потребности.

ПРИНЦИП проектирования

Целеориентированное взаимодействие отражает пользовательские ментальные модели.

Пользовательская ментальная модель не обязана быть правдивой или точной, но она непременно должна позволять пользователю эффективно работать. К примеру, в восприятии большинства пользователей, не являющихся технарями, экран – сердце компьютера. Это очень даже естественно, поскольку именно в экран они постоянно смотрят и именно на экране видят, что делает компьютер. Если такому пользователю сказать, что компьютер в действительности – лишь маленькая кремниевая микросхема вон в том ящике под столом, он, вероятно, пожмет плечами и проигнорирует эту бессмысленную (с его точки зрения) информацию. Тот факт, что центральный процессор – совсем не то же, что дисплей, не помогает ему решать вопросы взаимодействия с компьютером, хотя с технической точки зрения такое представление более точно.

Большинство программных продуктов следуют модели реализации

Программу, которая отражает собственную модель реализации, спроектировать существенно легче: по кнопке на каждую функцию, по полю на каждую порцию входных данных, по странице на каждый шаг транзакции, по диалоговому окну на каждый модуль кода... совершенно логично с точки зрения разработчика. Адекватно отражая инфраструктуру инженерной деятельности, такой подход, однако, слабо помогает создавать целостные механизмы для достижения пользователями своих целей. Результатом становится интерфейс, отталкивающий и запутывающий пользователя, – примерно как вездесущая паутина труб из фильма-антиутопии Терри Гиллиама (Terry Gilliam) «Бразилия» (фильм, полный замечательных и остроумных примеров ужасных интерфейсов).

Интерфейсы, спроектированные инженерами, следуют модели реализации

Пользовательские интерфейсы и схемы взаимодействия, спроектированные инженерами (которые досконально знают принципы работы программы), имеют модель представления, очень точно следующую модели реализации. С точки зрения инженера такая модель логична, достоверна и точна; к сожалению, она малопонятна и неэффективна для пользователя. Большинству пользователей не особенно интересно, как именно реализована функциональность программы.

Хороший пример пользовательского интерфейса, соответствующего модели реализации, – обычный компонентный домашний кинотеатр: пользователю необходимо знать, как соединены внутренние компоненты, чтобы перейти, скажем, от проигрывания DVD к просмотру канала кабельного телевидения. В большинстве систем пользователю приходится переключать источники видео, а иногда даже менять ис-

пользуемые пульты, чтобы просто получить доступ к функциям просмотра телепередачи. В более современных продуктах встречается альтернативный подход, в большей степени ориентированный на ментальные модели пользователей: система следит за конфигурационными настройками на пульте. Пользователю достаточно нажать кнопку «Смотреть ТВ» – и пульт отправляет нужные команды телевизору, тюнеру, DVD-проигрывателю и музыкальному центру, избавляя пользователя от необходимости знать о происходящем за кулисами.

Даже интерфейс Windows временами соскальзывает к модели реализации. Если перетащить мышью файл из одной папки в другую в пределах одного жесткого диска[1], система интерпретирует это как команду «Переместить», то есть файл удаляется из первой папки и попадает во вторую, что хорошо соответствует ментальной модели. Однако перетаскивание файла с диска C на диск D будет воспринято как команда «Копировать», то есть файл будет добавлен во вторую папку, но *не* будет удален из первой. Истоки этого поведения находятся в модели реализации – в том, каким образом работает файловая система. При перемещении файла из одной папки в другую в пределах одного диска операционная система всего лишь переносит запись имени файла в таблице размещения файлов. Стирания и повторной записи файла при этом не происходит. Однако перенос файла на другой диск требует физического копирования данных на целевой диск. Чтобы соответствовать ментальной модели пользователя, система должна после копирования удалить оригинал, хотя это и противоречит модели реализации.

Такая непоследовательность поведения компьютера при выполнении двух, казалось бы, похожих действий способна порождать у пользователей значительный *когнитивный диссонанс* (путаницу, возникающую в результате столкновения двух противоречащих друг другу картин реальности), что, в свою очередь, затрудняет освоение этого простого взаимодействия. Чтобы добиться желаемого результата, пользователь должен понимать, что поведение компьютера зависит от физической природы конкретных видов устройств хранения данных.

При определенных условиях трактовка перетаскивания файла с диска на диск как команды «Копировать» может быть желательным поведением системы, особенно при копировании файлов с жесткого диска на сменные носители вроде USB-накопителей типа flash, поэтому многие люди не осознают, что это лишь побочный эффект модели реализации. Однако по мере развития компьютерной техники логические тома перестают представлять собой только физические диски, и этот побочный эффект все реже оказывается полезным, а напротив, начинает раздражать, поскольку нам приходится держать в голове особенности поведения каждого вида накопителей.

---

[1] Более точно – в пределах одного логического раздела жесткого диска. – *Примеч. ред.*

Математическое мышление приводит к созданию интерфейсов, следующих модели реализации

Проектировщики взаимодействия должны защищать пользователей от моделей реализации. То, что некий подход хорош для решения задач создания программного обеспечения, еще не означает, что он годится для создания ментальной модели пользователя. То, что автомобиль сделан из сваренных между собой металлических деталей, еще не означает, что вам нужно быть опытным сварщиком, чтобы водить этот автомобиль.

Структуры данных и алгоритмы, служащие для представления и обработки информации в программах, в большинстве своем являются логическими средствами, основанными на математических алгоритмах. Программисты отлично разбираются в этих алгоритмах, включая рекурсию, иерархические структуры данных и многопоточную обработку. Проблема возникает в том случае, когда пользовательский интерфейс пытается точно отобразить понятия рекурсии, иерархических данных или многопоточной обработки.

Математическое мышление – ловушка модели реализации, в которую особенно легко попадают программисты. Поскольку они прибегают к математическому мышлению при решении задач программирования, им естественным образом начинает казаться, что эти математические модели – подходящая база для изобретения пользовательских интерфейсов. Это очень глубокое заблуждение.

Пользователи не понимают булеву алгебру.

Например, одним из наиболее надежных и полезных инструментов в арсенале программиста является булева алгебра. Это компактная математическая система, удобно описывающая поведение сугубо двоичной вселенной любого компьютера. В булевой алгебре есть лишь две основных операции: `И (AND)` и `ИЛИ (OR)`. Сложность в том, что в естественном языке есть слова **«и»** и **«или»**, которые людьми, не связанными с программированием, обычно трактуются ровно противоположным образом, нежели булевы `И` и `ИЛИ`. Если программа использует для общения булеву нотацию, *следует ожидать*, что пользователь проинтерпретирует ее неверно.

Например, эта проблема часто возникает при обращении к базам данных. Если из файла со списком сотрудников требуется извлечь тех, кто живет в Аризоне, вместе с теми, кто живет в Техасе, человеку мы скажем: «Найди всех моих сотрудников в Аризоне и Техасе». Чтобы правильно объяснить то же самое базе данных на языке булевой алгебры, мы должны сказать: «Найди сотрудников в Аризоне ИЛИ Техасе». Ни один сотрудник не живет сразу в двух штатах, поэтому фраза

«Найди сотрудников в Аризоне И Техасе» просто не имеет смысла. В булевой системе она практически всегда вернет пустое множество в качестве ответа.

Любая программа, взаимодействующая с пользователем на языке булевой алгебры, обречена на серьезнейшие проблемы с пользовательским интерфейсом. Неразумно ожидать, что пользователи преодолеют такую путаницу. Они хорошо знакомы со своим родным языком. Почему же они должны выражать свои мысли на незнакомом языке, который – вот еще фокусы! – дает новые значения базовым словам?

Модели представления механической и информационной эры

Мы живем в уникальное время перехода из эры промышленных, механических артефактов в эру цифровых, информационных объектов. Изменения только начались, и их темп быстро нарастает. Промышленный переворот, пережитый человечеством в результате индустриализации, вероятно, покажется малозначительным по сравнению с потрясениями информационной эры.

Представления механической эры

Для нас совершенно естественно пытаться перенести привычные представления и язык более ранней эпохи в новую, менее привычную эпоху. Как показывает история промышленной революции, плоды новой технологии поначалу часто приходится описывать на языке более старой технологии. К примеру, мы называли железнодорожные локомотивы *железными конями*, а за автомобилями закрепился ярлык *безлошадных повозок*. К сожалению, наше мышление зависит от языка и понятий сильнее, чем мы готовы признать.

Естественно, мы стремимся применять устаревшие представления в новой среде. Временами такой перенос уместен, поскольку функция осталась прежней, пусть даже изменилась технология, лежащая в ее основе. Так, переходя от процесса набора текста на пишущей машинке к обработке текста на компьютере, мы используем представление механической эры для задачи, суть которой не поменялась. В пишущих машинках применялись маленькие металлические пластины (tabs), позволявшие быстро передвинуть каретку через несколько пробелов в нужную колонку. Исходя из применявшейся технологии этот процесс именовался табулированием (настройкой табулятора). В текстовых процессорах тоже есть позиции табуляции, поскольку их функция осталась неизменной: работаете ли вы с бумагой, прокручиваемой при помощи валика, или с изображениями на экране компьютера, – вам требуется быстро попадать в позиции с заданным отступом.

Однако не все представления механической эры следует дословно переносить в цифровой мир. Мы ведь не управляем автомобилем при помощи поводьев или румпеля – хотя оба подхода были опробованы на заре автомобилестроения и на совершенствование идиомы руля, уместной в автомобиле, ушло много лет. В текстовых процессорах мы обходимся без загрузки чистого листа, когда текущий лист заполнен, – вместо этого мы работаем с непрерывно прокручивающимся документом, на котором визуально отмечены разрывы страниц.

Новая технология требует новых представлений

Иногда задачи, процессы, понятия и даже цели возникают только потому, что новая технология впервые делает их возможными. До определенного момента у них не было поводов для существования – и они никому не приходили в голову. Так, когда телефон только появился, его превозносили, в числе прочего, как средство вещания музыки и новостей, но именно личное общение стало самым востребованным и широко распространенным применением телефона. В то время никто даже не мог подумать о телефоне как о повсеместно распространенной личной вещи, которую люди носят в карманах и сумочках и которая имеет привычку раздражающе звонить во время театрального представления.

Нам, людям с образом мышления механической эры, в первый раз трудно воспринять удачную реализацию, соответствующую информационной эре. Действительные преимущества созданного программного продукта часто остаются незримыми, пока продукт не наберет достаточно большого числа пользователей. К примеру, действительное преимущество электронной почты – не в том, что она просто быстрее традиционной почты (с точки зрения механической эры), а в том, что она ведет к демократизации и сокращению иерархий на современных предприятиях – это преимущество информационной эры. Действительное преимущество Всемирной паутины состоит не в удешевлении коммуникации и распространения информации (взгляды механической эры) – настоящее преимущество заключено в создании виртуальных сообществ, и это преимущество информационной эры обнаружилось только после того, как такие сообщества возникли прямо на наших глазах. Поскольку нам сложно прогнозировать, как будут применяться программные продукты, мы склонны слишком сильно полагаться на представления прошлой, механической эры.

Представления механической эры ухудшают качество взаимодействия с пользователем

Перенося привычные представления механической эры в компьютер, мы сталкиваемся с проблемой. Проще говоря, процессы и представления механической эры не лучшим образом влияют на взаимодействие пользователей с продуктами информационной эры. Механические

процедуры легче выполнять вручную, чем при помощи компьютера. Скажем, печать почтового адреса на конверте при помощи компьютера требует значительно больших усилий, чем подписывание конверта авторучкой (хотя результат в первом случае может выглядеть аккуратнее). Ситуация улучшается только в случае автоматизации процесса, требующего большого числа повторов (когда требуется, допустим, подписать пятьсот конвертов).

А вот другой пример – список контактов на компьютере. Если мы выведем его на экран в виде маленькой книжечки в переплете, ею будет гораздо сложнее и на порядок менее удобно пользоваться, чем реальной записной книжкой. В реальной записной книжке, к примеру, имена хранятся в алфавитном порядке, ключом служат фамилии. И что делать, если требуется найти человека по имени? Артефакт механической эры в данном случае не поможет – придется просмотреть все страницы вручную. Точная цифровая копия этой записной книжки тоже будет бессильна – она точно так же не умеет искать по именам. Разница состоит в том, что с переносом книжки на экран компьютера вы теряете многочисленные легкие визуальные подсказки, доступные вам в бумажной книжке (загнутые уголки страниц, карандашные пометки на полях). В то же время полосы прокрутки и диалоговые окна сложнее использовать, сложнее визуализировать, сложнее понять, чем простую операцию перелистывания страниц.

ПРИНЦИП проектирования

Не копируйте артефакты механической эры в пользовательских интерфейсах без учета возможностей информационной эры.

Механическим системам реального мира присущи все плюсы и минусы среды, которой они принадлежат, как, например, в случае с бумагой и ручкой. Программное обеспечение обладает совершенно иным набором плюсов и минусов, однако, если механические представления копируются в неизменном виде, минусы старого и нового подходов суммируются. В примере с записной книжкой компьютеру ничего не стоит выполнять поиск по имени. Однако, храня записи тем же способом, что и при использовании механического артефакта, мы лишаем себя новых возможностей для поиска. Мы ограничиваем возможности информационной среды, не сохраняя при этом никаких преимуществ исходной физической среды.

Ориентируясь на представления механической эры, проектировщики упускают из виду мощные возможности компьютера, позволяющие управлять информацией на ином, гораздо более изощренном уровне. Использование представлений механической эры в интерфейсе пользователя может проявляться в виде *метафоры*, которая искусственно ограничивает результат проектирования. Более подробно об ошибках, связанных с упованием на метафоры в пользовательских интерфейсах, мы расскажем в главе 13.

Совершенствуем представления механической эры: пример

Хотя новые технологии могут приводить к совершенно новым концепциям, они способны также расширять и развивать существующие, что позволяет проектировщикам использовать мощь новых технологий на пользу людям, совершенствуя содержащиеся в интерфейсах представления.

Возьмем для примера календарь. В физическом мире календари сделаны из бумаги и разбиты на страницы по месяцам. Таков разумный компромисс, основанный на размерах листа бумаги, папки, портфеля и ящиков стола.

Программы, содержащие визуальное представление календаря, встречаются повсеместно и почти всегда отображают месяцы по одному. Даже если программа способна показать сразу несколько месяцев, как, скажем, Outlook, дни все равно выводятся в виде последовательностей длиной в один месяц. Почему?

Бумажные календари отображали месяцы по одному из-за ограничений, связанных с размерами бумаги, а конец месяца выглядел удобной точкой разрыва. Экраны компьютеров не столь ограничены, однако большинство проектировщиков настойчиво копируют этот артефакт механической эры (рис. 2.3). На компьютере календарь запросто может быть непрерывной прокручиваемой последовательностью из дней, недель или месяцев, как показано на рис. 2.4. Запланировать что-либо на период с 28 августа по 4 сентября просто, если недели следуют одна за другой непрерывно, а не разнесены волей проектировщика по месяцам.

К тому же сетка в электронных календарях практически всегда имеет фиксированный размер ячеек. Почему нельзя изменять ширину колонок с днями или высоту строк с неделями, как в электронной таблице? Вам, вероятно, хотелось бы увеличить размер клеток выходных дней, чтобы обозначить их важность в сравнении с буднями. Если вы – деловой человек, рабочая неделя в вашем календаре потребует больше пространства, чем неделя отпуска. Эти идиомы столь же хорошо известны, как и электронные таблицы, то есть их вполне можно считать универсальными; однако представления механической эры укоренились так глубоко, что мало кому из разработчиков программ удается отойти от них.

Проектировщик программы, представленной на рис. 2.3, вероятно, считает календарь каноническим объектом, который должен точно следовать установленным правилам. Поразительно, что большинство программ для планирования времени внутри – в модели реализации – используют непрерывную шкалу времени и лишь в пользовательском интерфейсе – в модели представления – отображают время в виде дискретных месяцев!

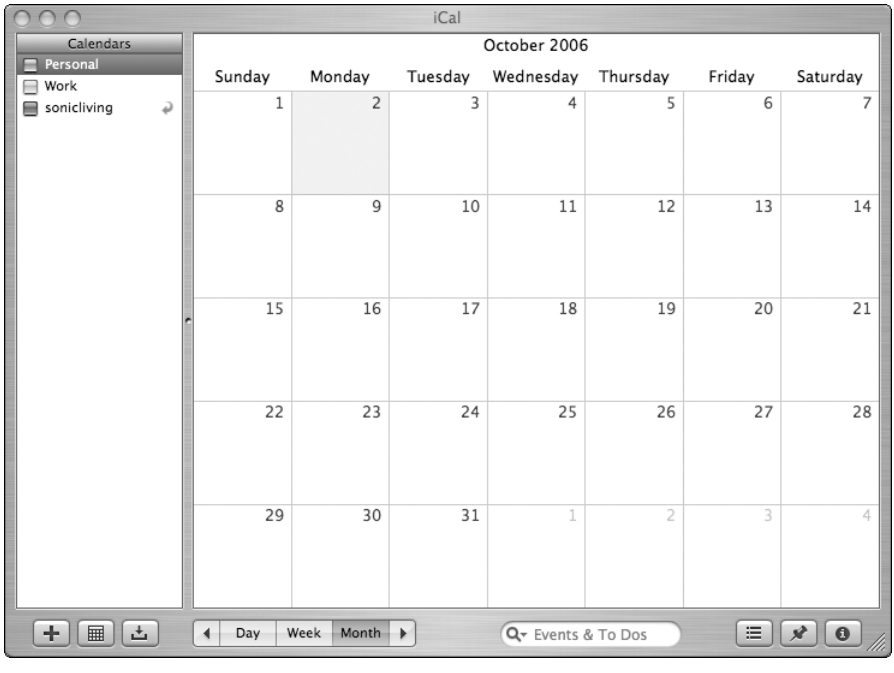


Рис. 2.3. *Вездесущий календарь настолько привычен, что при проектировании мы редко подходим к нему с мерками информационной эры. Календари исходно проектировались для печати на стопке бумаги, а не для отображения на экране компьютера. Как бы вы перепроектировали календарь? Какие из аспектов календаря являются артефактами устаревшего формата механической эры?*

Кто-то может возразить, что календарь, имеющий по месяцу на страницу, лучше, поскольку знаком и привычен для пользователей. Но ведь новая модель не так уж сильно отличается от старой модели, она всего лишь позволяет легко делать то, что раньше вызывало сложности, – планировать дела на границах месяцев. Людям не составляет труда адаптироваться к новым, более полезным представлениям уже знакомых систем.

Существенные изменения должны приносить значительные улучшения.

Выполненные в «бумажном» стиле календари в личных информационных системах (PIM – personal information manager) и планировщиках – это немое свидетельство того, в какой степени наш язык влияет на проектирование. Находясь в плену слов механической эры, мы будем создавать программы механической эры. Улучшить программы позволит мышление, соответствующее информационной эре.

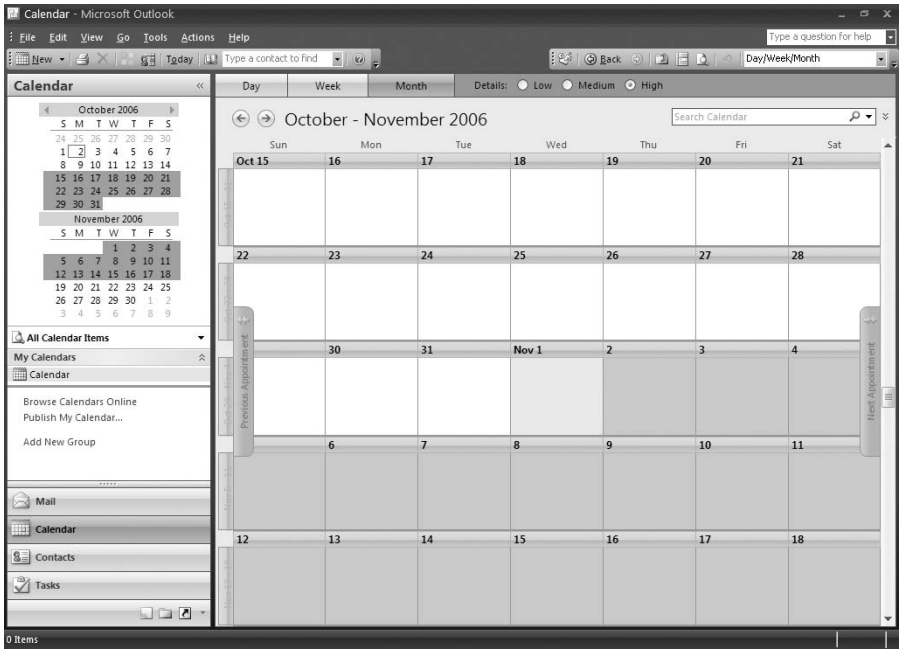


Рис. 2.4. *Прокрутка – хорошо знакомая пользователям компьютеров идиома. Почему бы не улучшить с ее помощью календарь, заменив ориентированное на постраничную печать представление прокручиваемым? Этот бесконечный календарь делает все то же самое, что и старый, а кроме того, он решает связанную с механическим представлением проблему планирования промежутков времени, пересекающих границы месяцев. Не следует переносить старые ограничения на новую платформу просто по привычке. Какие еще усовершенствования вы можете предложить?*

3

Новички, эксперты и середняки

Высвобождая из целлофана новенькую коробку с программой или мобильным телефоном, пользователи компьютеров в большинстве своем прекрасно отдают себе отчет в том, что впереди у них – несколько дней разочарования и раздражения в связи с освоением нового интерфейса. С другой стороны, многие опытные пользователи цифровых продуктов нередко испытывают постоянное раздражение из-за того, что программа продолжает считать их зелеными новичками. И найти точку равновесия между потребностями новичков и потребностями экспертов кажется невозможным.

Это одна из вечных головоломок в проектировании взаимодействия и интерфейсов: как отыскать единое интерфейсное решение, отвечающее потребностям и начинающих пользователей, и экспертов? Некоторые программисты и проектировщики решили вовсе отказаться от этой затеи; вместо этого они выделяют режим для начинающих с помощью набора мастеров, а функциональность, необходимую экспертам, прячут глубоко в меню. Естественно, делать лишнюю работу, связанную с продвижением по шагам мастера, не хочется никому, однако переключение в режим для экспертов требует от пользователя сакрального знания о назначении загадочных команд в длинной последовательности меню и обычно представляет собой прыжок с довольно высокой скалы в акулье логово интерфейса, спроектированного исходя из модели реализации. Как же выйти из этого положения? Решение загадки связано с иным пониманием того, как пользователи овладевают новыми понятиями и задачами.

Вечные середняки

Большинство пользователей – не начинающие и не эксперты; они *середняки*.

Распределение уровня опыта людей в определенной области деятельности, как и многие другие распределения генеральной совокупности, тяготеет к классической форме статистической колоколообразной кривой (рис. 3.1). Практически для любой сферы деятельности, требующей знаний или навыков, на графике, который связывает уровень навыков с числом обладателей навыков такого уровня, в левой части соберутся сравнительно немногочисленные начинающие, в правой окажется горстка экспертов, а большинство – середняки – попадет в центр.

Статистика, правда, кое о чем умалчивает. Колоколообразное распределение – это мгновенный снимок ситуации. Хотя середняки в большинстве своем остаются середняками, начинающие не слишком долго задерживаются в новичках. Сложность поддержания навыков на высоком уровне ведет к тому, что экспертами становятся и перестают быть довольно быстро, однако новички сменяются еще быстрее. Как начинающие, так и эксперты имеют тенденцию с течением времени переходить в разряд людей со средним уровнем навыков.

Хотя *все люди* какое-то минимальное время пребывают в ранге начинающих, *никто* не задерживается в этой группе надолго. Люди не любят быть некомпетентными, новички же некомпетентны по определению. Напротив, обучение и совершенствование приносят удовлетворение, поэтому начинающие очень быстро становятся середняками – или же вообще выпадают из процесса. Скажем, все лыжники некоторое время являются начинающими, но те, кто в течение заметного времени больше падает, чем катается, быстро покидают спорт. Остальные вскоре переходят от детских горок к нормальным склонам. И лишь очень немногие дорастают до самых сложных трасс «для смертников».

Рис. 3.1. *Требования, предъявляемые пользователями к цифровым продуктам, очень сильно зависят от их опыта*

ПРИНЦИП проектирования

Никто не желает оставаться начинающим.

Те, кто находится в левой части кривой распределения, либо смещаются к центральной части «колокола», либо полностью исчезают с графика и ищут себе такой продукт или род деятельности, который *позволит* им стать середняками. Таким образом, большинство пользователей постоянно обладают адекватными навыками и стремятся к их совершенствованию, а их навыки уходят и вновь возвращаются подобно приливу и отливу, в зависимости от того, как часто они работают с программой. Первым важность проектирования для середняков отметил Ларри Константайн. В своей книге «Software for Use»[1] (Constantine, 2004) он называет таких пользователей **совершенствующимися середняками** (improving intermediates). Авторы этой книги предпочитают термин **вечные середняки**, ибо хотя начинающие быстро переходят в разряд середняков, они редко продвигаются дальше и становятся экспертами.

На хорошем горнолыжном курорте есть пологий спуск для начинающих и несколько сложных склонов, бросающих серьезный вызов умелому лыжнику. Однако чтобы продолжать работать и приносить прибыль, курорт будет делать основную ставку на вечно среднего лыжника, при этом не отпугивая начинающих и не оскорбляя чувства экспертов. Начинающему должно быть легко перейти в мир средних лыжников, а профессионал не должен на своем отвесном спуске спотыкаться о средства помощи для боязливых и осторожных вечных середняков.

Хорошо сбалансированный пользовательский интерфейс во многом строится по тому же принципу. Вместо того чтобы потакать потребностям новичков или экспертов, он направлен главным образом на удовлетворение нужд вечных середняков. В то же время он обеспечивает механизмы, достаточные для эффективной работы «крайних» составляющих аудитории.

Зачастую пользователи-середняки были бы рады больше узнать о программе – просто у них нет на это времени. Однако порой у них появляется возможность плотно поработать с продуктом на протяжении нескольких недель – чтобы, скажем, закончить крупный проект. В этот период они узнают что-то новое о программе, и тогда их знание выходит за прежние границы.

А иногда они месяцами не запускают программу и забывают существенную часть того, что когда-то знали. Они не станут из-за этого начинающими, но по возвращении к программе им потребуются подсказки, чтобы освежить в памяти прежние знания.

---

[1] Л. Константайн, Л. Локвуд «Разработка программного обеспечения». – Пер. с англ. – СПб: Питер, 2004.

Для некоторых специализированных продуктов имеет смысл оптимизировать взаимодействие исходя из потребностей экспертов. Особенно это касается инструментов, поддерживающих профессиональную деятельность технически ориентированных людей, в которой важнее всего обеспечить высокий уровень эффективности. В эту категорию часто попадают инструменты для разработчиков, а также специальные измерительные и медицинские приборы. Мы ожидаем, что пользователи этих продуктов, приступая к работе, уже обладают необходимыми техническими познаниями и готовы потратить значительное время и силы на то, чтобы в совершенстве овладеть приложением.

Подобно этому есть и такие продукты, которые требуют оптимизации для начинающих. В частности, это продукты, которые используются редко или кратковременно, а также продукты для людей с определенными физическими ограничениями. В нашей практике примерами таких продуктов могут служить информационные киоски для общественных мест вроде музеев, а также устройства, помогающие пожилым пациентам с ограниченной подвижностью измерять кровяное давление.

Нас часто спрашивают, для какой части аудитории следует оптимизировать потребительские веб-сайты – для начинающих или для середняков? Мы полагаем, что здесь применимы те же соображения, что и в случае других цифровых продуктов. Качественно спроектированный интерфейс веб-сайта должен помогать пользователям быстро осваивать навигацию и функциональность. Здесь стоит обратить внимание на тот факт, что даже клиент, уже посетивший ваш сайт несколько раз и знакомый с вашими предложениями, а также в целом с идиомами взаимодействия, характерными для интернет-пространств, может заходить к вам недостаточно часто, чтобы запомнить, как сайт организован. Поэтому взаимодействие с сайтом важно делать как можно более прозрачным и очевидным. Кроме того, в последнее время набирает популярность идея отслеживать действия пользователя на сайте и на этой основе адаптировать сайт к потребностям пользователя. В этой связи полезно использовать cookie-файлы, чтобы идентифицировать новых посетителей и предлагать им ненавязчивую помощь при первом обращении к сайту.

Проектирование для пользователей с различной подготовкой

Теперь соотнесем наш «колокол» середняков с тем, как происходит разработка программного обеспечения. Программисты неизбежно становятся экспертами в создаваемых ими программах, поскольку вынуждены изучить все возможные варианты использования, вплоть до наиболее невероятных и сомнительных, чтобы реализовать их в программном коде. Они естественным образом склонны проектировать программы на основе модели реализации, назначив всем вариантам

взаимодействия *равные* приоритеты. Будучи экспертами, они могут легко разобраться в таком интерфейсе.

В то же время сотрудники отдела продаж, маркетологи, руководство часто демонстрируют продукт покупателям, журналистам, партнерам, инвесторам, которые совершенно не знакомы с продуктом. Из-за постоянного взаимодействия с начинающими у этих профессионалов возникает сильно искаженный взгляд на сообщество пользователей. Поэтому не удивительно, что отдел продаж и маркетологи отстаивают перекраивание интерфейса под нужды начинающих. Образно говоря, они требуют к каждому «велосипеду» прикрутить еще «пару маленьких колес», чтобы выручить несчастных новичков.

Программисты организуют взаимодействие способами, подходящими только для экспертов, тогда как маркетологи требуют инструментов взаимодействия, удобных только для начинающих, – и это при том, что, как мы только что видели, самая крупная, самая устойчивая и самая важная группа пользователей – это середняки.

Сложно поверить, что обычной практикой может стать игнорирование потребностей большинства реальных пользователей, однако чаще всего именно так и происходит. Это хорошо видно на примере многих корпоративных и коммерческих программных продуктов. Дизайн продуктов в целом скроен в расчете на пользователей-экспертов, однако, чтобы угодить представлениям маркетологов о новых пользователях, на эти продукты навешены неуклюжие инструменты вроде мастеров (wizards) и помощника Скрепыша. Эксперты вообще редко пользуются такими инструментами, а у начинающих очень скоро возникает желание избавиться от них как от надоедливого напоминания о былой некомпетентности. Однако вечные середняки, составляющие большинство, окружены этими недостатками вечно.

ПРИНЦИП проектирования

Оптимизируйте для середняков.

Наша цель состоит не в том, чтобы угождать начинающим, – равно как и не в том, чтобы подгонять середняков быстрее становиться экспертами. Наша цель тройственна: быстро и безболезненно переводить начинающих в середняки, не создавать препятствий на пути середняков, желающих стать экспертами, и – самое главное – следить за тем, чтобы вечные середняки, оставаясь в своей части спектра навыков, были счастливы.

Необходимо больше времени уделять тому, чтобы наши программы были мощными и простыми в использовании именно для пользователей из разряда вечных середняков. Нужды начинающих и экспертов, безусловно, следует учитывать – но только не ценой создания неудобств для самого крупного сегмента пользователей. В оставшейся части главы мы опишем ряд базовых стратегий для реализации этой задачи.

Что нужно начинающим

Начинающие, несомненно, – люди чувствительные, и деморализовать новичка очень легко. Однако не следует забывать, что человек *никогда* не ставит перед собой цели оставаться начинающим. Никто не хочет быть новичком. Это лишь обряд инициации, через который должны пройти все. Хорошие программы сокращают этот обряд, не заостряя на нем внимания.

Проектировщику взаимодействия лучше всего представлять себе, что пользователи (особенно начинающие) – люди одновременно очень умные и очень занятые. Им требуется некоторый инструктаж, но не слишком обширный, так что этот процесс должен быть скоротечным и целенаправленным. Если инструктор по лыжам начнет читать лекцию по метеорологии и экологии горных склонов, он растеряет своих слушателей независимо от их способностей к горнолыжному спорту. То, что пользователь желает научиться работать с программой, вовсе не означает, что ему хочется или необходимо знать, как эта программа устроена внутри.

Считайте пользователей людьми очень умными, но очень занятыми.

С другой стороны, умные люди всегда учатся лучше, если видят причины и следствия, так что вы должны дать им некоторое представление о том, почему все работает так, как работает. Чтобы избавиться от противоречия, мы используем ментальные модели. Когда модель представления интерфейса хорошо соответствует пользовательской ментальной модели (как это описано в главе 2), мы обеспечиваем пользователю необходимый уровень понимания, не заставляя его разбираться в модели реализации.

Встречаем новичков

Новый пользователь должен быстро усвоить используемые в программе понятия и уловить ее предназначение, иначе он от нее откажется. Поэтому первоочередная задача проектировщика – убедиться, что программа адекватным образом отражает ментальные модели задач пользователя. Возможно, между сеансами работы с программой пользователь не будет помнить, какая в точности команда требовалась для работы с определенным объектом, но, если концептуальная структура интерфейса соответствует его ментальной модели, он определенно запомнит такие важные понятия, как отношения между объектами и действиями.

Переход начинающих в разряд середняков требует дополнительной помощи со стороны программы, но эта дополнительная помощь станет для них обузой, как только они достигнут цели. Отсюда следует, что любая предлагаемая вами дополнительная поддержка не должна быть

фиксированной частью интерфейса. Она должна уметь исчезать тогда, когда необходимость в ней отпала.

Стандартная встроенная справка – неподходящий способ поддержки начинающих. Более подробно о справочной системе мы поговорим в главе 26, но сейчас стоит сказать, что ее основное назначение – быть источником справочной информации, а начинающим нужна не справочная информация, им нужна обзорная информация, такая как «Знакомство с программой» (guided tour).

«Знакомство с программой» как отдельная функция, представленная в диалоговом окне, – отличное средство для представления обзорной информации о программе, ее назначении и возможностях. Когда пользователь запускает программу, ему демонстрируется диалоговое окно, содержащее сведения об основных целях и возможностях программы и перечисляющее ключевые инструменты. Если этот справочный материал сосредоточен вокруг вопросов, возникающих у новичков, таких как цели и возможности продукта, и обходит стороной темы, актуальные для середняков и экспертов (об этом чуть ниже), он будет адекватной помощью начинающим.

Новички часто полагаются на меню при изучении и исполнении команд (в главе 22 мы подробно обсудим, почему это так). Каким бы медленным и тяжеловесным инструментом ни были меню – они полны и подробны, и это дает чувство уверенности. Открываемые командами меню диалоговые окна (если таковые имеются) также должны содержать (краткие) пояснения и удобную кнопку отмены (Cancel).

Что нужно экспертам

Эксперты – это также важнейшая группа, поскольку они оказывают непропорционально большое влияние на менее опытных пользователей. Потенциальный покупатель, рассматривая ваш продукт, будет больше полагаться на мнение эксперта, чем на мнение середняка. Если эксперт сказал: «Продукт не очень хороший», – то, возможно, имел в виду: «Продукт не очень хороший для экспертов». Однако начинающий не знает об этом и последует совету эксперта – хотя совет, возможно, неадекватен.

Экспертам временами требуются экзотические возможности, причем некоторые из этих возможностей они могут использовать очень часто. В то же время эксперты определенно выступают за более быстрый доступ к регулярно используемым инструментам из рабочего набора, который может быть довольно большим. Иначе говоря, экспертам нужны короткие пути ко всему.

Любой, кто пользуется программным продуктом по нескольку часов в день, очень быстро усваивает особенности интерфейса. Вопрос о том, *хотят* ли пользователи заучивать часто используемые команды, не ставится: это попросту неизбежно. Высокая частота использования программы одновременно оправдывает запоминание и требует его.

Эксперты все время активно ищут новую информацию о связях между своими действиями с одной стороны и поведением и внешним видом программы – с другой. Экспертам нравятся новые мощные функции. Их уровень владения программой позволяет им не испытывать беспокойства по поводу возрастающей сложности.

Что нужно вечным середнякам

Вечным середнякам нужен доступ к инструментам. Им не нужно объяснять назначение и возможности программы – они уже все это знают. Всплывающие подсказки (см. главу 23) – превосходная идиома для вечных середняков. Всплывающие подсказки ничего не говорят о назначении, смысле и возможностях – они лишь наикратчайшим образом обозначают функцию, занимая минимальное экранное пространство.

Вечные середняки знают, как пользоваться справочными материалами. Они обладают мотивацией копнуть глубже и научиться – при условии, что не потребуется осваивать сразу большие объемы информации. Это означает, что встроенная справка – инструмент для вечных середняков. Они пользуются ею посредством предметного указателя, так что эта часть справки должна быть очень хорошо проработана.

Вечные середняки разделяют функции на регулярно используемые и применяемые лишь изредка. Пользователь может экспериментировать с непонятными возможностями, но вскоре он выявит – вероятно, подсознательно – рабочий набор часто используемых инструментов и будет требовать, чтобы инструменты из этого набора были размещены на самом видном месте в пользовательском интерфейсе – там, где их будет легко найти и запомнить.

Вечные середняки обычно знают о существовании дополнительных возможностей – хотя могут не нуждаться в них и не представлять себе, как с ними работать. Однако знание о наличии таких возможностей добавляет вечному середняку уверенности, убеждает его, что он совершил правильный выбор, когда сделал ставку на эту программу. Лыжника со средними навыками может очень вдохновлять знание о том, что уже вот за теми деревьями лежит действительно крутой экстремальный спуск для экспертов, – даже если он не планирует когда-либо им воспользоваться. Это задает идеал, к которому можно стремиться и о котором можно мечтать, а также создает ощущение первоклассности лыжного курорта.

Код вашей программы обязан учитывать как потребности абсолютных новичков, так и все возможные ситуации, с которыми может столкнуться эксперт. Но не позволяйте этому техническому требованию влиять на проектные решения. Да, вы должны обеспечить экспертов такими возможностями. Да, вы должны обеспечить поддержку для начинающих. Но большую часть своих талантов, времени и ресурсов вы обязаны отдать проектированию наилучшего взаимодействия для самой представительной части аудитории – вечных середняков.

4

Как понять пользователей: качественные исследования

Оценивать результат усилий по проектированию в конечном итоге следует исходя из того, насколько успешно он отвечает требованиям как пользователей, так и компании – инициатора разработки. Если у проектировщика нет ясного и детального представления о пользователях, для которых выполняется проектирование, если у него отсутствует понимание имеющихся ограничений, организационных задач и бизнес-целей, которые являются движущей силой разработки, то шансов на успех остается очень мало – неважно, насколько при этом хороши навыки и творческие способности проектировщика.

Настоящего понимания в этих областях невозможно достичь, перелопачивая горы чисел, добытых в результате количественных исследований (скажем, маркетинговых), хотя для ответов на другие вопросы эти данные могут оказаться крайне важными. Такое глубокое знание можно получить лишь посредством *качественных* исследований. Существует множество видов качественных исследований, и каждый из них может сыграть важную роль в формировании общей картины проектируемого продукта. В этой главе мы рассмотрим конкретные методы проведения качественных исследований, которые служат фундаментом для приемов проектирования, описанных в последующих главах. В конце главы мы коротко поговорим о том, чем в этой работе могут (и чем не могут) помочь количественные исследования.

Качественные и количественные исследования

Слово «исследование» у большинства людей ассоциируется с научностью подхода и объективностью. Такие ассоциации не то чтобы непра-

вильны, но они склоняют многих к мысли, что достоверными можно считать только те исследования, которые имеют в качестве результата непреложное свидетельство объективности – количественные данные. Представление о том, что числа не лгут, широко распространено в инженерном и деловом сообществах, хотя все мы осознаем, что числа – а в особенности статистические данные, относящиеся к деятельности человека, – подвержены манипуляциям и искаженным интерпретациям по крайней мере не меньше, чем слова.

Данные, собираемые точными науками вроде физики, фундаментально отличаются от данных, касающихся человеческой деятельности: ведь у электронов нет ежеминутно меняющегося настроения, а строгие методы контроля, которые в физических экспериментах позволяют изолировать нужные аспекты поведения исследуемых объектов, в социальных науках практически отсутствуют. Любая попытка свести человеческое поведение к статистическим данным, вероятно, приведет к потере важных нюансов, имеющих решающее значение для проектирования продуктов. Количественные исследования способны отвечать лишь на вопросы «сколько?» и «в каких объемах?», используя нескольких простых шкал. Качественные исследования способны дать ответ на вопросы «что», «как» и «почему» развернуто и с разнообразными подробностями, отражающими сложность человеческой жизни.

Представители общественных наук давно осознали, что человеческое поведение слишком сложно и подвержено воздействию слишком многих факторов, чтобы полагаться при его анализе сугубо на количественные данные. Позаимствовав подходы из антропологии и других социальных наук, специалисты по юзабилити и проектированию разработали многочисленные качественные методы для сбора полезных данных о поведении пользователей с прагматичной целью: способствовать созданию продуктов, более точно отвечающих потребностям пользователей.

Значение качественных исследований

Качественные исследования помогают нам понять предметную область, контекст и ограничения продукта – причем иным, более действенным способом, чем количественные исследования. Они также помогают нам выявить шаблоны поведения пользователей и потенциальных пользователей продукта быстрее и проще по сравнению с количественными методами. В частности, качественные исследования позволяют нам изучить:

- поведение, взгляды, склонности потенциальных пользователей продукта;
- предметную область – технический, экологический и деловой контексты разрабатываемого продукта;
- используемый лексикон и прочие социальные аспекты предметной области;

- способы применения существующих продуктов.

Качественные исследования могут также способствовать ходу проектирования, поскольку:

- обеспечивают доверие и уважение к команде проектировщиков (так как источник проектных решений можно проследить вплоть до результатов исследований);
- объединяют команду разрабатываемого продукта общим для всех пониманием особенностей предметной области и проблем пользователей;
- дают руководителям возможность принимать решения по тем или иным вопросам проектирования продукта на основе данных – вместо догадок и личных предпочтений.

По опыту авторов, качественные методы в сравнении с количественными, как правило, оказываются более быстрыми, менее дорогостоящими и с большей вероятностью дают полезные ответы на важные вопросы, без которых невозможно достичь превосходных результатов. Вот эти вопросы:

- Как продукт вписывается в более широкий контекст жизни пользователей?
- Каковы основные цели людей при работе с продуктом, и какие базовые задачи позволяют людям достигать этих целей?
- Какой опыт люди находят привлекательным? Как этот опыт соотносится с проектируемым продуктом?
- С какими проблемами сталкиваются люди, когда используют при решении своих задач имеющиеся технологии?

Значение качественных исследований не ограничивается поддержкой процесса проектирования. По нашему опыту, время, потраченное на то, чтобы увидеть за пользовательской аудиторией человеческие существа, способно принести плоды в виде важных для бизнеса прозрений, к которым невозможно прийти средствами традиционных исследований рынка.

Вот весьма показательный случай: клиент попросил нас выполнить исследование пользователей в контексте разработки Windows-продукта, предназначенного для любительского видеомонтажа в домашних условиях. Будучи опытным разработчиком программ для монтажа и создания видео, клиент к тому моменту уже успел провести традиционное исследование рынка и обнаружил хорошую нишу для бизнеса: продукт для людей, которые приобрели цифровую видеокамеру и компьютер, но еще не успели связать эти два устройства.

В рамках полевого исследования мы провели интервью с десятками людей из числа потенциальных пользователей. Первое открытие было вполне предсказуемым: больше всех занимались видеосъемкой и стремились делиться смонтированными записями родители. А вот вторая

находка оказалась несколько пугающей: из двенадцати человек, к которым мы зашли в гости, лишь один сумел подключить видеокамеру к компьютеру – да и то при помощи коллеги, сведущего в информационных технологиях. Одним из обязательных условий успеха продукта была способность людей перенести видео на компьютер для монтажа, однако на тот момент, как оказалось, очень тяжело было заставить порт FireWire или карту видеозахвата как следует работать на персональном компьютере с процессором Intel.

После четырех дней исследований мы помогли клиенту принять решение: приостановить разработку продукта – что, вероятно, сэкономило компании значительные средства.

Виды качественных исследований

Книги по социальным наукам и юзабилити описывают множество методов и приемов проведения качественных исследований, и мы настоятельно рекомендуем читателям изучить эту литературу. В данной главе мы сосредоточим внимание на тех методиках, которые зарекомендовали себя в нашей практике в течение последних десяти лет. При этом время от времени мы будем отвлекаться на сходные методы, применяемые в сферах юзабилити и проектирования в целом. Мы постараемся представить эти техники с прагматической точки зрения, не углубляясь в теорию. Вот перечень методик качественных исследований, которые показали себя наиболее полезными в нашей деятельности:

- интервьюирование заинтересованных лиц;
- интервьюирование экспертов в предметной области (ЭПО);
- интервьюирование пользователей и покупателей;
- наблюдение за пользователями/этнографические полевые исследования;
- обзор литературы;
- аудит продукта/прототипа и конкурирующих решений.

Интервьюирование заинтересованных лиц

Исследование, предваряющее проектирование любого нового продукта, должно начинаться с получения представления о техническом окружении и бизнес-контексте продукта. Практически всегда продукт проектируется (или перепроектируется) для достижения одной или нескольких конкретных бизнес-целей (как правило, речь идет об извлечении прибыли). Обязанность проектировщиков – создавать решения, не теряя из виду эти бизнес-цели, поэтому крайне важно, чтобы команда проектировщиков начинала работу с изучения возможностей и ограничений, стоящих за краткой спецификацией проекта.

Как метко замечает Дональд Шён (Donald Schön), «проектирование – это общение с материалом» (Schön and Bennett, 1996). Чтобы проектировщик смог предложить подходящее решение, он должен понимать

возможности и ограничения «материала», используемого для создания продукта, будь то строки кода или формованный пластик.

В общем случае **заинтересованное лицо** – это любой человек, обладающий полномочиями в отношении проектируемого продукта и/или несущий ответственность за какой-либо его аспект. Говоря более конкретно, заинтересованные лица – это ключевые члены организации, инициирующей работы по проекту; как правило, в этот круг входят высшие должностные лица, менеджеры и представители отделов разработки, продаж, производства, маркетинга, юзабилити, дизайна, а также службы поддержки. Среди них могут быть также аналогичные специалисты из других организаций, состоящих в партнерских отношениях с организацией – инициатором разработки.

Интервьюирование заинтересованных лиц должно проводиться до начала любых исследований пользовательской аудитории, поскольку возникающие обсуждения нередко задают способы проведения пользовательских исследований. Кроме того, обычно оказывается более эффективным интервьюировать заинтересованных лиц поодиночке, а не в группах, объединяющих сразу несколько отделов. Интервью тет-а-тет способствует искренности заинтересованного лица и гарантирует, что взгляды отдельных людей не затеряются в общей массе. (В подобных интервью большой интерес представляет информация о том, насколько участники проекта разделяют или не разделяют общее видение продукта.) Для такого интервью обычно достаточно одного часа, хотя можно проводить дополнительные встречи, если кто-то из заинтересованных лиц окажется исключительно полезным источником информации.

От заинтересованных лиц важно получить информацию по следующим вопросам:

- **Предварительное видение продукта.** Может обнаружиться, что, как в притче о слоне и слепых мудрецах, каждый из отделов обладает отличным от других и несколько ограниченным представлением о проектируемом продукте, так что часть усилий в процессе проектирования придется направить на гармонизацию этих представлений с представлениями пользователей и покупателей.
- **Бюджет и график проекта.** Обсуждение этой темы часто помогает проверить адекватность масштабов планируемых работ по проектированию. Эти данные являются основой для принятия решений в тех случаях, когда исследования пользовательской аудитории показывают, что требуется больший (или, наоборот, меньший) масштаб работ.
- **Технические возможности и ограничения.** Еще один важный фактор, определяющий рамки проектирования, – четкое понимание того, какие технические возможности есть в распоряжении команды при имеющемся бюджете, сроках и технологических ограничениях. Часто бывает также, что продукт создается для зарабатывания де-

нег с помощью новой технологии. Понимание возможностей такой технологии способно помочь при выборе направления разработки.

- **Потребности бизнеса.** Команде проектировщиков важно понимать, чего старается добиться бизнес. Это создает еще одну точку принятия решений в том случае, если исследование пользователей вскроет конфликт между потребностями бизнеса и потребностями пользователей. В ходе проектирования необходимо, насколько это возможно, создать выигрышную ситуацию одновременно для пользователей, покупателей и поставщиков продукта.
- **Представления заинтересованных лиц о пользователях.** Заинтересованные лица, состоящие в контакте с пользователями (например, представители службы поддержки клиентов), могут представить важные наблюдения, которые помогут вам сформулировать план исследования пользовательской аудитории. Вы можете обнаружить также значительные расхождения между тем, как некоторые заинтересованные лица воспринимают пользователей, и тем, что выяснится в ходе исследований. Эта информация может позже послужить темой для обсуждения с руководством.

Понимание этих аспектов и того, как они влияют на проектные решения, способствует разработке удачных продуктов. Каким бы притягательным ни был спроектированный продукт для пользователей и клиентов, без учета осуществимости и жизнеспособности предложенного решения нет никаких шансов, что продукт преуспеет.

Обсуждение этих тем важно еще и для выработки общего языка и взаимопонимания между группами проектировщиков, руководителей и разработчиков. Ваша работа как проектировщика – создать видение продукта, в которое верит вся команда. Участники проекта вряд ли увидят в предлагаемых решениях отражение своей системы ценностей, если вы не потратите достаточное время на то, чтобы понять их взгляды. Поскольку вывод вашего продукта в свет попадает в сферу ответственности и полномочий этих людей, знания, которыми они располагают, и их мнение неизбежно оказываются важными. Если не спросить их об этом мнении сразу, оно больно ударит по вам впоследствии, приняв, к примеру, форму критики предложенных вами решений.

Интервьюирование экспертов в предметной области (ЭПО)

На ранних стадиях проектирования неоценимый вклад часто дает выявление и интервьюирование нескольких **экспертов в предметной области** (ЭПО) – людей, сведущих в предметной области, на которую ориентирован проектируемый продукт. Многие ЭПО когда-то сами были пользователями текущей версии продукта или его прежних версий, а теперь могут быть преподавателями, менеджерами или консультантами. Часто это не сами заинтересованные лица, а нанятые ими эксперты. Как и заинтересованные лица, ЭПО способны представить продукт и его пользователей под интересным углом зрения, однако проек-

тировщикам следует проявлять осторожность и понимать, что точка зрения ЭПО в определенном смысле искажена. Вот некоторые из вещей, которые следует знать о работе с ЭПО:

- **ЭПО – это зачастую пользователи-эксперты.** Длительный опыт работы с продуктом или в предметной области продукта означает, что они могли привыкнуть к существующим интерфейсам. Они также могут предпочитать профессиональные инструменты инструментам, спроектированным для вечных середняков. Часто ЭПО уже не являются пользователями продукта и смотрят на вещи скорее с точки зрения менеджера.
- **ЭПО хорошо осведомлены, но они не проектировщики.** У них может быть множество идей о том, как улучшить продукт. Некоторые из идей могут быть верными и ценными, однако наиболее полезная часть информации, содержащейся в их предложениях, – это исходные *проблемы*, побуждающие ЭПО предлагать такие решения. Как и в случае с пользователями, получив рацпредложение, спросите: «Как это поможет вам или пользователям?»
- **ЭПО необходимы в сложных или специализированных предметных областях**, таких как медицина, наука или финансовые службы. Когда вы проектируете продукт для технической или какой-либо другой специализированной предметной области, вам, вероятно, потребуются советы ЭПО, если вы сами не являетесь таковым. Обращайтесь к ЭПО за информацией об имеющихся нормах и о зарекомендовавших себя на практике подходах. Знания ЭПО о пользовательских ролях и характеристиках имеют критическое значение при планировании исследования пользовательской аудитории в сложных предметных областях.
- **Вам понадобится общаться с ЭПО в течение всего процесса проектирования.** Если предметная область продукта требует работы с ЭПО, у вас должна быть возможность обращаться к ним на различных стадиях проектирования, чтобы проверять интерфейсные решения на соответствие реальной деятельности. Не забудьте договориться о такой возможности на первых же интервью.

Интервьюирование покупателей

Очень легко спутать пользователей с покупателями. В случае с потребительскими продуктами покупатель и пользователь часто представлены в одном лице, однако в корпоративных и технических областях слова «пользователи» и «покупатели» редко относятся к одной и той же группе людей. Хотя интервьюировать следует обе группы, у каждой из них будет собственная точка зрения на продукт. Эти точки зрения следует совершенно по-разному учитывать при принятии окончательных проектных решений.

Покупатели продукта – это люди, принимающие решение о его приобретении. Если речь идет о потребительских продуктах, покупатели за-

частую являются пользователями продукта, хотя если продукты ориентированы на детей или подростков, то покупателями будут родители или другие взрослые, опекающие их. В случае с большинством технических, медицинских и производственных продуктов покупателем оказывается вовсе не пользователь – часто это кто-либо из руководства компании либо менеджер по информационным технологиям, имеющий свои цели и потребности, отличные от пользовательских. Чтобы сделать продукт *жизнеспособным*, важно понимать покупателей и их цели. Не менее важно осознавать, что покупатели редко сами пользуются продуктом, а когда все же делают это, то совсем не так, как пользователи.

В интервью с покупателями вам необходимо понять:

- каковы их цели в контексте приобретения продукта;
- что их не устраивает в существующих решениях;
- каков процесс принятия решения при покупке продуктов наподобие того, который вы проектируете;
- их роль в установке, обслуживании и управлении продуктом;
- проблемы предметной области и особенности используемой терминологии.

Подобно ЭПО, покупатели могут предлагать множество идей о том, как улучшить продукт. Как и в случае с ЭПО, важно анализировать их предложения с точки зрения тех особенностей или проблем, которые послужили поводом для возникновения этих идей, поскольку лучшие, более целостные решения могут стать очевидными позже в процессе проектирования.

Интервьюирование пользователей

Пользователи продукта в процессе проектирования должны находиться в центре внимания. Именно эти люди (а не их руководители и не команда поддержки) лично пытаются добиться каких-то результатов с помощью продукта. Если вы перепроектируете или улучшаете существующий продукт, важно общаться не только с нынешними, но и с **потенциальными пользователями**, то есть с людьми, которые сегодня не пользуются продуктом, однако являются хорошими кандидатами на его использование в будущем, поскольку имеют потребности, удовлетворяемые продуктом, и входят в его целевую аудиторию. Интервьюирование обеих категорий позволяет выявить то влияние, которое оказывает на поведение и образ мысли пользователя опыт работы с существующей версией продукта.

Мы заинтересованы в том, чтобы получить от пользователей следующую информацию:

- контекст интеграции продукта (или аналогичной системы, если продукт еще не создан) в жизнь или рабочий процесс пользователей – когда, почему и каким образом применяется (или будет применяться) продукт;

- осведомленность в предметной области с точки зрения пользователя – что необходимо знать пользователям, чтобы делать свою работу;
- существующие задачи и виды деятельности – как те, которые выполняются при помощи данного продукта, так и те, которые не поддерживаются им;
- цели и мотивы использования продукта;
- ментальная модель – как пользователи думают о своей работе и деятельности, а также чего они ожидают от продукта;
- проблемы и сложности при работе с продуктом (или с аналогичной системой, если продукт еще не создан).

Наблюдение за пользователями

Большинство людей не способны точно описывать собственное поведение (Pinker, 1999), особенно когда находятся вне контекста своей деятельности. Верно также и то, что из боязни показаться тупыми, некомпетентными или невежливыми многие люди избегают обсуждать поведение программ, которое кажется им проблемным или непонятным.

Из этого следует, что интервью, проводимое вне контекста ситуации, которую стремится понять проектировщик, даст менее полные и менее точные данные. Вы можете обсудить с пользователями их представления о собственном поведении, а можете непосредственно наблюдать это поведение. Второй вариант дает лучшие результаты.

Вероятно, наиболее эффективным методом сбора качественных данных о пользователях является сочетание интервьюирования и наблюдения. Это дает проектировщику возможность задавать уточняющие вопросы и получать пояснения к тем ситуациям и действиям, которые он наблюдает в реальном времени.

Для фиксации того, что говорят и делают пользователи, многие юзабилити-специалисты применяют технические средства, такие как аудио- и видеозапись. Следует проявлять осторожность и не пользоваться такими устройствами слишком бесцеремонно, в противном случае пользователи будут отвлекаться и вести себя не так, как в отсутствие записывающих устройств. По своему опыту можем сказать, что записная книжка и фотоаппарат позволяют нам фиксировать все, что необходимо, не влияя на искренность обмена информацией. Как правило, мы не расчехляем фотоаппарат до тех пор, пока не возникнет хороший контакт с интервьюируемым человеком, а затем снимаем те особенности рабочей среды, которые тяжело зафиксировать при помощи блокнота. С другой стороны, разумное использование видеозаписей иногда становится мощным риторическим инструментом, позволяющим убедить заинтересованных лиц в реальности спорных или неожиданных результатов исследований. Видеозапись полезна также в ситуациях, когда тяжело делать заметки, скажем в движущемся автомобиле.

Обзор литературы

Параллельно с интервьюированием заинтересованных лиц команде проектировщиков следует изучить какую-либо литературу, касающуюся продукта или его предметной области. Сюда могут и должны быть включены маркетинговые планы, стратегия бренда, исследования рынка, опросы пользователей, технические спецификации и информационные материалы, статьи в деловых и технических журналах, связанных с предметной областью, сравнительный анализ конкурентных решений, результаты поиска в Интернете похожих продуктов и новостей о них, результаты и метрики юзабилити-исследований, а также данные службы поддержки, такие как статистика обращений пользователей за поддержкой.

Команда проектировщиков должна собрать эту литературу и использовать ее как основу для формирования списка вопросов к заинтересованным лицам и ЭПО, а затем в качестве источника дополнительных данных о предметной области и терминологии, а также для сравнения с уже собранными данными о пользователях.

Аудит продукта и конкурирующих решений

Часто оказывается полезным параллельно с интервьюированием заинтересованных лиц и ЭПО изучить любые существующие версии или прототипы продукта, а также его основных конкурентов. Тем самым команда проектировщиков получает хорошее представление о состоянии дел в области и почву для подготовки вопросов к интервью. В идеале команде проектировщиков следует провести неформальную **эвристическую** или **экспертную оценку** интерфейсов как своего продукта, так и конкурирующих продуктов, проверяя их на соответствие принципам качественного взаимодействия и визуального дизайна (подобных тем, что будут представлены далее в этой книге). Данная процедура позволяет команде ознакомиться с сильными и слабыми сторонами доступных пользователю продуктов и дает общее представление о текущем объеме функциональности продукта.

Этнографические интервью: интервьюирование и наблюдение за пользователями

Исходя из многолетнего опыта практических исследований в области проектирования мы полагаем, что сочетание индивидуальных интервью с наблюдением – самый полезный и эффективный в арсенале проектировщика инструмент для сбора качественных данных о пользователях и их целях. Техника **этнографических интервью** представляет собой сочетание методик включенного наблюдения и направленного интервьюирования.

Хью Бейер (Hugh Beyer) и Карен Хольцблат (Karen Holtzblatt) создали технику проведения этнографического интервью, которую они назы-

вают **контекстным исследованием** (Contextual Inquiry). Их методика быстро и заслуженно получила широкое распространение в отрасли и является хорошей основой для качественных исследований пользовательской аудитории. Она подробно описывается в первых четырех главах их книги «Contextual Design»[1] (Beyer and Holtzblatt, 1998). Методика контекстного исследования по своей сути близка к описываемым здесь методикам, однако имеются тонкие и важные различия.

Контекстное исследование

Согласно Бейеру и Хольцблат, контекстное исследование основано на **ремесленнической (мастер – подмастерье) модели** обучения: необходимо наблюдать за пользователем и задавать ему вопросы так, как если бы он был высококлассным ремесленником, а интервьюер – его новым подмастерьем. Бейер и Хольцблат перечисляют также четыре базовых принципа организации этнографических интервью:

- **Контекст.** Вместо того чтобы проводить интервью в чистой белой комнате, следует взаимодействовать с пользователями и наблюдать за ними в естественной рабочей среде или в ином уместном для данного продукта физическом контексте. Наблюдение за пользователями, когда они заняты своей деятельностью, и расспросы, происходящие в привычном для них окружении с множеством артефактов, используемых ими ежедневно, помогают извлечь на поверхность самые важные особенности их поведения.
- **Сотрудничество.** Интервью и наблюдение должны иметь характер совместного с пользователем исследования, в котором наблюдение деятельности перемежается обсуждением ее структуры и тонкостей.
- **Интерпретация.** Работа проектировщика в большой степени сводится к выявлению того, что стоит за словами и поведением пользователей и особенностями их среды обитания. Задача проектировщика – рассмотреть собранные данные как единое целое и раскрыть то влияние, которое они должны оказать на проектирование. При этом интервьюер, однако, должен проявлять осторожность и не выдвигать предположений, основанных на собственной интерпретации фактов, не проверив эти предположения вместе с пользователями.
- **Направленность.** Вместо того чтобы жестко придерживаться заранее составленного опросника или, наоборот, отпускать интервью «в свободное плавание», проектировщик должен аккуратно направлять ход беседы в поисках данных, имеющих отношение к вопросам проектирования.

---

[1] Стоит также обратить внимание на книгу, написанную при участии Карен Хольцблат: «Rapid Contextual Design». – *Примеч. науч. ред.*

Усовершенствование контекстного исследования

Контекстное исследование закладывает прочный теоретический фундамент для качественных исследований, однако как частный метод оно имеет определенные ограничения и недостатки. По нашему опыту, перечисленные ниже дополнения к процессу позволяют значительно повысить эффективность фазы исследований, что помогает создать лучшую базу для успешного проектирования.

- **Сокращение продолжительности интервью.** Контекстное исследование предполагает, что интервью с пользователем длится целый день. Авторы выяснили, что для сбора необходимых данных достаточно даже часовой беседы при условии, что запланировано достаточное количество интервью (около шести вдумчиво выбранных пользователей на каждую гипотетическую роль или тип). Гораздо легче и продуктивнее сделать более разнообразную выборку пользователей, которые согласятся провести с проектировщиком по часу, чем найти пользователей, которые согласятся потратить на интервью целый день.
- **Использование компактной команды проектировщиков.** Контекстное исследование подразумевает наличие большой команды проектировщиков, проводящей параллельно сразу несколько интервью, за которыми следует специальное совещание (дебрифинг) с участием всей команды. Мы обнаружили, что более эффективно проводить сессии интервью последовательно с участием одних и тех же проектировщиков. Тем самым размер команды остается небольшим (два или три человека) и, что гораздо важнее, вся команда получает возможность напрямую взаимодействовать с каждым из интервьюируемых пользователей, что позволяет ее членам максимально эффективно анализировать и синтезировать собранные данные.
- **Определение целей в самом начале.** Контекстное исследование, как оно описано Бейером и Хольцблат, рассчитано на процесс проектирования, принципиально ориентированный на задачи. При проведении этнографических интервью мы предлагаем прежде всего выявлять цели пользователей и назначать им приоритеты, и только затем приступать к выяснению того, какие задачи соответствуют этим целям.
- **Выход за пределы бизнес-контекста.** Лексикон контекстного исследования рассчитан на проектирование бизнес-продуктов, используемых в корпоративной среде. Проведение этнографических интервью возможно также и для потребительских продуктов, хотя, как будет видно в дальнейшем, направленность вопросов в этом случае становится несколько иной.

В оставшейся части главы представлены общие приемы и советы для подготовки и проведения этнографических интервью.

Подготовка к этнографическим интервью

Заимствованный из антропологии термин *«этнография»* обозначает систематическое изучение человеческих культур, подразумевающее погружение в эти культуры. Антропологи, занимающиеся этнографическими исследованиями, проводят годы в среде той культуры, которую они изучают и описывают. Этнографические интервью проникнуты философией подобных исследований и применяют ее на микроуровне. Они предназначены не для того, чтобы понять поведение и социальные ритуалы целой культуры, а для того чтобы понять поведение и ритуалы людей, взаимодействующих с конкретным продуктом.

Выявление кандидатов

Поскольку проектировщики должны охватить весь спектр поведения пользователей в отношении продукта, крайне важно при планировании серии интервью отобрать в нужной степени разнородную группу пользователей и типов пользователей. Основываясь на информации, полученной от заинтересованных лиц и ЭПО, а также в результате обзора литературы, проектировщики должны выдвинуть гипотезу, которая послужит точкой отсчета в определении того, каких именно пользователей и потенциальных пользователей необходимо интервьюировать.

Построение гипотезы о персонажах

Мы назвали отправную точку **гипотезой о персонажах**, поскольку создание такой гипотезы служит первым шагом на пути выявления и синтеза персонажей – архетипов пользователей, о которых мы поговорим подробно в следующей главе. Гипотезу о персонажах следует строить на вероятных шаблонах поведения и отличительных факторах этих шаблонов, а не просто на демографических данных. В случае потребительских продуктов демографические данные часто используются в качестве критерия отсева при подборе участников интервью, но и тут демографию необходимо рассматривать лишь как вспомогательное средство, дающее первое приближение к гипотезе о шаблонах поведения.

При формулировании гипотезы о персонажах большое значение имеет предметная область продукта. Пользователи бизнес-среды и обычные потребители зачастую сильно различаются мотивами и шаблонами поведения, поэтому в каждом случае используются свои методы построения гипотезы о персонажах.

Гипотеза о персонажах – первая попытка выделить различные типы пользователей (и иногда покупателей) продукта. Эта гипотеза служит основой для предварительного планирования интервью; при этом по мере реализации плана могут появиться данные, указывающие на существование таких типов пользователей, которые не были выявлены в самом начале, и тогда потребуются дополнительные интервью.

Гипотеза о персонажах пытается дать общие ответы на следующие три вопроса:

- Каковы категории людей, которые могут использовать данный продукт?
- Как могут различаться их потребности и поведение?
- Какие шаблоны поведения и типы рабочей обстановки необходимо изучить?

Роли для корпоративных и потребительских рынков

Для бизнес-продуктов **роли**, то есть общие наборы задач и информационных потребностей, связанные с отдельными классами пользователей, являются важным изначальным организующим принципом. Так, для офисной телефонной станции мы можем определить следующие примерные роли:

- люди, принимающие и совершающие звонки на своем рабочем месте;
- люди, которые много путешествуют и нуждаются в удаленном доступе к своей телефонной станции;
- секретари, принимающие звонки за многих людей;
- люди, занимающиеся техническим администрированием телефонной станции.

В технических и бизнес-контекстах роли зачастую примерно соответствуют описаниям должностных обязанностей, так что получить в первом приближении перечень типов пользователей, которых необходимо интервьюировать, несложно – достаточно понять, какие должности могут занимать пользователи (или потенциальные пользователи) системы.

В отличие от бизнес-пользователей потребители не имеют конкретных описаний должностей и часто используют продукт в разнообразных контекстах. Как следствие, для гипотезы персонажей потребительского продукта использование ролей в качестве организующей схемы не всегда оказывается осмысленным. Наиболее важные шаблоны выявляются скорее при анализе привычек и наклонностей, определяющих поведение пользователей.

Поведенческие и демографические переменные

Гипотеза о персонажах должна опираться не только на роли, но и на переменные, помогающие различать виды пользователей на основе их потребностей и поведения. Часто этот способ оказывается наиболее эффективным для разграничения типов пользователей и ложится в основу процесса создания персонажей, описываемого в следующей главе. Хотя предсказать эти поведенческие переменные заранее, не проводя исследований, бывает сложно, они часто становятся фундаментом гипотезы о персонажах для потребительских продуктов. К примеру,

в случае интернет-магазина мы можем выделить следующий ряд поведенческих шкал, относящихся к совершению покупок:

- частота совершения покупок (часто – нечасто);
- стремление делать покупки (обожает покупать – ненавидит покупать);
- мотивация для совершения покупок (поиск наилучшего по цене предложения – поиск наиболее подходящей вещи).

Комбинации значений поведенческих переменных часто задают в первом приближении типы пользователей в сфере потребления, однако поведенческие переменные важны и для выявления типов бизнес-пользователей и пользователей технических продуктов. Люди, подпадающие под одно определение бизнес-роли, могут иметь различную мотивацию и различные потребности. Поведенческие переменные позволяют отразить такие различия, хотя обычно для этого приходится сначала собрать данные о пользователях.

В связи с тем, что до сбора данных о пользователях точно предсказать поведенческие переменные сложно, часто применяется другой подход к построению гипотезы о персонажах – использование *демографических переменных*. При планировании интервью вы можете с помощью исследований рынка выявить распределение целевой аудитории по половозрастному и географическому составу, а также по уровню дохода. Участники интервью должны как можно шире распределяться по этим демографическим спектрам, чтобы в получившейся выборке респондентов оказались представлены все важные шаблоны поведения.

Компетентность в предметной области и техническая компетентность

Стоит отдельно упомянуть такой важный вид поведенческих различий, как различие между технической компетентностью (познания в цифровых технологиях) и компетентностью в предметной области (познания в специализированной предметной области, относящейся к продукту). Различные пользователи обладают разной степенью технической компетентности; аналогичным образом некоторые из пользователей продукта могут быть менее сведущими в соответствующей предметной области (к примеру, в вопросах бухгалтерии, когда речь идет о продукте для ведения бухгалтерского учета). Таким образом, в зависимости от того, на кого ориентирован продукт, может оказаться важна не только простота применения, но и поддержка предметной области. Неподготовленный пользователь, вероятно, никогда не сможет выйти за пределы небольшого подмножества специфичных для предметной области функций продукта, если интерфейс не обеспечивает полноценную поддержку предметной области. Если частью рынка продукта, ориентированного на определенную предметную область, являются неподготовленные пользователи, следует внимательно отнестись к поддержке наивных в отношении предметной области вариантов поведения.

Соображения, касающиеся рабочей среды

Последнее соображение, в большей степени касающееся бизнес-продуктов, относится к разнице в культурных особенностях тех организаций, где работают пользователи. В малых компаниях, как правило, личные контакты между сотрудниками сильнее, а круг обязанностей каждого сотрудника шире, тогда как в огромных компаниях существуют многочисленные бюрократические барьеры, а каждый сотрудник имеет узкую специализацию. Примерами таких средовых переменных могут служить:

- размер компании (малая – транснациональная);
- география компании (Северная Америка, Европа, Азия и т. д.);
- отрасль/сектор (производство электроники, товары массового потребления и т. п.);
- использование информационных технологий (применяются произвольно – имеются «драконовские» стандарты);
- уровень обеспечения безопасности (низкий – высокий).

Как и в случае с поведенческими переменными, выявление средовых переменных может оказаться затрудненным без исследований предметной области, поскольку на культурные шаблоны оказывают большое влияние такие параметры, как отрасль промышленности и географическое положение.

Планирование

После того как вы сформулировали гипотезу о персонажах, содержащую описание потенциальных ролей, а также поведенческих, демографических и средовых переменных, необходимо составить план проведения интервью, который можно передать человеку, отвечающему за организацию встреч с пользователями.

Из нашего опыта следует, что для проверки каждого потенциального поведенческого шаблона требуется примерно шесть интервью (а в особо сложных предметных областях и больше). Это означает, что каждая роль или переменная (поведенческая, демографическая или средовая), обозначенная в гипотезе о персонажах, исследуется в четырех-шести интервью (и, соответственно, более, если речь идет о сложной предметной области).

Однако эти интервью могут иметь пересечения. Если мы предполагаем существование различий в применении бизнес-продукта, зависящих от географии, отрасли и размера компании, исследование на фабрике небольшого производителя электроники на Тайване позволит нам изучить сразу несколько переменных. Вдумчивое соотнесение переменных с анкетными данными кандидатов позволяет удержать число требуемых интервью в разумных пределах.

Проведение этнографических интервью

Когда гипотеза о персонажах сформулирована и на ее основе создан план интервью, вы готовы к интервьюированию – разумеется, если у вас уже есть доступ к респондентам! В процессе формирования плана проведения интервью проектировщикам следует работать в тесном контакте с заинтересованными в проекте лицами, которые имеют доступ к пользователям. Обычно вовлечение заинтересованных лиц – лучший способ сделать так, чтобы интервью состоялись, особенно в случае с техническими и бизнес-продуктами.

Если заинтересованные лица неспособны помочь в налаживании контактов с пользователями, вы можете обратиться в компанию, проводящую исследования рынка или юзабилити-исследования и специализирующуюся на поиске людей для анкетирования и участия в фокус-группах. Такие фирмы полезны, когда необходимо связаться с покупателями из различных демографических групп. Сложность этого подхода в том, что иногда оказывается непросто найти людей, которые согласятся на интервью у себя дома или на рабочем месте.

В случае с потребительскими продуктами в качестве последнего варианта проектировщики могут привлекать своих друзей и родственников. Это упрощает наблюдение за респондентами в естественной обстановке, но вместе с тем накладывает ограничения по демографическим и поведенческим переменным.

Команды интервьюеров и координация

При проведении интервью авторы отдают предпочтение командам из двух проектировщиков: один управляет ходом интервью и делает простые заметки, а другой ведет подробные записи (в середине интервью проектировщики могут меняться ролями). Часового интервью на каждого пользователя обычно достаточно, за исключением случаев очень сложных предметных областей (медицина, наука, финансовые службы), когда для более полного понимания конечных результатов работы пользователя может потребоваться больше времени. Не забудьте заложить в график запас времени на переезды из одной точки проведения интервью в другую, особенно если речь идет об интервьюировании потребителей в спальных районах или о «слежке» за пользователями в процессе взаимодействия с продуктом (обычно мобильным) при перемещении из одного места в другое. Следует ограничиваться шестью интервью в день, чтобы между ними оставалось достаточно времени на внутреннее обсуждение и выработку стратегических планов дальнейших интервью, а также чтобы избежать переутомления интервьюеров.

Фазы этнографических интервью

Полный набор этнографических интервью в проекте можно разбить на три отдельные группы по хронологическим фазам. Подход к интервью в каждой следующей фазе несколько отличается от подхода в преды-

дущей, отражая растущее с каждым новым интервью знание поведения пользователей. Поначалу интервью направлены на разрешение вопросов, связанных с общей структурой предметной области и целями пользователей, и тематический охват весьма широк; в конце цикла он сужается, и исследование сосредотачивается на вопросах, связанных с конкретными функциями и задачами.

- **Первичные интервью** по своей природе служат для «разведки территории» и концентрируются на сборе знаний о предметной области, как ее видит пользователь. Преобладают широкие открытые вопросы, погружение в детали практически отсутствует.
- **Уточняющие интервью** – этап, на котором проектировщики начинают осознавать основные шаблоны использования продукта и задают открытые уточняющие вопросы, помогающие собрать из фрагментов целостную картину. Вопросы в целом больше сосредоточены на специфике предметной области, поскольку к этому моменту проектировщики уже усвоили ее основные правила, структуру и терминологию.
- **Завершающие интервью** дают подтверждение ранее выявленным шаблонам использования и дополнительно проясняют пользовательские роли и варианты поведения, корректируют предположения о задачах и информационных потребностях. Закрытых вопросов здесь гораздо больше – с их помощью проясняются все нестыковки в собранных данных.

После того как вы составили представление о людях, которых будете интервьюировать, может оказаться полезным поработать в контакте с заинтересованными лицами, чтобы распределить респондентов по этапам наиболее подходящим образом. Так, в технически сложной предметной области разумно проводить первичные интервью с наиболее терпеливыми респондентами, хорошо умеющими формулировать свои мысли. Иногда в конце полного цикла интервью у вас может возникнуть желание вернуться к общению с кем-то из этих людей, чтобы обсудить темы, которые всплыли уже после проведения первых интервью.

Основные приемы

Основные приемы этнографического интервьюирования просты, прямолинейны и малотехнологичны. Хотя овладение нюансами интервьюирования потребует некоторого времени, любой специалист, который последует приводимым ниже советам, соберет щедрый урожай полезных качественных данных.

- Проводите интервью там, где происходит взаимодействие пользователя с продуктом.
- Избегайте жесткого следования предопределенным наборам вопросов.
- Сначала концентрируйтесь на целях – и лишь потом на задачах.

- Не делайте из пользователя проектировщика.
- Избегайте дискуссий по технологическим вопросам.
- Поощряйте пользователей рассказывать истории.
- Просите показывать и рассказывать.
- Избегайте наводящих вопросов.

В следующих подразделах мы подробно описываем каждый из этих приемов.

Проводите интервью там, где происходит взаимодействие пользователя с продуктом

В соответствии с первым принципом контекстного исследования проведение интервью в том месте, где человек на самом деле использует продукт, имеет решающее значение. Интервьюеры получают не только возможность присутствовать при использовании продукта, но и доступ к той среде, в рамках которой происходит взаимодействие. Это может привести к неожиданным открытиям в том, что касается ограничений продукта, а также потребностей и целей пользователя.

Внимательно анализируйте рабочую среду – она с большой вероятностью переполнена подсказками, указывающими на какие-либо задачи, о которых респондент мог и не упомянуть. В частности, обращайте внимание на следующие вещи: информация, необходимая человеку (бумаги на столе или клейкие листочки по периметру монитора); свидетельства неадекватности систем («шпаргалки» и пользовательские руководства); частота и приоритетность задач (папки входящих и исходящих); используемые виды отображения рабочих процессов (памятки, графики, календари). Не суйте нос без разрешения, однако если увидите что-то интересное, попросите респондента обсудить это с вами.

Избегайте жесткого следования предопределенным наборам вопросов

Если подойти к этнографическому интервью так, словно это анкетирование, вы рискуете не только вызвать отчуждение респондента, но упустить ценнейшие сведения. Суть этнографического интервьюирования (и контекстного исследования) в том и состоит, что мы как интервьюеры знаем о предметной области недостаточно, чтобы заранее предполагать какие-либо вопросы. О том, что важно, а что нет, мы должны узнать у людей, с которыми общаемся. При этом, несомненно, полезно иметь некое представление о *типах* вопросов. В некоторых предметных областях может принести пользу стандартизированный набор *тем*, которые желательно охватить в ходе интервью. Перечень тем может корректироваться от встречи к встрече, но само его наличие поможет обеспечить достаточно хорошую детализацию информации в каждом интервью и упростит выявление значимых шаблонов поведения.

Вот некоторые заслуживающие внимания **целеориентированные вопросы**:

- *Цели*: что делает ваш день хорошим? а плохим?
- *Возможности*: какие виды деятельности сейчас отнимают у вас время?
- *Приоритеты*: что для вас наиболее важно?
- *Информация*: что помогает вам принимать решения?

Другой полезный тип вопросов – **системоориентированные вопросы**:

- *Функция*: что вы чаще всего делаете при помощи этого продукта?
- *Частота*: какими модулями продукта вы пользуетесь чаще всего?
- *Предпочтения*: какие стороны этого продукта вам особенно полюбились? что вызывает восхищение?
- *Отказы системы:* как вы справляетесь с проблемами в функционировании системы?
- *Профессиональность*: какие приемы вы используете для ускорения работы?

Для бизнес-продуктов могут быть полезны **вопросы, ориентированные на рабочий процесс**:

- *Процесс*: что вы сделали сегодня на работе первым делом? а что после этого?
- *Повторяющиеся действия и их регулярность*: как часто вы это делаете? что вы делаете еженедельно и каждый месяц, но не каждый день?
- *Исключения*: каков типичный рабочий день? что стало бы необычным событием?

Чтобы лучше понять мотивацию пользователей, можно применять **вопросы, ориентированные на мировоззрение и отношение к окружающему**:

- *Устремления*: чем бы вы хотели заниматься через пять лет?
- *Избегание*: что вы предпочли бы не делать? что откладываете?
- *Мотивация*: что вам больше всего нравится в вашей работе (или жизни)? какие вопросы вы всегда решаете в первую очередь?

Сначала концентрируйтесь на целях – и лишь потом на задачах

В отличие от контекстных интервью и большинства других методов качественных исследований, в этнографическом интервьюировании на первом месте стоит необходимость понять, *почему* и *как* действуют пользователи (что движет поведением людей в различных ролях и каким образом они рассчитывают в конечном итоге достичь цели), а не *что* за задачи они решают. Понимание задач тоже важно – и их необходимо тщательно фиксировать. Однако в результате проектирования

эти задачи в конце концов претерпят изменения, чтобы лучше соответствовать целям пользователей.

Не делайте из пользователя проектировщика

Направляйте респондента в сторону формулирования проблем и уводите его от формулирования конкретных интерфейсных решений. Такие решения, как правило, несут на себе отпечаток приоритетов пользователя: *ему* они представляются хорошими, однако на практике оказываются односторонними и недальновидными, лишенными того равновесия и той проработанности, которых способен достичь проектировщик, располагающий результатами исследований и обладающий многолетним опытом. Предложенное пользователем решение можно использовать в качестве отправной точки для обсуждения целей пользователя и проблем, с которыми он сталкивается в существующих системах. Если у пользователя проскользнет интересная идея, спросите, какую из его проблем это решает или почему он считает это решение хорошим.

Избегайте дискуссий по технологическим вопросам

Не стоит делать из пользователя не только проектировщика, но также программиста или инженера. Обсуждение технических деталей лишено смысла, если отсутствует понимание стоящих за техническими решениями целей. В случае технических или научных продуктов, для которых технологические вопросы неизбежно оказываются существенными, следует различать технологию, связанную с предметной областью, и технологию, относящуюся к продукту, – и держаться подальше от обсуждения последней. Если респондент настаивает на обсуждении того, как должен быть реализован продукт, переведите разговор на его цели и мотивы вопросом о том, как это поможет ему.

Поощряйте пользователей рассказывать истории

Просить у пользователей рекомендаций в области дизайна – занятие гораздо менее продуктивное, нежели побуждать их рассказывать конкретные истории о своем опыте работы с продуктом (со старой версией продукта, перепроектированием которого вы занимаетесь, или с аналогичным продуктом): каким образом они его используют, что о нем думают, с кем общаются при работе с ним, куда берут продукт с собой – и т. д. Подробные рассказы такого рода – один из лучших способов понять, как пользователи уживаются с продуктами и взаимодействуют с ними. Старайтесь поощрять рассказы как о типичных случаях, так и об исключительных.

Просите показывать и рассказывать

Когда прочие вопросы исчерпаны и у вас сложилось ясное представление о процессах и структуре деятельности пользователя и его взаимодействия с продуктом, часто бывает полезно попросить пользователя

рассказать и показать, или **продемонстрировать** что-либо, относящееся к решаемой задаче проектирования. Это может быть что угодно – объекты, связанные с предметной областью, интерфейсы программ, традиционные «бумажные» системы, экскурсии по месту работы, а в идеале – все вышеперечисленное. Старайтесь не просто фиксировать все, что вам показывают (здесь будут весьма кстати цифровые фото- и видеокамеры), но также обращать внимание на то, *как* респондент это описывает. Не забывайте про уточняющие вопросы.

Избегайте наводящих вопросов

В интервью очень важно избегать *наводящих вопросов*. Подобно тому, как в зале суда адвокаты способны давить на свидетелей своим авторитетом, склоняя их к нужным ответам, проектировщики могут непреднамеренно подтолкнуть респондента, явным или неявным образом предложив решение либо мнение о поведении. Вот некоторые примеры наводящих вопросов:

- Вам бы помогла функция X?
- Вам нравится X, верно?
- Как вы думаете, стали бы вы пользоваться X, если бы она была доступна?

После интервью

После каждого интервью команды сопоставляют заметки и обсуждают наиболее интересные тенденции или конкретные моменты, обнаружившиеся в ходе последнего интервью. Если осталось время, следует также просмотреть старые записи, чтобы выяснить, получены ли ответы на неразрешенные вопросы из прежних интервью и исследований. Эта информация должна быть задействована при планировании последующих интервью.

Когда цикл интервью завершен, полезно еще раз пройтись по всем записям, отмечая или подчеркивая тенденции и шаблоны в полученных данных. Это подготовка к следующему шагу – созданию персонажей на основе данных всех исследований. Если команда сочтет это полезным, можно создать подборку из заметок, разбора видеозаписей, а также распечаток сопутствующих изображений и поместить ее на видном месте – скажем, на стене, чтобы все объекты были видны одновременно. Такая «раскадровка» будет полезна на последующих стадиях проектирования.

Прочие виды исследований

В этой главе речь шла главным образом о качественных исследованиях, ориентированных на сбор пользовательских данных, которые впоследствии используются (как описано в следующей главе) для конструирования достоверных моделей пользователей и предметной облас-

ти – ключевых инструментов в методологии целеориентированного проектирования. Эксперты в сфере проектирования и юзабилити применяют широкий спектр других методов исследований – от подробного анализа рабочих заданий до проведения фокус-групп и юзабилити-тестов. Хотя многие из этих видов деятельности обладают несомненным потенциалом в деле создания полезных и желанных продуктов, мы выяснили, что при проектировании цифровых продуктов наиболее ценный вклад в процесс дает качественный подход, описываемый в этой главе. Коротко говоря, этот подход позволяет с минимальными затратами сил и денег получить ответы как на общие, высокоуровневые, так и на частные, детальные вопросы о продукте. Ни один другой подход не может похвастать такими результатами.

Книга Майка Кунявски (Mike Kuniavsky) «Observing the User Experience» (Kuniavsky, 2003) может послужить отличным путеводителем по разнообразным методам исследования пользователей, применимым на различных стадиях процесса проектирования и разработки. В оставшейся части главы мы рассмотрим лишь некоторые наиболее заметные методы исследований и их вклад в процесс разработки.

Фокус-группы

В маркетинговых организациях особенно популярен метод сбора пользовательских данных посредством **фокус-групп**, в котором репрезентативная группа пользователей, отобранных по заранее выявленным демографическим параметрам, собирается в одной комнате и отвечает на структурированный набор вопросов и/или выбирает ответы из предложенных вариантов. Часто встреча записывается на диктофон или видеокамеру для последующего анализа. Фокус-группы – стандартный прием в маркетинге традиционных продуктов. Они полезны для оценки непосредственных реакций на *форму* продукта, его внешний вид или технический дизайн. Фокус-группы также позволяют оценивать реакции на продукт, которым респонденты уже пользовались в течение определенного времени.

Хотя может возникнуть ощущение, что фокус-группы дают необходимый контакт с пользователями, этот метод во многих случаях малопригоден в качестве инструмента проектирования. Фокус-группы хороши, когда нужно получить информацию о продуктах, которыми люди владеют или которые желали бы (либо не желали бы) приобрести, но не очень подходят для сбора данных о том, что люди делают с этими продуктами, а также как и почему они это делают. Кроме того, фокус-группы в силу своей коллективной природы имеют свойство приводить к консенсусу: мнением группы становится мнение, высказанное большинством или озвученное громче других. А это уже в корне противоречит сути процесса проектирования, поскольку проектировщикам необходимо уловить все особенности поведения, чтобы учесть их при разработке продукта. Фокус-группы, как правило, выхолащива-

ют то разнообразие поведения и мнений, которое как раз и должны принимать в расчет проектировщики.

Демография рынка и сегменты рынка

Маркетинг как род деятельности существенно преуспел в систематическом выявлении стимулов, побуждающих людей покупать. Одним из наиболее мощных инструментов в этой области является сегментирование рынка, разделяющее потенциальных потребителей на классы по демографическим признакам (возраст, пол, уровень образования, почтовый индекс и т. п.) на основании данных фокус-групп и исследований рынка, чтобы определить те группы потребителей, которые будут наиболее восприимчивы к определенному продукту или рекламному сообщению. Более сложные модели пользовательских данных включают также психографические и поведенческие переменные, такие как взгляды и привычки, стиль жизни, ценности, идеология, стремление к стабильности, шаблоны принятия решений. Такие системы классификации, как сегментация VALS (SRI) и геодемографические кластеры PRIZM Джонатана Робина (Jonathan Robbin), способны привнести существенную ясность в данные, предсказывая покупательские способности потребителей, мотивацию, степень ориентации на себя, ресурсы.

Эти приемы рыночного моделирования позволяют точно прогнозировать готовность рынка принять определенный продукт или услугу. Они служат бесценным инструментом для оценки *жизнеспособности* продукта. Они могут стать мощным средством воздействия на тех, кто принимает решение о запуске продукта в производство. Наконец, зная, что X людей могут купить продукт или услугу за Y долларов, вы можете легко оценить потенциальную окупаемость вложений.

Однако представление о том, хочет ли человек купить вещь, не тождественно ответу на вопрос, какой она должна быть. Сегментирование рынка – отличное средство для выявления и оценки рыночных возможностей, но слабый инструмент для определения вида и поведения продуктов, использующих эти возможности для извлечения прибыли.

На практике оказывается, что данные, собранные при исследованиях рынка, и данные, добытые в ходе качественных пользовательских исследований, достаточно хорошо дополняют друг друга. Поскольку исследования рынка способны помочь в обнаружении благоприятных рыночных возможностей, они часто с неизбежностью оказываются отправной точкой для принятия решения о начале проектировании. Без оценки этих возможностей будет очень трудно убедить бизнесмена финансировать проект. Кроме того, как мы уже говорили, при проведении этнографических интервью исследования рынка упрощают отбор кандидатов. Наконец, рассказанная выше история о программе для видеомонтажа демонстрирует, что качественные исследования способны представить результаты количественных исследований под

очень важным углом зрения. Различия между моделями сегментирования и пользовательскими моделями мы рассмотрим более подробно в главе 5.

Юзабилити и пользовательское тестирование

Юзабилити-тестирование (или, как его еще не очень удачно называют, пользовательское тестирование) – это набор методик, позволяющих измерить характеристики взаимодействия пользователя с продуктом с целью оценки уровня юзабилити продукта. Как правило, в ходе юзабилити-тестирования изучается, насколько хорошо пользователи выполняют конкретные стандартизированные задачи и с какими проблемами они при этом сталкиваются. Результаты такого тестирования часто помогают выявить как аспекты, затрудняющие понимание и использование продукта, так и удачные решения.

Для проведения юзабилити-тестирования требуется, чтобы предмет тестирования обладал относительной завершенностью и внутренней согласованностью. Смысл тестирования состоит в проверке качества интерфейса продукта – независимо от того, тестируется ли готовое программное обеспечение, минимально функциональный макет или же вовсе бумажный прототип. Отсюда вытекает, что юзабилити-тестирование должно проводиться на поздних стадиях цикла проектирования, когда уже существует связная концепция и имеется достаточное число деталей, чтобы можно было создать такого рода макет или прототип. Мы обсудим так называемое полное юзабилити-тестирование как часть стадии детализации в главе 7.

Можно привести определенные доводы в пользу уместности юзабилити-тестирования на ранних стадиях перепроектирования существующего продукта. В таких проектах эта методика действительно позволяет выявить аспекты продукта, нуждающиеся в улучшении. Однако мы выяснили, что крупные недостатки продукта можно обнаружить посредством наших качественных исследований, и, если бюджет проекта не позволяет выполнить более одного юзабилити-тестирования, гораздо полезнее использовать его как средство проверки конкретных элементов нового дизайна, когда будет готово решение-кандидат.

Поскольку результаты пользовательского тестирования обычно измеримы и поддаются количественному выражению, исследования юзабилити продукта особенно ценны при сравнении конкретных вариантов дизайна с целью отбора наиболее эффективного решения. Собранные в ходе юзабилити-тестирования отзывы потребителей наиболее полезны, когда вы хотите проверить либо усовершенствовать механизмы взаимодействия или форму и реализацию определенных элементов продукта.

Вот аспекты продукта, для оценки которых юзабилити-тестирование особенно эффективно:

- **Наименование**. Осмысленны ли названия разделов и надписи на кнопках? Возможно, какие-то из этих слов воспринимаются легче, чем другие?
- **Архитектура**. Осмысленно ли информация разбита на категории? Расположены ли информационные элементы в тех местах, где их ожидают найти потребители?
- **Первое знакомство и доступность**. Легко ли новые пользователи находят базовые элементы интерфейса? Понятны ли инструкции? Есть ли в них необходимость?
- **Эффективность**. Могут ли потребители эффективно решать конкретные задачи? Ошибаются ли они? При выполнении каких шагов? Как часто?

Из сказанного видно, что юзабилити-тестирование сосредоточено преимущественно на оценке первого опыта использования продукта. Зачастую очень сложно (и всегда трудоемко) измерять эффективность решения при пятидесятом использовании продукта, то есть, другими словами, для самой интересной целевой аудитории – вечных середняков. Это порождает определенные трудности при оптимизации интерфейса для середняков и экспертов. Один из методов преодоления этих трудностей носит название *дневниковые исследования*: испытуемые ведут дневники с подробными записями о своем взаимодействии с продуктом. Хорошее описание этого метода дается в книге Майка Кунявски «Observing the User Experience» (Kuniavsky, 2003).

Наконец, при проведении юзабилити-тестирования следует убедиться, что вы тестируете то, что можно измерить, что тестирование выстроено корректно, что результаты будут полезны для выявления проблем проектирования и у вас есть ресурсы, необходимые для исправления этих проблем. Отличным руководством по теме юзабилити-тестирования служит классическая работа Якоба Нильсена (Jakob Nielsen) «Usability Engineering» (Nielsen, 1993).

Карточная сортировка

Карточная сортировка – техника, снискавшая популярность благодаря информационным архитекторам. Она позволяет понять, как пользователи организуют идеи и информацию. Существует ряд вариантов этой техники, но обычно она сводится к тому, что пользователей просят выполнить сортировку колоды карт, каждая из которых описывает определенную функциональность продукта или веб-сайта либо содержит связанный с ним фрагмент информации. Сложной частью является анализ результатов: необходимо найти паттерны и зависимости путем выявления тенденций или посредством статистического анализа.

Карточная сортировка, несомненно, может стать ценным инструментом для раскрытия определенных аспектов пользовательской ментальной модели, однако эта техника предполагает, что респондент об-

ладает хорошими навыками организации информации и то, как он сортирует набор абстрактных тем, напрямую связано с подходом, который он в конце концов выберет, когда пожелает воспользоваться вашим продуктом. Очевидно, так получается не всегда. Один из способов преодолеть возможное расхождение – попросить пользователя упорядочивать карты, подготовленные на основе способов выполнения задач, решать которые призван проектируемый продукт. Другой способ повысить ценность карточных исследований – после завершения процедуры побеседовать с респондентами, чтобы выявить принципы организации информации, которые они применяли при сортировке (памятуя, что вы пытаетесь понять их ментальную модель).

В конечном итоге мы считаем, что должным образом построенные открытые интервью вполне способны раскрыть и эти аспекты ментальной модели пользователей. Задавая правильные вопросы и внимательно изучая разъяснения пользователя, касающиеся его деятельности и предметной области, вы можете расшифровать умозрительные связи между отдельными функциональными и информационными элементами.

Анализ рабочих заданий

Под анализом рабочих заданий понимается набор методик, задействующих анкетирование или открытые интервью для формирования детального представления о том, как люди в настоящий момент выполняют конкретные задания. В таком исследовании нас интересуют следующие вопросы:

- стоящая за заданием реальная цель – для чего пользователь выполняет задания;
- частота и важность выполнения заданий;
- триггеры – что служит поводом или сигналом для выполнения задания;
- зависимости – что требуется для выполнения задания и что зависит от ее выполнения;
- люди, которые вовлечены в выполнение задания, их роли и зоны ответственности;
- конкретные действия, которые требуется выполнить;
- решения, которые необходимо принять;
- информация, которая нужна для принятия решений;
- ошибки и исключительные ситуации – что может пойти не так;
- способы исправления ошибок и обработки исключений.

После того как заполнены анкеты или проведены интервью, выполняется формальная декомпозиция и анализ рабочих заданий – как правило, при помощи диаграммы потоков или сходной диаграммы, пере-

дающей отношения между действиями и зачастую отношения между людьми и процессами.

Мы обнаружили, что такого рода опросы стоит включать в этнографические интервью. Анализ, как мы расскажем в следующей главе, – также важная часть нашей деятельности по моделированию. Однако, хотя анализ заданий является важным для понимания того, как именно пользователи что-то делают в настоящий момент, и позволяет выявлять неудобства и возможности для улучшений, мы хотели бы еще раз подчеркнуть приоритет целей пользователей. То, как люди выполняют задания сейчас, зачастую есть лишь наследие устаревших систем, с которыми они вынуждены работать. Обычно их образ действий далек от эффективности и имеет мало общего с теми способами, которыми люди предпочли бы пользоваться.

Пользовательские исследования – фундамент проектирования. Уделяйте планированию исследований достаточное время. Подбирайте методы исследования сообразно этапам разработки. Вашему продукту это пойдет на пользу, а вы сможете избежать потерь времени и ресурсов. Лабораторное тестирование продукта может дать большой объем информации, но не обязательно будет иметь большую ценность. Этнографическое интервьюирование в начале процесса позволяет вам как проектировщику глубоко понять своих пользователей, их потребности и мотивы. Когда ваша концепция продукта построена на основе качественных пользовательских исследований и моделей, опирающихся на результаты этих исследований, юзабилити-тестирование становится еще более эффективным инструментом для оценки решений, принятых в ходе проектирования. Качественные исследования позволяют взять хороший старт уже в самом начале проекта.

5

Модели пользователей: персонажи и цели

Чтобы понять жизнь наших пользователей, их мотивацию и среду обитания, мы погружаемся в их мир – и здесь сталкиваемся с серьезным вопросом: как использовать данные исследований для проектирования успешного продукта? У нас в руках блокноты, полные расшифровок бесед и наблюдений, причем, по всей вероятности, каждый человек, с которым мы общались, несколько отличался от других. Трудно вообразить, что каждый раз при выработке проектного решения мы будем просматривать сотни страниц заметок. Но даже если мы найдем на это время, остается совершенно непонятным, как эти заметки могут нам помочь.

Мы решаем эту проблему посредством мощной концепции **моделей**. Модели применяются в естественных и общественных науках для представления сложных явлений посредством абстрагирования. Подобно тому, как экономисты создают модели, описывающие поведение рынков, а физики – модели, описывающие поведение частиц, мы создаем описательные модели пользователей на основе исследований и получаем таким образом уникальный, мощный инструмент для проектирования взаимодействия. Эти модели пользователей мы называем **персонажами**.

Персонажи дают нам строгие методы для обдумывания и описания поведения пользователей, их образа мыслей, их устремлений и причин этих устремлений. Персонажи – не настоящие люди, но они создаются на основе мотивации и поведения реальных людей, которых мы наблюдали, и выступают в роли представителей реальных людей в процессе проектирования. Это *составные архетипы*, создаваемые на основе поведенческих данных, собранных в ходе этнографических интервью со многими реальными пользователями. Основой для персона-

жей являются *поведенческие шаблоны, которые мы наблюдаем* на стадии исследований и формализуем на стадии моделирования. С помощью персонажей мы приходим к пониманию целей пользователей в конкретных ситуациях, что абсолютно необходимо для преобразования данных о пользователях в проектные решения.

Персонажи, как многие другие мощные инструменты, просты по замыслу, но должны применяться с известным вниманием к деталям. Недостаточно просто на скорую руку слепить из стереотипов и обобщений пару пользовательских профилей или прицепить фотографию из фотобанка к описанию должности и назвать результат «персонажем». Чтобы персонажи стали эффективным инструментом проектирования, нужно со всей возможной строгостью и тщательностью подходить к процессу выявления значимых и осмысленных шаблонов в поведении пользователей и последующего преобразования этих шаблонов в архетипы, представляющие значимые подмножества целевой аудитории.

Есть, конечно, и другие полезные виды моделей, которые могут служить инструментами проектировщика взаимодействия, скажем модели рабочих потоков и физические модели, но мы считаем, что персонажи – наиболее мощный инструмент, который к тому же позволяет использовать лучшие приемы моделирования из других моделей. Эта глава посвящена персонажам и целям; прочие модели мы кратко обсудим в конце главы.

Для чего нам модели?

Модели широко применяются в проектировании, в разработке и в науке. Это мощные инструменты для представления сложных структур и связей с целью их лучшего понимания, обсуждения и визуализации. В отсутствие моделей мы были бы вынуждены осмысливать неструктурированные, сырые данные, не имея подспорья в виде каких-либо организующих принципов. Хорошие модели подчеркивают характерные особенности структур и связей и уводят в тень менее важные детали.

Поскольку мы проектируем для пользователей, важно суметь понять и визуализировать характерные аспекты их отношений друг с другом, отношений с социальной и физической средой и, конечно, с продуктом, который мы намереваемся спроектировать.

Подобно тому как физики разрабатывали модели атома, опираясь на данные наблюдений и интуитивно выявляя общие зависимости, проектировщики должны создавать модели пользователей на основе данных наблюдений за их поведением, интуитивно выявляя общие зависимости в этих данных. Лишь после формализации найденных зависимостей можно надеяться на систематическое конструирование таких шаблонов взаимодействия пользователя с продуктом, которые

будут хорошо соответствовать поведению, ментальным моделям и целям пользователей. Такую формализацию обеспечивают персонажи.

Персонажи

Логика может подсказывать вам, что для создания продукта, который призван удовлетворить самую широкую аудиторию пользователей, нужно сделать его функциональность как можно более широкой, чтобы приспособить его к нуждам большинства людей. *Однако эта логика порочна*. Лучший способ успешно удовлетворить потребности широкой аудитории – проектировать для *конкретных типов людей с конкретными потребностями*.

Беспорядочно наращивая функциональность продукта, чтобы охватить как можно более широкую аудиторию, вы увеличиваете когнитивную нагрузку на всех пользователей и осложняете им ориентирование в пределах продукта. Возможности, которые понравятся одним пользователям, вероятно, войдут в противоречие с потребностями других (рис. 5.1).

Решение состоит в том, чтобы сначала правильным образом выбрать представителей, чьи потребности лучше всего отражают интересы большей части интересующей нас аудитории (см. рис. 5.2), а затем назначить приоритеты этим представителям так, чтобы учет потребностей самых важных пользователей не приносил существенных неудобств второстепенным пользователям. Персонажи представляют собой мощный инструмент для обсуждения различных типов пользователей и их потребностей, а также для выбора наиболее важных пользователей, на которых следует ориентироваться при проектировании функциональности и поведения продукта.

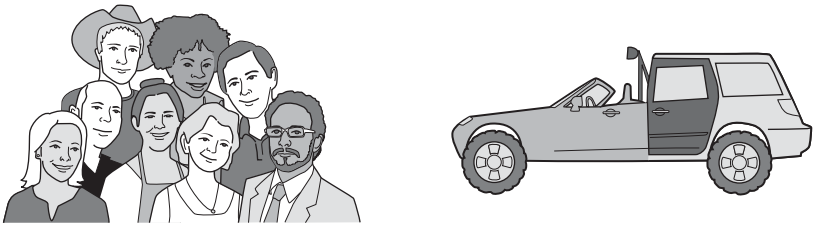


Рис. 5.1. *Упрощенный пример того, насколько полезны персонажи. Пытаясь спроектировать автомобиль, способный понравиться любому водителю, вы получите машину, обладающую всеми возможными особенностями, которая не понравится никому. Программное обеспечение сейчас слишком часто проектируют, пытаясь удовлетворить чересчур широкую аудиторию, что приводит к низкой удовлетворенности пользователей. На рис. 5.2 представлен альтернативный подход*

После того как персонажи были представлены в качестве инструмента моделирования пользователей в книге «The Inmates Are Running The Asylum»[1] (Cooper, 1999), они приобрели огромную популярность в сообществе людей, занимающихся проектированием опыта взаимодействия, но вместе с тем стали предметом некоторого недопонимания. Нам хотелось бы прояснить и более подробно изложить некоторые понятия и соображения, касающиеся персонажей.

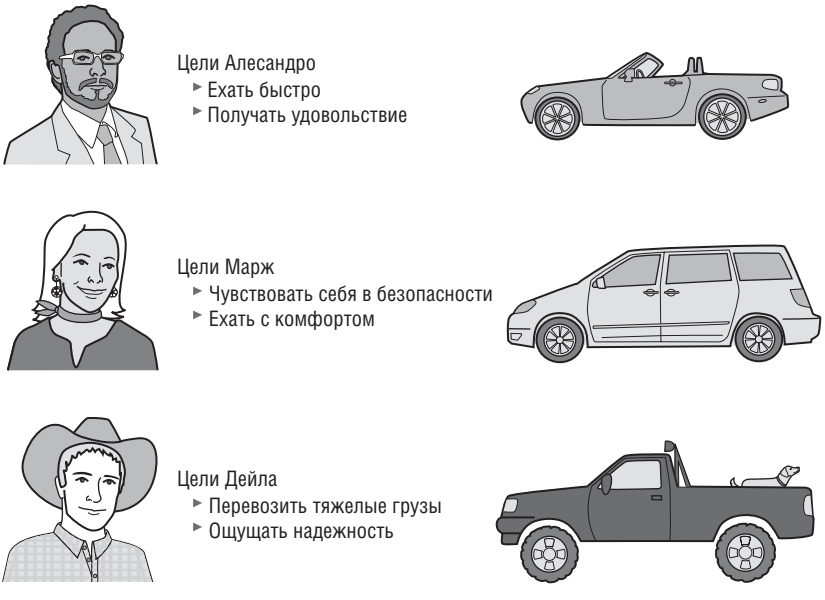


Рис. 5.2. *Упрощенная демонстрация полезности персонажей. Проектируя для разных людей с разными целями разные автомобили, мы способны создавать машины, удовлетворяющие и других людей, потребности которых аналогичны потребностям «целевых» водителей. То же самое верно для проектирования цифровой техники и программного обеспечения*

Преимущества персонажей как средства проектирования

Персонажи – мощное и универсальное средство проектирования, способствующее преодолению ряда проблем, которые в настоящее время неотступно преследуют разработчиков цифровых продуктов. Персонажи помогают проектировщикам:

- **Определять**, что должен делать продукт и каким должно быть его поведение. Цели и задачи персонажей образуют фундамент для проектирования.

[1] А. Купер «Психбольница в руках пациентов. Почему высокие технологии сводят нас с ума и как восстановить душевное равновесие», дополненное издание. – Пер. с англ. – СПб: Символ-Плюс, 2009.

- **Общаться** с заинтересованными лицами, разработчиками и другими проектировщиками. Персонажи задают общий язык для обсуждения проектных решений, а также помогают удерживать фокус на пользователях на каждом шаге процесса.
- **Достигать взаимопонимания и согласия** в вопросах проектирования. С общим языком приходит и общее понимание. Персонажи сокращают потребность в подробных моделях-диаграммах: многочисленные нюансы поведения пользователей проще понять благодаря повествовательным средствам, на которых основана работа с персонажами. Проще говоря, поскольку персонажи напоминают живых людей, они воспринимаются легче, чем диаграммы и списки функций.
- **Оценивать** эффективность решений. На персонажах можно испытывать проектные решения в процессе их формирования так, словно вы показываете их реальным пользователям. Хотя этот метод не отменяет необходимость тестирования на реальных пользователях, он является мощным средством проверки найденных при проектировании решений на соответствие реальности. Это позволяет быстро и недорого выполнять итерации проектирования, пользуясь лишь набросками на доске, и при этом подойти к моменту тестирования на реальных пользователях с более сильными интерфейсными решениями.
- **Вносить вклад** в работу других подразделений, связанных с продуктом, например в планы по маркетингу и продажам. Авторы сталкивались с тем, что клиенты начинают применять персонажей внутри своей организации в иных целях, например в качестве источника информации для маркетинговых кампаний, развития структуры организации и других задач стратегического планирования. Подразделения, не имеющие непосредственного отношения к разработке продукта, жаждут подробной информации о пользователях продукта и обычно проявляют к персонажам огромный интерес.

Кроме того, персонажи позволяют решить три ключевые проблемы проектирования, возникающие при разработке продукта. Вот эти проблемы:

- Проблема пластилинового пользователя
- Проектирование под себя
- Проектирование в расчете на исключительные ситуации

Мы коротко опишем каждую из них в последующих подразделах.

Проблема пластилинового пользователя

Наша цель состоит в удовлетворении пользователя, однако слово «*пользователь*» вызывает проблемы при решении конкретных вопросов в конкретных контекстах проектирования. Нечеткость термина делает его опасным в качестве средства проектирования: у каждого

человека в команде разработчиков есть свое представление о пользователе и его потребностях. Когда наступает момент принятия решений по продукту, этот «пользователь» становится *пластилиновым*: он всегда прогибается так, чтобы соответствовать мнениям и предположениям конкретного говорящего.

Если разработчикам удобно использовать для представления информации устрашающий элемент управления с деревом папок, они определяют пластилинового пользователя как сведущего в компьютерах «продвинутого» пользователя. Когда же им удобнее пошагово провести пользователя через сложный процесс при помощи мастера (wizard), они определяют пластилинового пользователя как неподготовленного новичка. Проектируя в расчете на пластилинового пользователя, разработчик сам себе выдает индульгенцию на то, чтобы программировать продукт так, как ему заблагорассудится, не переставая при этом якобы «служить пользователю». Но ведь наша цель – проектировать программное обеспечение, которое удовлетворяет потребности *реальных* пользователей. Реальные пользователи – и представляющие их персонажи – не пластилиновые, у них есть конкретные требования, основанные на их целях, способностях и контексте использования продукта.

Даже использование ролей пользователей или названий должностей вместо конкретизированных архетипов способно внести в процесс проектирования неопределенность. К примеру, при проектировании медицинских продуктов может оказаться соблазнительным счесть, что у всех медсестер одинаковые потребности. Как известно любому, кто бывал в больнице, есть медсестры из отделения травматологии, из отделения детской реанимации, из операционной – и у каждой группы собственные взгляды, способности, потребности, мотивы. Нехватка точности в определении пользователя может привести к недостаточно ясному представлению о требуемом поведении продукта.

Проектирование под себя

Проектирование под себя имеет место тогда, когда проектировщики или разработчики отражают в проектных решениях собственные цели, мотивацию, навыки и ментальные модели. В эту категорию попадает большинство «смелых дизайнерских решений»: аудитория продукта мыслится состоящей из людей, подобных самому проектировщику, что приемлемо для узкого спектра продуктов и совершенно недопустимо для большинства других. Программисты, создавая продукты на основе модели реализации, также занимаются проектированием для себя. *Они* прекрасно понимают, как структурированы данные и как устроен программный код, и поэтому находят такие продукты удобными. А вот те, кто не занимается программированием, вряд ли с ними согласятся.

Проектирование в расчете на исключительные ситуации

Еще один синдром, который помогают предотвратить персонажи, – проектирование в расчете на исключительные ситуации. Такие ситуации в принципе могут возникнуть, но целевые персонажи, как правило, с ними не сталкиваются. Разумеется, исключительные ситуации следует учитывать при проектировании и программировании, но нельзя *строить вокруг них весь процесс проектирования*. Персонажи дают возможность проверять решения на соответствие реальности. Мы можем спросить себя: «Насколько часто Джули захочет выполнять это действие? Захочет ли вообще?» Вооруженные этим знанием, мы можем очень четко назначать приоритеты различным функциям.

Персонажи основаны на исследованиях

Как любая модель, персонажи должны основываться на наблюдениях за реальностью. В предыдущей главе мы говорили, что основным источником данных при синтезе персонажей служат интервью, основанные на этнографических методах, контекстное исследование либо другие формы обсуждений с действительными и потенциальными пользователями и наблюдения за ними. Качество данных, собранных в ходе этого процесса (описанного в главе 4), напрямую определяет действенность персонажей как средства, позволяющего упростить и направить процесс проектирования. Прочие данные, способные помочь в процессе создания персонажей и уточнить информацию о них, включают в себя (в примерном порядке убывания эффективности):

- данные интервью с пользователями вне контекста использования;
- информацию о пользователях, предоставленную заинтересованными лицами и экспертами в предметной области (ЭПО);
- данные исследований рынка, таких как фокус-группы и опросы;
- модели сегментации рынка;
- результаты обзора литературы и более ранних исследований.

При этом никакие вспомогательные данные не могут заменить прямое взаимодействие и наблюдение за пользователями в привычной для них среде. Практически каждое слово из описания хорошо проработанного персонажа можно поставить в соответствие некоторой цитате из интервью или наблюдавшемуся поведению.

Персонажи представляются как отдельные личности

Персонажи – это модели пользователей, представленные в виде отдельных, конкретных людей. Они не являются реальными людьми, но синтезируются непосредственно по результатам наблюдений за реальными людьми. Успех персонажей как моделей пользователей во многом определяется тем, что персонажи *олицетворяют личности* (Constantine and Lockwood, 2002). Такое представление является уместным и действенным благодаря уникальным особенностям персонажей как

моделей: они вызывают *эмпатию* (*сопереживание*) проектировщиков и разработчиков по отношению к людям, ради которых выполняется проектирование. Чувство эмпатии абсолютно необходимо проектировщикам, которым предстоит при проектировании принимать концептуальные и детальные решения исходя как из когнитивных, *так и* из эмоциональных свойств персонажей, представленных целями этих персонажей. (О важных связях между пользовательскими целями, поведением пользователей и персонажами мы поговорим далее в этой главе.) Однако возможности эмпатии не следует сбрасывать со счетов и тогда, когда речь идет о других участниках команды. Персонажи не только помогают проектировщикам найти решения, лучше отвечающие действительным потребностям пользователей, но и делают эти решения более обоснованными в глазах заинтересованных в реализации лиц. Если персонажи разработаны тщательно и корректно, заинтересованные лица и инженеры начинают думать о них как о настоящих пользователях и проявляют гораздо больший интерес к созданию продукта, который станет источником положительного опыта для этих персонажей.

Известно, какой властью над читателями и зрителями обладают вымышленные персонажи книг, фильмов и телепрограмм. Джонатан Грудин (Jonathan Grudin) и Джон Прюит (John Pruitt) рассмотрели эту особенность применительно к проектированию взаимодействия (Grudin and Pruitt, 2002). Среди прочего они отметили важность методики **вживания в роль**, применяемой актерами для понимания и воссоздания реалистичных действующих лиц. Процесс создания персонажей на основе наблюдений за пользователями и последующего проигрывания ролевых сценариев в роли этих персонажей во многом аналогичен методике вживания в роль. (Нам доводилось даже слышать, как целеориентированное применение персонажей называют системой Станиславского в проектировании взаимодействия.)

Персонажи являются представителями групп пользователей

Хотя персонажи изображаются как конкретные личности (поскольку выполняют функцию архетипов), каждый из них является *представителем* класса или типа пользователей *конкретного* интерактивного продукта. Персонаж вбирает в себя уникальный набор **шаблонов поведения**, связанных с использованием определенного продукта (или с аналогичной деятельностью в предметной области, если продукт еще не существует), которые выявляются посредством анализа данных интервью и при необходимости подкрепляются дополнительными количественными данными. Наряду с конкретными мотивами и целями эти шаблоны задают персонажей. Иногда персонажей называют также **составными архетипами пользователей**, поскольку они создаются путем группировки родственных шаблонов использования, наблюдавшихся в ходе фазы исследований у пользователей, играющих сходные роли (Mikkelson and Lee, 2000).

Персонажи и повторное использование

Организации, выпускающие несколько продуктов, часто желают повторно использовать тех же персонажей. Но чтобы быть эффективными, персонажи должны быть четко привязаны к контексту – то есть ориентированы на особенности поведения и цели, связанные с конкретной предметной областью определенного продукта. Поскольку персонажи конструируются на основе наблюдений за пользователями, работающими с определенными продуктами в уникальном контексте, их не получится легко перенести на другие продукты, даже если эти продукты относятся к одному узкому семейству (Grudin and Pruitt, 2002).

Чтобы набор персонажей был эффективным средством проектирования для нескольких продуктов, персонажей следует создавать на основе исследований, охватывающих контексты использования всех этих продуктов. Это приведет не только к увеличению масштаба исследований, но и к более сложной проблеме выявления управляемых и связных наборов шаблонов поведения, действительных для всех контекстов. Ошибкой будет считать, что два пользователя, проявляющих сходное поведение в отношении одного продукта, будут сходно вести себя и в отношении другого. Таким образом, чем больше продуктов вы пытаетесь охватить, тем сложнее сохранять четкость и связность набора персонажей, представляющего разнообразие настоящих пользователей. Мы считаем, что в большинстве случаев для различных продуктов следует изучать и создавать самостоятельных персонажей.

Архетипы и стереотипы

Не путайте архетипы, которыми, по сути, являются персонажи, со **стереотипами**. Стереотипы во многих отношениях являются противоположностью хорошо проработанных персонажей, так как представляют предубеждения и предположения проектировщиков и исследователей, а не фактические данные. Персонажи, созданные на основе неадекватных исследований (или составленные без должной степени эмпатии и участия по отношению к участникам интервью), рискуют деградировать до уровня карикатурных стереотипов. Персонажи должны создаваться и использоваться с уважением и почтением к тем людям, которых они представляют. Если проектировщик не уважает своих персонажей, другие люди тоже не будут их уважать.

Кроме того, персонажи выводят на передний план вопросы социальной и политической ответственности (Grudin and Pruitt, 2002). Поскольку персонажи четко задают цели проектирования и выступают в качестве средства коммуникации в команде разработчиков, проектировщик должен особенно осторожно выбирать демографические характеристики. В идеальном случае демография персонажа должна отражать сочетание характеристик, наблюдавшихся в выборке респондентов, уточненное широкоохватными исследованиями рынка. Персонажи должны быть *типичными* и правдоподобными, но не стереотипными. Если

данные недостаточно точны или же характеристика не является важной для проектирования либо его результатов, мы предпочтем ошибиться в пользу полового, этнического, возрастного и географического разнообразия.

Персонажи описывают варианты поведения

Целевой рынок продукта задает демографию, стиль жизни людей, а иногда и их рабочие роли. Однако целевой рынок не определяет весь спектр вариантов поведения его участников, связанный с самим продуктом и контекстом его использования. Варианты поведения – это не описание «*среднего*» поведения: смысл персонажей не в определении среднего пользователя, а в выявлении *образцов* поведения в рамках определенного спектра.

Поскольку любой продукт обязан учитывать *варианты* поведения пользователей, спектр их принципов и склонностей, проектировщики должны для каждого продукта определить **набор персонажей**. Множественные персонажи разбивают спектр вариантов поведения на дискретные кластеры. Каждый персонаж представляет набор взаимосвязанных шаблонов поведения. Эти связи выявляются при анализе результатов исследований. Более подробно процесс кластеризации вариантов поведения будет описан далее в этой главе.

Персонажи должны обладать мотивацией

Поведение каждого человека определяется его мотивами; некоторые очевидны, но многие скрыты. Крайне важно, чтобы персонажи отражали эти мотивы – в виде описания целей. Перечисление целей наших персонажей (их мы более подробно обсудим в оставшейся части главы) – это краткое описание мотивов, не только отсылающее к определенным шаблонам использования, но и объясняющее существование определенных шаблонов поведения. Понимание того, *почему* пользователь выполняет те или иные задачи, дает проектировщикам хорошую возможность усовершенствовать способы решения этих задач или даже вовсе исключить эти задачи на пути достижения все тех же целей.

Персонажи могут представлять не только пользователей

Для проектировщика взаимодействия пользователи и потенциальные пользователи продукта всегда стоят на первом месте, однако иногда бывает полезно иметь представление целей и потребностей людей, которые не пользуются продуктом, но должны быть приняты во внимание в процессе проектирования. Так часто происходит с программным обеспечением для бизнеса (и с детскими игрушками): пользуется продуктом не тот человек, который его приобретает. В таких случаях в дополнение к персонажам пользователей могут оказаться полезными один или несколько **персонажей покупателей**. Разумеется, как

и в случае с пользователями, эти персонажи должны создаваться на основе шаблонов поведения, выявленных в ходе этнографических исследований.

Еще один пример: во многих медицинских продуктах пациенты не взаимодействуют непосредственно с пользовательским интерфейсом, однако обладают собственными мотивами и целями, которые могут значительно отличаться от мотивов и целей использующих эти продукты врачей. В таких случаях бывает полезно создавать **обслуживаемого персонажа**, представляющего потребности пациента. Обслуживаемых персонажей и персонажей покупателей мы более подробно обсудим далее в этой главе.

Персонажи и другие модели пользователей

При проектировании интерактивных продуктов применяется ряд других моделей пользователей – в частности, роли пользователей, профили пользователей и сегменты рынка. Эти модели похожи на персонажей в том смысле, что направлены на описание пользователей и их связей с продуктом. Однако персонажи, а также способы их создания и применения в процессе проектирования значительно отличаются от других моделей рядом важных моментов.

Роли пользователей

Роли пользователей, или ролевые модели, в определении Ларри Константайна, являются *абстракцией* – определенным соотношением между классом пользователей и их задачами, включая потребности, интересы, ожидания и шаблоны поведения (Constantine and Lockwood, 1999). Будучи абстракциями (принимающими обычно форму списка атрибутов), роли не визуализируются в виде людей, а потому обычно неспособны передавать более широкие человеческие мотивы и контексты.

Сходным образом используют роли Хольцблат и Бейер при объединении потоковых, культурных, физических моделей и диаграмм последовательностей – они пытаются абстрагировать различные атрибуты и типы отношений, отделив их от людей, которые обладают этими атрибутами и отношениями (Beyer and Holtzblatt, 1998).

Эти подходы представляются нам не слишком удачными ввиду следующих причин:

- Абстрагирование и работа в отрыве от конкретных людей мешают должным образом передавать другим особенности поведения пользователей и их отношения – человеческая эмпатия плохо работает с абстрактными классами людей.
- Оба подхода практически полностью сосредотачиваются на *задачах* и пренебрегают целями как полезным способом организации процессов мышления и синтеза.

- Объединенные модели Хольцблат и Бейера, несмотря на их полезность и широкий охват, недостаточно хороши как целостный инструмент, который можно применять при выработке, передаче и оценке проектных решений.

Персонажи решают все эти проблемы. Хорошо проработанные персонажи охватывают все те же виды отношений, что и роли, но выражают их в виде целей и дают примеры в повествовательной форме. В итоге как проектировщикам, так и заинтересованным лицам становится проще понимать последствия проектных решений в «человеческой» перспективе. Описание целей персонажей задает контекст и структуру для задач, указывая, как культурная среда и рабочий процесс влияют на поведение.

Кроме того, сосредоточение на ролях пользователей вместо более сложных шаблонов поведения может привести к чрезмерному выхолащиванию отличительных признаков, определяющих похожесть и непохожесть пользователей. Можно создать персонажа, представляющего потребности нескольких ролей (скажем, при проектировании мобильного телефона путешествующий коммивояжер может представлять также потребности занятого руководителя, который постоянно находится в разъездах). Может оказаться также, что одну роль выполняют разные люди, которые по-разному мыслят и действуют (скажем, сотрудник отдела закупок в химической промышленности может думать о своей работе совсем иначе, чем такой же сотрудник в производстве бытовой электроники). В случае потребительских товаров роли практически бесполезны. Если вы проектируете веб-сайт для автомобильной компании, роль «покупатель автомобиля» не имеет смысла как средство проектирования: разные люди очень по-разному подходят к вопросу приобретения автомобиля.

В общем, персонажи дают более целостную модель пользователей и контекста, в котором эти пользователи находятся, тогда как многие другие модели, напротив, стремятся к большей схематичности. Персонажей, конечно же, можно применять в сочетании с другими методиками моделирования, и, как мы увидим в конце главы, некоторые модели становятся крайне полезными дополнениями к персонажам.

Персонажи и профили пользователей

Многие юзабилити-специалисты считают термины **персонаж** и **профиль пользователя** синонимами. Здесь нет проблемы, если профиль действительно создан на основе этнографических данных и содержательно отражает всю глубину описанных выше характеристик. К сожалению, слишком часто нам приходилось видеть профили пользователей, соответствующие определению «профиля» в словаре Вебстера: «краткий биографический очерк». Иначе говоря, профиль пользователя – это зачастую просто его имя (и, возможно, фотография) плюс краткие, как правило, демографические данные, а также небольшой абзац

с *вымышленным* описанием того, на какой машине человек ездит, сколько у него детей, где он живет и чем зарабатывает на жизнь. Такого рода профиль пользователя с большой вероятностью окажется стереотипом и будет бесполезным в качестве средства проектирования. Хотя мы придумываем для своих персонажей имена, а иногда даже автомобили и членов семьи, все это используется осторожно, как средство повествования, позволяющее лучше передавать стоящие за этим реальные данные. Вспомогательные вымышленные детали играют минимальную роль в создании персонажа и служат лишь для того, чтобы оживить персонажа в глазах проектировщиков и разработчиков продукта.

Персонажи и рыночные сегменты

Маркетологам может быть знаком процесс, сходный с созданием персонажа, – процесс определения рынка. Основное различие между сегментами рынка и персонажами для проектирования состоит в том, что первые основаны на демографии, каналах распространения и поведении потребителей, а вторые – на шаблонах использования и мотивах пользователей. Это разные модели, которые служат различным целям. Маркетинговые персонажи проливают свет на процесс продажи, а персонажи проектировщиков – на процесс определения и разработки продукта.

Тем не менее сегменты рынка играют свою роль в создании персонажей: они помогают задавать демографический диапазон, в рамках которого определяется гипотеза о персонажах (см. главу 4). Персонажи сегментируются соответственно вариантам поведения, а не по демографическим данным или потребительскому поведению, поэтому сегменты рынка и персонажи редко совпадают один в один. Скорее, сегменты рынка способны послужить первичным фильтром, ограничивающим масштабы выборки респондентов для интервью рамками целевого рынка (рис. 5.3). Кроме того, мы обычно назначаем персонажам приоритеты, помогающие принимать стратегические решения по определению продукта (о типах персонажей мы поговорим чуть позже). Эти решения должны основываться в том числе и на результатах анализа рынка; понимание связи между персонажами и сегментами рынка может оказаться весьма важным.

Условный персонаж: что делать, если нет точного персонажа

Крайне желательно, чтобы персонажи основывались на подробных качественных данных, однако в некоторых случаях просто не хватает времени, ресурсов или поддержки со стороны руководства, чтобы выполнить всю необходимую работу «в поле». В таких случаях полезными риторическими инструментами, ясно выражающими предположения относительно того, какие пользователи являются важными и каковы их потребности, а также вынуждающими участников процесса

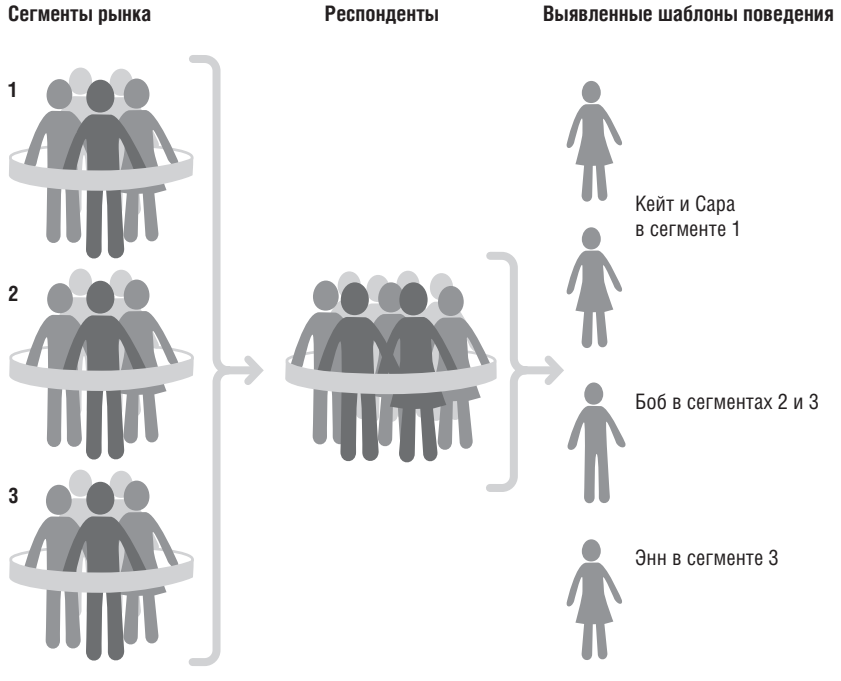


Рис. 5.3. *Персонажи и сегменты рынка. Сегменты рынка можно использовать на стадии исследований для ограничения диапазона персонажей пределами целевых рынков. Однако соответствие один к одному между сегментами рынка и персонажами встречается редко*

четко мыслить о реализации конкретных потребностей пользователей (даже если эти потребности не проверены на соответствие действительности) могут стать *условные* персонажи (или, как их называет Дон Норман, персонажи «на данный случай»).

Условные персонажи структурируются аналогично реальным персонажам, но основаны на доступных данных, а также на догадках проектировщика о поведении, мотивах и целях. Как правило, в основу кладется сочетание экспертных знаний заинтересованных лиц и ЭПО о пользователях (когда такая информация доступна), а также выводы о пользователях, сделанные на основе существующих данных по рынку. По сути дела условные персонажи – это гипотеза о персонажах, на которую нарастили немного «мяса» (см. главу 4).

По нашему опыту, применение условных персонажей, даже невзирая на недостаточно обширные исследования, дает лучшие результаты, чем полное отсутствие моделей пользователей. Как и реальные персонажи, условные персонажи помогают команде разработчиков фокусировать усилия и создают согласованное представление о возможностях и требуемом поведении продукта. Правда, есть и тонкости: условные

персонажи потому и называются условными, что служат всего лишь дублерами персонажей, основанных на полноценных качественных данных. Условные персонажи помогают проектировщикам и разработчикам фокусировать усилия, однако в отсутствии данных, подтверждающих предположения, кроются следующие риски:

- сфокусировать усилия на неверных целях проектирования;
- сфокусировать усилия на верных целях, но упустить из виду ключевые аспекты поведения, которые сделали бы ваш продукт особенным;
- столкнуться со сложностями в получении поддержки лиц и групп, которые не участвовали в создании персонажей;
- дискредитировать ценность персонажей и получить долгосрочную аллергию на них на уровне организации в целом.

При использовании условных персонажей важно:

- недвусмысленно пометить их в качестве таковых;
- визуально представлять их через наброски вместо фотографий, чтобы подчеркнуть условность;
- стараться использовать как можно больше существующих данных (обзоры рынка, исследования предметной области, ЭПО, полевые исследования, персонажи для сходных продуктов);
- отмечать, какие данные были использованы и какие предположения выдвинуты; держаться подальше от стереотипов (это сложнее делать, когда нет данных полевых исследований);
- сосредотачиваться на поведении и мотивах, а не на демографических характеристиках.

Цели

В то время как персонажи задают контекст для шаблонов вариантов поведения, **цели** являются стимулами этого поведения. Персонаж без целей полезен в качестве инструмента коммуникации, но теряет ценность в качестве средства проектирования. Цели пользователя – это призма, через которую проектировщики должны рассматривать функции продукта. Функциональность и поведение продукта должны быть ориентированы на достижение целей пользователей через решение задач – причем посредством минимально возможного числа этих задач. Помните, что задачи – лишь средство движения вперед, а цель – и есть действительный пункт назначения.

Цели мотивируют использовать продукт определенным образом

Именно цели людей или персонажей побуждают их вести себя определенным образом. Таким образом, цели не только отвечают на вопрос,

почему и как персонажи желают пользоваться продуктом, но могут также служить проектировщику компактным внутренним представлением для подчас весьма сложного поведения персонажа, равно как и решаемых персонажем задач.

Источником целей должны быть качественные данные

Обычно нельзя прямо спросить у человека, каковы его цели: он либо не сможет четко выразить их, либо будет не до конца откровенен и точен. Люди попросту не обладают нужной подготовкой для точных ответов на такие вопросы. Следовательно, проектировщики и исследователи должны скрупулезно воссоздавать цели исходя из наблюдаемого поведения, ответов на другие вопросы, невербальных признаков, а также подсказок, предоставляемых средой, таких как названия книг на полке. Одной из важнейших задач в моделировании персонажей является выявление целей и их краткое выражение: каждая цель должна описываться одним простым предложением.

Цели пользователей и когнитивные процессы обработки

В своей книге «Emotional Design» (Norman, 2005) Дон Норман выдвигает идею о том, что проектирование продукта должно затрагивать три различных уровня когнитивной и эмоциональной обработки, которые он называет физиологическим, поведенческим и аналитическим. Идеи Нормана, основанные на годах исследований человеческого сознания, предлагают внятную структуру для процесса моделирования реакции пользователей на продукт и бренд, а также рациональный контекст для интуитивных догадок, давно будораживших умы профессиональных проектировщиков.

Вот три уровня когнитивной обработки по Норману:

- **Физиологический.** Самый непосредственный уровень обработки, на котором человек реагирует на визуальные и другие сенсорные аспекты продукта еще до того, как случится какое-либо значимое взаимодействие. Обработка на физиологическом уровне позволяет нам быстро принимать решения о том, что хорошо или плохо, безопасно или опасно. Это одна из замечательных особенностей человеческого поведения, но ее эффективная поддержка входит в число наиболее сложных задач при создании цифровых продуктов. Малкольм Гладуэлл (Malcolm Gladwell) исследует этот уровень когнитивной обработки в своей книге «Blink»[1]. Еще более подробное изучение принятия решений на интуитивном уровне предпринято в книгах «Sources of Power» Гэри Клейна (Gary Klein) и «Hare Brain, Tortoise Mind» Гая Клэкстона (Guy Claxton).

[1] М. Гладуэлл «Озарение. Сила мгновенных решений». – Пер. с англ. – М.: Вильямс, Альпина Бизнес Букс, 2008.

- **Поведенческий.** Средний уровень обработки, позволяющий нам контролировать простое, повседневное поведение. Этот уровень, по Норману, представляет большую часть человеческих действий. Норман утверждает – и в этом он прав, – что традиционно в практике проектирования и юзабилити рассматривался почти исключительно этот уровень когнитивной обработки. Поведенческая обработка способна *усиливать* или *сглаживать* как более низкоуровневые физиологические реакции, так и более высокоуровневые аналитические. И, наоборот, физиологические и аналитические реакции способны усиливать или сглаживать реакции уровня поведения.
- **Аналитический.** Самый поздний уровень обработки, на котором происходит сознательное изучение и анализ уже приобретенного опыта. Аналитическая обработка способна усиливать или сглаживать поведенческую обработку, но не оказывает прямого влияния на физиологические реакции. Этот уровень когнитивной обработки доступен только через память, но не через прямое взаимодействие или восприятие. С точки зрения проектирования наиболее интересный аспект аналитической обработки состоит в том, что посредством анализа и рефлексии мы способны интегрировать наш опыт обращения со спроектированными артефактами в более широкий жизненный опыт и со временем увязывать смыслы и ценности непосредственно с артефактами.

Проектирование для физиологических реакций

Проектирование для физиологического уровня означает проектирование того, что изначально воспринимается человеком через органы чувств, прежде чем он более близко познакомится с продуктом или артефактом. Для большинства из нас это сводится к проектированию внешнего облика и динамики, хотя некоторая роль может отводиться и звуку – вспомним хотя бы узнаваемый аккорд включения компьютеров Apple. Проектировщики устройств могут проектировать также тактильные ощущения.

При обсуждении физиологического уровня часто всплывает заблуждение о том, что при проектировании на этом уровне речь идет о создании *красивых* вещей. Программы для ведения боевых действий и системы радиотерапии – вот всего два примера ситуации, где проектирование *красоты* может оказаться неподходящим направлением приложения усилий. Действительный смысл проектирования на физиологическом уровне – в том, чтобы произвести впечатление, то есть вызвать уместную в определенном контексте психическую или эмоциональную реакцию, а не только в эстетике. Красота – и связанные с ней ощущения возвышенности и удовольствия – лишь малая часть палитры вызывающего эмоции проектирования. Проигрыватель MP3-музыки и сетевая банковская система требуют очень разных воздействий на пользователя. В этом плане нас очень многому могут научить архитектура, кино и сцена, промышленное проектирование.

Однако в мире потребительских продуктов и услуг привлекательные пользовательские интерфейсы, как правило, *уместны*. Интересный факт: исследователям в области юзабилити удалось показать, что пользователи изначально считают привлекательные интерфейсы более удобными – и эта уверенность зачастую сохраняется еще долго после того, как пользователь приобрел достаточный опыт обращения с интерфейсом и обладает неоспоримыми свидетельствами обратного (Dillon, 2001). Возможно, причина в том, что пользователи, вдохновленные видимой простотой использования, прикладывают больше усилий для изучения сложных интерфейсов, а впоследствии не желают соглашаться с тем, что время было потрачено зря. Что это означает для вдумчивого проектировщика? Если пользовательский интерфейс обещает простоту использования (или что-либо еще) на физиологическом уровне, то это обещание следует выполнить на уровне поведения.

Проектирование для уровня поведения

Проектирование для уровня поведенческих реакций означает создание такого поведения продукта, которое дополнит собственное поведение пользователя, входя в соответствие его неявным предположениям и ментальным моделям. Из трех уровней проектирования, анализируемых Норманом, проектирование поведения, вероятно, лучше всего знакомо проектировщикам взаимодействия и юзабилити-специалистам.

Интересный аспект трехуровневой модели Нормана в приложении к проектированию – его утверждение о том, что из всех трех уровней лишь поведенческий уровень обработки напрямую влияет на два других уровня и, в свою очередь, подвергается непосредственному влиянию обоих. Отсюда можно заключить, что именно поведенческие аспекты ежедневной работы с продуктом должны быть основной заботой при проектировании, а соображения физиологии и анализа играют вторую скрипку. Правильное проектирование поведения – если, разумеется, уделить достаточно внимания другим уровням – дает самую серьезную возможность влиять на опыт взаимодействия пользователей с продуктами.

Отход от такого образа мысли может привести к тому, что первые впечатления пользователя войдут в противоречие с реальностью. Кроме того, трудно представить себе проектирование для аналитических процессов мозга без обращения к явно выраженной цели и набору поведений, отражающих настоящий момент. Следовательно, в идеальном случае опыт взаимодействия пользователя с продуктом или артефактом будет *гармонизацией элементов проектирования физиологического и аналитического с фокусировкой на проектировании поведения*.

Проектирование для аналитического уровня

Уровень аналитических процессов (особенно в его связи с проектированием) – самый сложный из трех уровней обработки, описываемых

Норманом. Понятно, что проектирование для аналитического уровня означает выстраивание отношений пользователя с продуктом. Однако совершенно неясно, каков наилучший способ добиться успеха (и существует ли он вообще) на аналитическом уровне. Случаен ли успех и следует ли просто быть в нужном месте в нужное время – или же продуманное проектирование способно сыграть свою роль в достижении требуемого результата?

Описывая проектирование для аналитического уровня, Норман в качестве примеров приводит концептуальный дизайн некоторых товаров массового потребления – непрактичные чайники и поразительную соковыжималку Phillipe Starck, украшающую обложку его книги. Легко понять, как такие продукты, ценность и смысл которых, по существу, – в визуальной эстетике, могут хорошо отвечать человеческому стремлению к уникальности или культурной утонченности, источником которого, возможно, является имиджевая или артистическая грань представления людей о самих себе.

Труднее понять, как добиться баланса стиля и элегантности с функциональным наполнением в действительно полезных продуктах. Очень близко к достижению такого баланса подошло устройство Apple iPod. Навигация посредством колеса Click Wheel в некоторых отношениях не оптимальна, но физиологическая реакция пользователей на этот продукт невероятна благодаря элегантному внешнему дизайну. Аналитический потенциал iPod также весьма значителен из-за мощных эмоциональных связей, образуемых музыкой. Конкуренты пока что не нашли, что можно противопоставить этой выигрышной комбинации.

Мало какие продукты становятся столь неотъемлемой составляющей жизни людей, как Sony Walkman или iPod. Очевидно, что у многих продуктов, например у Ethernet-маршрутизаторов, попросту нет шансов стать символами независимо от того, насколько привлекательно они выглядят и как хорошо показывают себя в работе. И все же если проектирование продукта или услуги ориентировано на цели и мотивы пользователей (возможно, с выходом за рамки основного назначения, но все же сохраняя связь с ним посредством личных или культурных ассоциаций), возможности для создания аналитических смыслов значительно расширяются.

Три типа пользовательских целей

В книге «Emotional Design» (Norman, 2005) Норман представил свою теорию трехуровневой когнитивной обработки и продемонстрировал важность этой теории для проектирования. При этом автор не предложил метода для систематической интеграции своей модели познания и воздействия в практику проектирования продуктов или исследования пользователей. Наш опыт показывает, что ключ к такой связи – в соответствующем разграничении и моделировании трех конкретных типов целей пользователей в рамках описания каждого персонажа (Goodwin, 2001).

Три типа пользовательских целей соответствуют уровням когнитивной обработки по Норману – физиологическому, поведенческому, аналитическому:

- эмоциональные цели;
- конечные цели;
- жизненные цели.

Рассмотрим подробнее каждую из категорий.

Эмоциональные цели

Эмоциональные цели просты и универсальны и являются сугубо личными. Парадоксально, но именно из-за этих качеств эмоциональных целей большинству людей так трудно их обсуждать, особенно в контексте обезличенных бизнес-процессов. Эмоциональные цели выражают то, как человек хочет себя *чувствовать*, работая с продуктом, или описывают качество его взаимодействия с продуктом. Такие цели придают ясность требуемым визуальным и аудиальным характеристикам продукта, его интерактивной природе (например, анимированным переходам, задержке отклика, скорости щелчка физической кнопки) и физической форме, давая понимание мотивов персонажа, выраженных на интуитивном уровне. Вот примеры эмоциональных целей:

- Чувствовать уверенность в том, что ситуация под контролем.
- Получать удовольствие.
- Ощущать душевный подъем или расслабленность.
- Быть собранным и сосредоточенным.

Когда продукт заставляет пользователей чувствовать себя глупо или неудобно, их самоуважение и производительность труда снижаются независимо от наличия других целей, а недовольство растет. Стоит чуть переборщить с таким отношением к пользователям – и они воспользуются первым же шансом, чтобы избавиться от системы. Любая система, явно идущая вразрез с личными целями, в конечном итоге потерпит неудачу – независимо от того, насколько хорошо она позволяет пользователям достигать всех прочих целей.

Проектировщики взаимодействия, дизайнеры и промдизайнеры должны переводить эмоциональные цели персонажа в элементы формы, поведения, динамики, звукового сопровождения продукта, чтобы передать нужные эмоции и тон, произвести требуемое впечатление и оказать желаемое воздействие. Полезным инструментом для фиксации эстетических ожиданий персонажей являются исследования визуальных предпочтений, а также вдохновляющие плакаты[1], на ко-

[1] Вдохновляющие плакаты представляют собой большие листы ватмана, на которые в стиле аппликации наклеены журнальные вырезки и другие изображения, позволяющие прочувствовать, что любит персонаж. – *Примеч. науч. ред.*

торых отображаются отношения персонажей к окружению и их поведение.

Конечные цели

Конечные цели отражают мотивацию пользователей при выполнении задач, связанных с использованием конкретного продукта. Когда вы берете в руки сотовый телефон или открываете документ в текстовом редакторе, вы представляете себе определенный результат. Продукт или услуга может способствовать достижению таких целей косвенно либо напрямую. Эти цели являются предметом рассмотрения при проектировании взаимодействия, создании информационной архитектуры и проработке функциональных аспектов промдизайна. Поскольку поведенческий уровень обработки влияет как на физиологические, так и на аналитические реакции, конечные цели входят в число самых значимых факторов, определяющих опыт взаимодействия с продуктом в целом. Чтобы пользователи не пожалели о потраченных деньгах и времени, продукт должен работать на их конечные цели.

Вот некоторые примеры конечных целей:

- Узнавать о проблемах до того, как они послужат причиной катастрофы.
- Поддерживать контакт с родными и друзьями.
- Заканчивать запланированные дела к пяти часам вечера ежедневно.
- Найти музыку, которая мне понравится.
- Получить наилучшее предложение.

Для проектировщиков взаимодействия конечные цели должны служить основой при проработке поведения, задач, вида продукта, а также создаваемых им впечатлений. Контекстные сценарии («день из жизни») и критический анализ (cognitive walkthrough) – вот эффективные инструменты для обнаружения целей и исследования ментальных моделей пользователей, которые, в свою очередь, способствуют качественному проектированию поведения.

Жизненные цели

Жизненные цели представляют личные стремления пользователя и обычно выходят за пределы контекста работы с проектируемым продуктом. Эти цели связаны с глубинными стимулами и мотивами, помогающими объяснить, *почему* пользователь пытается достичь конечных целей. Жизненные цели описывают долгосрочные желания и мотивы персонажа, а также атрибуты представления о самом себе, приводящие к знакомству с продуктом. Эти цели служат маяком для проектирования продукта в целом, создания стратегии и брендинга для него. Вот примеры таких целей:

- Прожить хорошую жизнь
- Преуспеть в реализации амбиций

- Стать знатоком в определенной области
- Быть привлекательным, популярным, завоевать уважение коллег

Проектировщики взаимодействия должны переводить жизненные цели в основополагающие возможности системы, общую концепцию интерфейса и в стратегию бренда. Вдохновляющие плакаты и контекстные сценарии способны помочь при исследовании различных аспектов замысла продукта, а обширные этнографические исследования и культурное моделирование оказываются крайне важными для выявления шаблонов поведения пользователей и их глубинных мотивов. Жизненные цели редко реализуются в проектировании конкретных элементов или особенностей поведения интерфейса напрямую, но о них следует постоянно помнить. Продукт, приближающий пользователя к достижению жизненных целей, а не только конечных целей, завоюет этого пользователя более решительно, чем любая маркетинговая кампания. Ориентация на жизненные цели пользователей и составляет разницу (при условии, что достигаются прочие цели) между пользователем удовлетворенным и пользователем, фанатично преданным.

Цели суть мотивы

В целом очень важно запомнить, что понимание персонажей заключается скорее в уяснении мотивов и целей, нежели в представлении о конкретных задачах или демографических характеристиках. Связав цели персонажа с моделью Нормана, мы можем обозначить самые важные побуждающие стимулы пользователей:

- Эмоциональные цели, связанные с физиологическим уровнем обработки: как пользователь хочет себя *чувствовать*.
- Конечные цели, связанные с поведением: что пользователь хочет *делать*.
- Жизненные цели, связанные с анализом и рефлексией: кем пользователь хочет *быть*.

Применение персонажей, целей и сценариев, как мы увидим в последующих главах, дает доступ ко всей мощи физиологического, поведенческого и аналитического проектирования и связывает эти три направления в гармоничное целое. Некоторые из наших лучших проектировщиков понимают эти аспекты проектирования и действуют сообразно им почти интуитивно, однако сознательное проектирование на всех уровнях человеческого познания и эмоций открывает немыслимые возможности для создания более качественного и приятного пользовательского опыта.

Прочие типы целей

Пользовательские цели – не единственная разновидность целей, которую проектировщики должны принимать во внимание. Цели покупателей, бизнес-цели и технические цели – это непользовательские це-

ли. Обычно эти цели следует учитывать и рассматривать, однако они не влияют на основное направление проектирования. На эти цели также нужно ориентироваться, но их нельзя удовлетворять за счет пользователя.

Цели покупателей

Как мы уже говорили, покупатели преследуют иные цели, нежели пользователи. Точная природа этих целей несколько различается для продуктов потребительских и корпоративных. Рядовые покупатели – это часто родители, родственники или друзья, обычно озабоченные личной безопасностью и настроением человека, для которого приобретается продукт. Корпоративные покупатели – это обычно менеджеры по информационным технологиям, чаще всего обеспокоенные информационной безопасностью, простотой обслуживания и настройки. Персонажи покупателей также могут иметь свои жизненные, эмоциональные и в особенности конечные цели, связанные с продуктом, если они имеют какое-либо отношение к непосредственному его использованию. Цели покупателей не должны преобладать над другими целями, однако их следует в целом учитывать при проектировании.

Бизнес-цели и цели организации

Коммерческие и некоммерческие организации предъявляют собственные требования к продуктам, услугам и системам, и эти требования также требуется моделировать и учитывать при выработке проектных решений. Цели бизнеса, в котором трудятся пользователи и клиенты, как правило, отражены персонажами пользователей и клиентов, однако часто бывает полезно определить бизнес-цели организации, создающей и продающей (или иным способом распространяющей) проектируемый продукт. Очевидно, что организация надеется достичь чего-то, создавая продукт, именно поэтому и тратит деньги и ресурсы на проектирование и разработку.

Вот некоторые из целей бизнеса:

- Увеличить прибыль
- Расширить свое присутствие на рынке
- Удержать клиентов
- Обойти конкурентов
- Эффективнее использовать ресурсы
- Расширить спектр предлагаемых продуктов или услуг

Может оказаться, что вы проектируете для организации, не занятой в бизнесе, скажем для музея, некоммерческой организации или школы (хотя такие организации все чаще выступают в роли бизнеса). У такой организации тоже есть свои цели, которые необходимо принимать во внимание:

- Повысить уровень образованности публики
- Получить достаточно денег, чтобы покрыть расходы

Технические цели

Большинство программных продуктов, которыми мы ежедневно пользуемся, создавались с учетом определенных технических целей. Многие из таких целей состоят в том, чтобы облегчить создание программ, то есть являются целями программистов. Именно поэтому они часто удовлетворяются в ущерб целям пользователей. Вот примеры технических целей:

- Поддерживать различные броузеры
- Сохранять целостность данных
- Повысить эффективность работы программы
- Использовать какой-то конкретный язык разработки или библиотеку
- Сохранить межплатформенное единообразие

Технические цели особенно важны для команды разработчиков. Необходимо в самом начале процесса обучения персонала подчеркнуть, что эти цели должны служить целям пользователей и целям бизнеса. Технические цели не особо важны для успеха продукта, если не являются производными от необходимости достижения других, более человекоориентированных целей. Использование новой технологии может быть *задачей* компании, занимающейся программным обеспечением, но это никогда не входит в *цели пользователя*. Как пользователям нам в большинстве случаев безразлично, делается ли наша работа посредством иерархических баз данных, реляционных баз данных, объектно-ориентированных баз данных, структурированных файлов или же черной магии. Что нас заботит – так это быстрое эффективное выполнение работы с толикой простоты и без ущерба для чувства собственного достоинства.

Успешные продукты в первую очередь служат целям пользователей

Понятие «хороший дизайн» имеет смысл только с позиции человека, применяющего продукт с каким-либо намерением. Намерений без людей не бывает, это неразделимые вещи. Именно поэтому персонажи являются ключевым инструментом в процессе проектирования: они представляют конкретных людей, действующих с конкретными намерениями и целями.

При проектировании продукта важнее всего обращать внимание на намерения или цели, присущие людям, которые применяют его на практике, и это не обязательно покупатели. С вашим продуктом взаимодействует реальный человек, а не корпорация и даже не менеджер по ин-

формационным технологиям, и личные цели этого человека вы должны ставить выше целей корпорации, на которую он работает, и выше целей менеджера по информационным технологиям, который обеспечивает поддержку его деятельности. Ваши пользователи будут прилагать все усилия для достижения бизнес-целей своего работодателя, в то же время стараясь достичь и своих личных целей. И самая важная цель любого пользователя – всегда сохранять человеческое достоинство, то есть не чувствовать себя дураком.

Мы можем с уверенностью говорить, что заставляем пользователя чувствовать себя глупо, если позволяем ему делать серьезные ошибки, препятствуем выполнению адекватного объема работы или заставляем его скучать.

Не заставляйте пользователей чувствовать себя дураками.

Это, вероятно, самый важный принцип проектирования взаимодействия. На протяжении этой книги мы рассмотрим различные способы, которыми существующие программы заставляют пользователей чувствовать себя глупыми, а также способы избежать этой ловушки.

Сущность качественного проектирования взаимодействия – создание такого взаимодействия, которое позволяет достичь целей производителя или поставщика услуг и его партнеров, не препятствуя при этом достижению целей пользователей.

Разработка персонажей

Как уже говорилось ранее, персонажи возникают на основе общих тенденций, выявленных в ходе интервью с пользователями и потенциальными пользователями (а иногда и покупателями) продукта, а также в процессе наблюдений за ними. Пробелы в этих данных восполняются путем дополнительных исследований, а также с помощью информации, представленной ЭПО и заинтересованными лицами и почерпнутой из существующей литературы. При разработке набора персонажей мы стремимся представить все разнообразие наблюдавшихся мотивов, поведений, взглядов, склонностей, ментальных моделей, процессов деятельности или работы, окружений, а также причин недовольства существующими продуктами или системами.

Создание правдоподобных и действенных персонажей в равной степени требует детального анализа и творческого синтеза. Оба вида деятельности в значительной степени выигрывают от применения стандартизированного процесса. Описываемый в этом разделе процесс разработан Робертом Рейманом, Ким Гудвин и Лэйн Хэлли (Robert Reimann, Kim Goodwin, Lane Halley) в компании Cooper, является результатом эво-

люции практических наработок в ходе сотен проведенных авторами проектов по проектированию взаимодействия и задокументирован в ряде статей (Goodwin, 2002, 2002a). Существует множество эффективных методов выявления шаблонов поведения в ходе исследований и преобразования таких шаблонов в архетипы пользователей, однако мы находим, что прозрачность и точность описываемого здесь процесса идеальны для того, чтобы обучить проектировщиков, не знакомых с персонажами, правильной разработке персонажей и помочь опытным проектировщикам фокусировать внимание на реально существующих шаблонах поведения – особенно в области потребительских товаров. Вот основные шаги процесса:

1. Выявить поведенческие переменные.
2. Сопоставить респондентов с поведенческими переменными.
3. Выявить значимые шаблоны поведения.
4. Синтезировать характеристики и соответствующие им цели.
5. Проверить полноту и выявить избыточность.
6. Расширить описание атрибутов и поведений.
7. Назначить персонажам типы.

Рассмотрим каждый из этапов более подробно.

Шаг 1: выявить поведенческие переменные

После того как вы завершили исследования и провели предварительную обработку данных, следующий шаг – перечислить самостоятельные аспекты наблюдавшихся вариантов поведения, то есть создать набор **поведенческих переменных**. Может показаться, что демографические переменные, такие как возраст или место жительства, также влияют на поведение, однако старайтесь не сосредотачиваться на демографии, поскольку поведенческие переменные гораздо более полезны при разработке действенных пользовательских архетипов.

В общем случае наиболее важные различия между шаблонами поведения определяются при рассмотрении следующих типов переменных:

- **Деятельность:** чем занят пользователь, частота и объем.
- **Взгляды:** каким образом пользователь думает о предметной области и технологии продукта.
- **Наклонности:** каковы образование и подготовка пользователя, его способность обучаться.
- **Мотивация:** каким образом пользователь вовлечен в предметную область продукта.
- **Навыки:** умения пользователя, связанные с предметной областью продукта и используемой технологией.

Для корпоративных приложений поведенческие переменные зачастую тесно связаны с ролями, и мы предлагаем перечислять перемен-

ные для каждой роли отдельно. Число переменных будет зависеть от конкретного проекта, но обычно таких переменных от пятнадцати до тридцати на роль.

Эти переменные могут быть очень похожи на входящие в состав гипотезы о персонажах. Сравните переменные, выявленные в собранных данных, с предположениями, сделанными при выработке гипотезы о персонажах. Оказались ли выделенные потенциальные роли действительно различными? Корректны ли найденные поведенческие переменные (см. главу 4)? Обнаружились ли дополнительные переменные, о наличии которых вы не подозревали? Возможно, некоторые переменные не нашли подтверждения в данных?

Перечислите весь набор наблюдавшихся поведенческих переменных. Если данные не сходятся с предположениями, следует добавить, удалить или изменить роли и варианты поведения. Если расхождение достаточно значительное, возможно, имеет смысл провести дополнительные интервью, чтобы заполнить пробелы во вновь обнаруженных диапазонах поведений.

Шаг 2: сопоставить респондентов с поведенческими переменными

Убедившись в том, что вы правильно определили набор поведенческих переменных, продемонстрированных вашими респондентами, переходите к следующему шагу – назначению каждому респонденту соответствующего места в диапазоне каждой переменной. Некоторые из переменных будут отражать непрерывный диапазон поведения (к примеру, от новичка до эксперта в компьютерной области), а некоторые – дискретные варианты выбора (скажем, использование цифрового либо пленочного фотоаппарата).

Здесь важна не столько точность значений, сколько взаимное расположение респондентов. Иначе говоря, не так важно, располагается ли респондент на шкале на уровне 45% или 50%, и часто вообще нет способа измерить это точно и приходится полагаться на интуицию исходя из наблюдений за респондентом. Результатом этого шага должна стать группировка всех респондентов по каждой из осей (рис. 5.4).

Шаг 3: выявить значимые шаблоны поведения

Разместив респондентов по осям, найдите группы (кластеры) отдельных респондентов, близких сразу по нескольким диапазонам или переменным. Группа респондентов, кластеризованная сразу по шести-восьми различным переменным, вероятнее всего, представляет значимый **шаблон поведения**, который ляжет в основу персонажа. У некоторых специализированных ролей может быть лишь один значимый шаблон, однако обычно таких шаблонов два или даже три.

Рис. 5.4. *Сопоставление респондентов с поведенческими переменными. Пример взят из области электронной коммерции. Респонденты располагаются на каждой из поведенческих осей. Точность позиционирования отдельных респондентов в абсолютных координатах не так важна, как их взаимное расположение. Тесные группы респондентов по всем осям указывают на наличие важных шаблонов поведения*

Чтобы шаблон считался корректным, должна существовать логическая или причинно-следственная связь между кластеризованными поведениями, а не просто случайная корелляция. К примеру, когда данные показывают, что люди, регулярно заказывающие компакт-диски, также любят загружать файлы MP3, в этом определенно усматривается логическая связь; однако такой связи, вероятно, нет, когда данные показывают, что люди, регулярно заказывающие компакт-диски, являются также вегетарианцами.

Шаг 4: синтезировать характеристики и соответствующие им цели

Для каждого выявленного значимого шаблона поведения необходимо синтезировать детали на основе ваших данных. Опишите среду потенциального использования, типичный рабочий день (или иной подходящий контекст), существующие решения задач и причины недовольства пользователей этими решениями, а также значимые отношения с окружающими.

На этом этапе достаточно простого перечисления различных характеристик поведения, представленного в сжатой форме. Старайтесь держаться как можно ближе к наблюдавшемуся поведению. Описание одной-двух характерных для личности персонажа деталей поможет вдохнуть в него жизнь. В то же время слишком длинная и чересчур необычная вымышленная биография, напротив, сделает персонажа менее правдоподобным. Помните, что вы создаете инструмент проектирования, а не набросок персонажа для романа. Лишь конкретные данные могут способствовать принятию вашей командой решений в проектировании и в бизнесе.

И все же одна вымышленная деталь на этой стадии является существенной – это имена и фамилии персонажей. Имя должно вызывать од-

нозначные ассоциации с персонажем, но при этом нельзя скатываться к карикатуре или стереотипу. Авторы при создании персонажей предпочитают обращаться к книгам с перечнем имен для детей. На этой же стадии вы можете добавить некоторую демографическую информацию – возраст, место жительства, относительный доход (если требуется) и должность. Эта информация нужна главным образом затем, чтобы помочь вам лучше визуализировать персонажа в процессе описания деталей поведения. С этого момента персонажей следует называть по имени.

Синтез целей

Цели – самая важная из деталей, синтезируемых на основе данных интервью и наблюдений за поведением. Лучше всего эта задача решается посредством анализа шаблонов поведения, присущих каждому персонажу. Определив логические связи внутри вариантов поведения каждого из персонажей, мы можем сделать вывод о целях, определяющих наблюдаемое поведение. То же самое можно сделать путем наблюдения за действиями (чего именно и по каким причинам пытаются достичь респонденты, относящиеся к кластеру данного персонажа) и анализа ответов респондентов на целеориентированные вопросы (см. главу 4) в интервью.

Чтобы стать эффективным инструментом проектирования, цели должны всегда иметь некоторое непосредственное отношение к проектируемому продукту. Как правило, большинство полезных целей персонажа оказываются *конечными целями*. Вы можете ожидать, что у каждого персонажа будет от трех до пяти конечных целей. Жизненные цели наиболее полезны в создании персонажей для массовых продуктов, однако могут иметь смысл и для продуктов корпоративного использования при создании персонажей, работающих на временных должностях. У большинства персонажей следует ожидать не более одной жизненной цели. Практически каждому персонажу неявным образом присущи общие эмоциональные цели, вроде «не чувствовать себя глупо» и «не терять время попусту». Конкретная предметная область может потребовать более специфических эмоциональных целей; для большинства персонажей число таких целей – от нуля до двух.

Взаимоотношения персонажей

Для некоторых проектов имеет смысл делать набор персонажей, входящих в состав одной семьи или работающих в одной компании, описывая межличностные или социальные связи между ними. Однако в типичной ситуации персонажи совершенно не связаны друг с другом, часто живут в различных регионах и относятся к разным социальным группам.

Принимая решение о том, задавать ли социальные отношения между персонажами, обдумайте следующие моменты:

1. Наблюдали ли вы какие-либо различия в поведении респондентов в зависимости от размеров компании, отрасли, в которой они работают, или социальной/семейной динамики? (Если ответ положительный, то необходимо убедиться, что набор персонажей должным образом представляет разнообразие ситуаций, попадая, как минимум, в два различных социальных или деловых контекста.)
2. Важно ли проиллюстрировать особенности рабочего процесса или социальных взаимодействий между сотрудниками или членами семьи либо социальной группы?

При создании персонажей, работающих на одну компанию или связанных между собой социальными отношениями, вы можете столкнуться с трудностями, если потребуется выразить значимую цель, не вписывающуюся в картину существующих отношений. Связать набор персонажей единственной социальной связью, безусловно, проще, чем задать несколько различных и неродственных социальных связей между отдельными персонажами и другими, не входящими в набор персонажей лицами, однако гораздо лучше изначально приложить усилия к созданию разнообразных персонажей, чтобы потом не возникало искушения подчинить совершенно непохожие сценарии динамике одной социальной группы.

Шаг 5: проверить полноту и выявить избыточность

На этом этапе ваши персонажи уже должны начать оживать. Проверьте соответствие персонажей и поведенческих переменных, а также характеристики и цели персонажей – нет ли существенных пробелов, которые требуют заполнения? Наличие таких пробелов, опять же, может свидетельствовать о необходимости дополнительных исследований, ориентированных на поиск конкретных вариантов поведения, не представленных на поведенческих осях. Имеет смысл также просмотреть свои записи – возможно, какого-либо персонажа следует включить из политических соображений, чтобы удовлетворить предположения или просьбы заинтересованных лиц

Если обнаружится, что два персонажа различаются лишь демографическими характеристиками, вы можете исключить одного из них или подстроить его характеристики таким образом, чтобы персонажи стали более непохожими. Каждый персонаж должен отличаться от всех прочих по меньшей мере одним значимым вариантом поведения. Если вы хорошо выполнили задачу по соотнесению персонажей с поведенческими переменными, это не должно быть проблемой.

Убедившись в полноте набора персонажей и в том, что все персонажи осмысленно уникальны, вы получите набор персонажей, достаточно хорошо представляющий разнообразие вариантов поведения и потребности реальных людей и при этом максимально компактный, что позволит сократить усилия на этапе проектирования взаимодействия.

Шаг 6: расширить описание атрибутов и поведений

Перечень характеристик и целей, полученный на шаге 4, очерчивает сущность сложного поведения, однако оставляет многие вещи нераскрытыми. Повествование от третьего лица является гораздо более ярким способом представить взгляды, потребности и проблемы персонажа другим участникам процесса разработки. Оно позволяет также углубить связь проектировщиков/авторов со своими персонажами и их мотивацией.

Типичное описание персонажа – это синтез наиболее важных деталей, полученных в ходе исследований и относящихся к этому персонажу. Такое описание становится очень действенным средством коммуникации. В идеальном случае описание персонажа содержит большую часть находок исследования пользовательской аудитории. Тем самым исследования напрямую питают деятельность по проектированию, как мы увидим в последующих главах.

Как правило, рассказ об одном персонаже не должен занимать более одной-двух страниц. Повествование не обязано включать все наблюдавшиеся детали, поскольку в идеальном варианте проектировщики принимали участие в исследовании, а большинству людей за пределами команды проектировщиков более подробное изложение и не требуется.

Повествование по самой своей природе должно содержать некоторые вымышленные ситуации, но, как уже говорилось, это не художественный рассказ. Наилучшее повествование кратко представляет персонажа через его работу или стиль жизни, дает компактный очерк одного дня из его жизни, включая неприятности, заботы и интересы, имеющие непосредственное отношение к продукту. Любая деталь должна быть продолжением вашего перечня характеристик и включать дополнительные данные из наблюдений и интервью. Если совсем кратко, то повествование должно включать в себя описание того, чего персонаж ждет от продукта.

Будьте осторожны с точными деталями в описаниях. Подробность деталей не должна превышать глубины исследований. В научных дисциплинах результат измерения, равный 35,421 метра, означает, что точность ваших измерений – 0,001 метра. Подобно этому детальное описание персонажа предполагает соответствующую глубину наблюдений в проведенных исследованиях.

В начале создания повествовательных описаний выберите фотографии для своих персонажей. Фотографии позволят вам видеть их более реальными в процессе создания рассказа и помогут передать это чувство другим участникам разработки, когда создание повествований будет закончено. Фотографии следует выбирать очень внимательно. Лучшие фотографии соответствуют демографической информации, содержат намеки на окружение персонажа (персонаж медсестры должен быть в больничной форме и находиться в больничном окружении, возможно,

рядом с пациентом), а также передают общее состояние персонажа (клерк, тонущий в бумажной работе, может выглядеть на фотографии обеспокоенным). Мы используем для поиска подходящих иллюстраций ряд фотобанков с возможностями поиска в них.

Нам представляется также полезным создание для каждого персонажа вдохновляющих плакатов – фотоколлажей, передающих дополнительные эмоциональные и эмпирические стимулы, под воздействием которых существует персонаж (рис. 5.5). Такая подборка небольших изображений способна передать идеи, которые сложно сформулировать словами. Бывают также моменты, когда нам кажется полезным моделировать среду существования персонажей (скажем, в виде плана этажа). Это помогает сделать переменные среды более осязаемыми.

При создании подобных вспомогательных средств коммуникации важно помнить, что персонажи – не самоцель, а лишь инструменты для проектирования и принятия решений. В создании целостного образа персонажа заключен особый смысл, но слишком большое количество украшений и оживляющих деталей повышает риск того, что персонажи будут казаться бесполезной тратой времени. А это, в конечном итоге, снизит ценность персонажей как моделей пользователей.

Рис. 5.5. *Такие коллажи в сочетании с проработанными повествованиями – эффективный способ передать эмоциональные и эмпирические аспекты персонажа*

Шаг 7: назначить персонажам типы

К этому моменту ваши персонажи должны восприниматься уже как настоящие люди, которых вы знаете. Последний шаг в конструировании персонажа завершает процесс превращения качественных данных исследования в набор мощных средств проектирования.

У проектирования всегда должна быть цель – та аудитория, на которой сосредоточен процесс проектирования. И, как правило, чем конкретнее обозначена эта цель, тем лучше. Создание интерфейсного решения, отвечающего потребностям хотя бы трех-четырех персонажей одновременно, уже может оказаться весьма сложной задачей.

Посему необходимо назначить персонажам *приоритеты*, определяющие, кто из них станет основной целью проектирования. Задача в том, чтобы выбрать из набора того персонажа, чьи нужды и цели можно полностью удовлетворить посредством одного интерфейса, не забывая при этом об остальных персонажах. Мы решаем эту задачу, назначая **персонажам типы**. Существует шесть типов персонажей, которые назначаются обычно в таком порядке:

- ключевой
- второстепенный
- дополнительный
- покупатель
- обслуживаемый
- отрицательный

В следующих подразделах мы рассмотрим эти типы персонажей и их значимость с позиций проектирования.

Ключевые персонажи

Ключевые персонажи задают основную цель в проектировании интерфейса. На один *интерфейс* в продукте может приходиться только один ключевой персонаж, однако для некоторых продуктов (в особенности для корпоративного пользования) возможно наличие нескольких различных интерфейсов, каждый из которых ориентирован на отдельного ключевого персонажа. Так, медицинская информационная система может иметь отдельные клинический и финансовый интерфейсы, каждый из которых ориентирован на определенного персонажа. Следует заметить, что здесь мы используем термин *интерфейс* в абстрактном смысле. В некоторых случаях два самостоятельных интерфейса реализуются двумя приложениями, работающими с одним набором данных, в других ситуациях два интерфейса – это просто два набора функций, предназначенных для двух различных пользователей в соответствии с их ролями или настройками.

Нужды ключевого персонажа невозможно удовлетворить, если проектирование сосредоточено вокруг любого другого персонажа из набора. Однако когда целью является ключевой персонаж, то, что приносит удовлетворение ему, у всех прочих персонажей по крайней мере не вызывает неудовольствия. (Из дальнейшего будет видно, что следующим шагом мы находим способы удовлетворить остальных персонажей, не доставляя неудобств ключевому.)

При проектировании каждого интерфейса сосредотачивайтесь на единственном ключевом персонаже.

Ключевой персонаж выбирается методом исключения: цели каждого персонажа следует рассмотреть в сравнении с целями остальных. Если не очевидно, какой из персонажей является ключевым, это может означать одно из двух: или продукту требуется несколько интерфейсов, каждый из которых предназначен для своего ключевого персонажа (так часто бывает в корпоративных и технических продуктах), или же продукт «берет на себя» слишком многое. Если у потребительского продукта несколько ключевых персонажей, то, возможно, объем его функциональности слишком широк.

Второстепенные персонажи

Второстепенный персонаж в основном оказывается доволен интерфейсом ключевого персонажа, но имеет дополнительные потребности, которые можно включить в продукт, не нарушая его способности служить ключевому персонажу. Потребность во второстепенных персонажах есть не всегда, а если их больше трех или четырех, возможно, это свидетельствует об излишнем функциональном охвате или слабой нацеленности на определенную аудиторию. При поиске решений следует прежде всего выбирать решения, подходящие для ключевого персонажа, и затем корректировать их с учетом нужд второстепенного персонажа.

Дополнительные персонажи

Пользовательские персонажи, не являющиеся ни ключевыми, ни второстепенными, – это **дополнительные персонажи**. Их нужды полностью представлены сочетанием нужд ключевого и второстепенных персонажей, их полностью удовлетворяет один из ключевых интерфейсов. На каждый интерфейс может приходиться любое количество дополнительных персонажей. Зачастую дополнительными персонажами становятся политические персонажи – те, которых включили в набор, чтобы учесть предположения заинтересованных лиц.

Персонажи покупателей

Персонажи покупателей отражают потребности покупателей, а не конечных пользователей, как уже обсуждалось ранее. Обычно персона-

жи покупателей используются в качестве второстепенных персонажей. Однако в некоторых корпоративных средах кто-то из таких персонажей может оказаться ключевым, если ему предназначается собственный административный интерфейс.

Обслуживаемые персонажи

Обслуживаемые персонажи несколько отличаются от тех типов персонажей, о которых говорилось до этого. Они вовсе не являются пользователями продукта, однако их *непосредственно затрагивает применение продукта*. Пациент, которого лечат прибором для радиотерапии, не является пользователем этого прибора, однако хороший интерфейс этого продукта в существенной степени *обслуживает* его. Обслуживаемые персонажи – это способ отслеживать социальные и физические воздействия второго порядка, оказываемые продуктом. Эти персонажи используются так же, как второстепенные персонажи.

Отвергаемые персонажи

Отвергаемые персонажи используются, чтобы демонстрировать заинтересованным лицам и участникам разработки, что существуют пользователи, для которых продукт *не предназначен*. Подобно обслуживаемым персонажам, отвергаемые персонажи не являются пользователями продукта. Они применяются в чисто риторических целях – чтобы сообщить другим участникам команды, какой персонаж определенно *не является* целью проектирования данного продукта. Как правило, хорошие кандидаты в отвергаемые персонажи – это технически подкованные пользователи («старая гвардия») для потребительских продуктов и специалисты по информационным технологиям для интерфейсов корпоративных продуктов, предназначенных конечным пользователям.

Прочие модели

Персонажи – крайне полезный инструмент, однако они не являются единственным инструментом, помогающим моделировать пользователей и их окружение. В работе «Contextual Design» Хольцблат и Бейера содержится огромный объем информации о моделях, кратко рассмотренных ниже.

Модели бизнес-процессов

Диаграммы бизнес-процессов и **диаграммы последовательностей** полезны для отображения информационных потоков и процессов принятия решений в организациях и, как правило, имеют вид потоковых диаграмм или ориентированных графов, отражающих следующие аспекты:

- цель или желаемый результат процесса;
- частота и важность процесса и каждого действия;
- событие, инициирующее процесс в целом и каждое действие;
- зависимости – что требуется, чтобы выполнить процесс в целом и каждое действие в отдельности, а также какие события и действия зависят от завершения процесса в целом и каждого из действий;
- участники процессов, их роли и ответственности;
- конкретные действия;
- принимаемые решения;
- информация, используемая при принятии решений;
- что может пойти не так – ошибки и исключительные ситуации;
- как исправляются ошибки и обрабатываются исключения.

Хорошо проработанный персонаж должен в достаточной степени отражать личный рабочий процесс, однако на межличностном и организационном уровне модели бизнес-процессов все же необходимы для отражения рабочего взаимодействия. При этом пользовательский интерфейс, опирающийся преимущественно на модели бизнес-процессов, часто терпит неудачу по той же причине, что и программы, спроектированные в модели реализации, взаимодействие с которыми основано на их внутренней технической структуре. Поскольку бизнес-процессы для бизнеса являются по сути тем же, чем структура для программирования, проектирование на основе бизнес-процессов в результате дает некую «бизнес-модель реализации», охватывающую всю функциональность, но не учитывающую человека.

Модели артефактов

Модели артефактов, как следует из названия, представляют различные артефакты, которые пользователь использует для решения своих задач и осуществления рабочих процессов. Часто такими артефактами являются бумажные формы и формы в виде веб-интерфейса. Модели артефактов обычно отражают общие места и значимые различия сходных артефактов, что позволяет в ходе проектирования выделить и применить лучшие из существующих решений. Модели артефактов могут быть полезны на более поздних стадиях процесса проектирования, только следует помнить, что буквальный перенос решений из бумажных систем в цифровые, без подробного анализа целей и учета принципов проектирования (особенно тех, которые описаны во второй части книги), обычно приводит к ошибкам юзабилити.

Физические модели

Физические модели, как и модели артефактов, являются попыткой описать элементы окружения пользователя. Физические модели в первую очередь отражают размещение физических объектов, составляю-

щих рабочее пространство пользователей, и могут служить источником сведений о частоте использования различных предметов, а также о физических ограничителях производительности. Добротные описания персонажей включают в себя фрагменты такой информации, однако когда речь идет о сложной физической среде (такой как больничные этажи и сборочные линии), может быть полезным создание отдельных подробных физических моделей (карт или планов этажей) пользовательской среды.

Персонажи и прочие модели вносят упорядоченность в обширные и запутанные данные о пользователях. Теперь, когда вы вооружены мощными моделями пользователей в качестве инструментов проектирования, в следующей главе мы покажем вам, как применять эти инструменты для преобразования целей и потребностей пользователей в реально работающие решения.

6

Основы проектирования: сценарии и требования

В двух предыдущих главах мы описывали, как фиксировать качественную информацию о пользователях и создавать модели на ее основе. Посредством тщательного анализа этой информации и синтеза персонажей и прочих моделей пользователей мы можем получить четкое представление о пользователях и их целях. Таким образом, мы вплотную подошли к ключевому вопросу метода в целом: как преобразовать наши знания о пользователях в решения, которые будут радовать и вдохновлять пользователей, при этом решая задачи бизнеса и укладываясь в технические ограничения.

В этой главе мы опишем первую часть процесса, ликвидирующего разрыв между исследованиями и проектированием. Персонажи служат здесь главным пунктом в ряде техник, позволяющих быстро получать проектные решения путем итеративной, воспроизводимой и доступной для проверки процедуры. В составе этой процедуры есть четыре основных вида деятельности: создание повествований, или *сценариев*, как средства описания идеального для пользователя взаимодействия, использование этих сценариев для выработки *требований*, определение на основе этих требований *инфраструктуры взаимодействия* для продукта и пошаговое наполнение этой структуры все более детальными решениями. Связующим звеном между процессами является *нарративная техника* – использование персонажей для создания рассказов, задающих направление при выработке проектных решений.

Сценарии: повествование как средство проектирования

Повествование, или рассказывание историй, – один из древнейших видов деятельности человека. О силе повествования как средства *передачи*

идей написано много. Однако повествование – еще и один из самых мощных методов творчества. С детских лет мы используем рассказы для размышлений о возможном, и это невероятно эффективный путь *придумать* новое и более удобное будущее для наших пользователей. Создавая вымышленную историю о том, как человек использует наш продукт, мы получаем от своего творчества гораздо больше выгоды, чем если просто пытаемся выдумать лучший форм-фактор или расположение элементов на экране. Более того, повествование благодаря присущему ему аспекту социальности является очень действенным способом обмена хорошими идеями с участниками команды и заинтересованными лицами. В конечном счете, проектирование опыта, основанное на повествовании, дает более понятный и интересный для пользователей результат, поскольку основой служил рассказ.

Нас окружают свидетельства эффективности повествования как средства проектирования. Знаменитые инженеры-затейники студии Disney[1] не знали бы, что делать, не будь у них современных мифов, которые они используют в качестве основы создаваемого опыта. Об этой идее писали многие: Бренда Лорел (Brenda Laurel) исследовала идею структурирования взаимодействия посредством театральных техник в своей книге «Computers as Theater» (Laurel, 1991), где призывала «...сосредоточиться на проектировании действий. Проектирование объектов, среды и ролей второстепенно в сравнении с этой главной целью». Джон Рейнфранк (John Rheinfrank) и Шелли Ивенсон (Shelley Evenson) также говорили об огромной пользе «историй о будущем» для разработки концептуально сложных интерактивных систем (Rheinfrank and Evenson, 1996), а из-под пера Джона Кэрролла (John Carroll) вышло значительное количество работ, посвященных сценарному проектированию, которое мы обсудим далее в этой главе.

Повествование отлично подходит для визуализации интерактивных продуктов. Поскольку проектирование взаимодействия – это прежде всего проектирование поведения, а поведение характеризуется протяженностью во времени, повествовательная структура в сочетании с простейшими инструментами визуализации, такими как доска с маркерами (whiteboard), идеально подходит для описания, представления и проверки концепций проектирования.

Повествования в проектировании взаимодействия во многом напоминают комикс-раскадровку в киноиндустрии: их общими чертами являются наличие сюжета и краткость. Подобно тому, как раскадровка способна вдохнуть жизнь в сценарий фильма, проектные решения должны создаваться и воплощаться в соответствии с сюжетом, то есть следовать истории. Излишне детальная проработка раскадровок является

[1] Walt Disney Imagineering – отделение студии Disney, создающее диснеевские парки аттракционов, курорты, отели, аквапарки, круизные корабли и т. п. Несмотря на «инженерное» происхождение слова imagineer среди затейников есть и художники, и дизайнеры, и писатели. – *Примеч. перев.*

пустой тратой времени и денег и имеет тенденцию приводить к созданию неоптимальных идей – просто потому, что рисование поглощает значительные ресурсы.

На начальной стадии выработки требований мы можем спокойно сосредоточиться на структурных моментах, что позволит нам свободно исследовать концепции проектирования. Голливуд совершает многомиллионные вложения на основании рисунков и карандашных набросков – этого достаточно, поскольку они способны передать действие будущего фильма и производимое им впечатление. Сосредоточившись исключительно на повествовании, мы можем быстро и гибко находить концептуальные решения, избегая при этом неповоротливости и дороговизны, присущих качественно проработанным результатам работы (хотя такие результаты определенно понадобятся после создания общей инфраструктуры продукта).

Сценарии проектирования

В девяностые годы сообществом HCI (Human-Computer Interaction – взаимодействие человека и компьютера) была проделана значительная работа в области проектирования программ, ориентированных на варианты использования. Здесь находятся истоки понятия **сценария**, которое широко используется как указание на метод *решения задач проектирования через конкретизацию* – использование специально составленного рассказа, чтобы одновременно конструировать и иллюстрировать проектные решения. Джон Кэрролл пишет об этих идеях в своей книге «Making Use» (Carroll, 2000):

> Парадоксально, но сценарии одновременно и конкретны, и приблизительны, и осязаемы, и гибки <...> они неявным образом поощряют все стороны мыслить в стиле «А что, если?..» Они позволяют определять рамки возможностей проектировщиков, не препятствуя инновациям. <...> Сценарии привлекают внимание к тому, как будет использоваться проектируемый продукт. Они могут описывать ситуации с разной степенью детализации, с различными целями, способствуя координации различных аспектов проектирования.

Применение **сценарного подхода к проектированию** по Кэрроллу сосредоточено на описании того, как *пользователи решают задачи*. Такое описание включает характеристику *обстановки* рабочей среды, а также *агентов,* или *действующих лиц*, которые являются абстрактными представителями пользователей. Каждого агента называют исходя из его роли, например Бухгалтер или Программист.

Кэрролл явно понимает возможности сценариев и их важность для процесса проектирования, однако мы усматриваем в его подходе к сценариям две проблемы:

- Действующее лицо как абстрактная, ролеориентированная модель в представлении Кэрролла недостаточно конкретно, чтобы обеспечить понимание пользователей или вызвать эмпатию по отноше-

нию к ним. Невозможно адекватно спроектировать поведение системы, не имея подробного знания о пользователях системы.

- Сценарии по Кэрроллу слишком быстро перескакивают к проработке задач, упуская из поля зрения цели и мотивы пользователей, определяющие необходимость этих задач и направляющие их отбор. Хотя Кэрролл вкратце обсуждает цели, он пишет только о *целях сценария*. Тем самым он попадает в логический порочный круг, определяя цели по результатам конкретных задач. По нашему опыту, чтобы выявить задачи и назначить им приоритеты, необходимо сначала изучить цели пользователя. Не обращаясь к мотивам человеческого поведения, трудно определить высокоуровневые требования к проекту.

Недостающий элемент в сценарном подходе к проектированию по Кэрроллу – это использование персонажей. Персонаж является достаточно рельефным представлением пользователя, чтобы выступать правдоподобным агентом в сценарии. Отражая существующие шаблоны поведения и мотивы, персонажи позволяют исследовать влияние мотивов пользователей на задачи и приоритеты задач в будущем. Поскольку персонажи моделируют *цели*, а не просто задачи, круг вопросов, к которым применимы сценарии, может быть расширен до общих требований к продукту. Персонажи помогают ответить на вопросы «Что должен *делать* этот продукт?» и «Как должен выглядеть и вести себя этот продукт?»

Использование персонажей в сценариях

Сценарии, основанные на персонажах, суть краткие нарративные описания одного или более персонажей, применяющих продукт для достижения конкретных целей. Сценарии позволяют начинать проектирование с рассказа, описывающего идеальный с точки зрения персонажа опыт, фокусируя внимание на людях, их образе мысли и поведении, а не на технологии или бизнес-целях.

Сценарии способны фиксировать ход *невербального диалога* (Buxton, 1990) между пользователем и продуктом, средой или же системой, а также структуру и поведение интерактивных функций. Цели выступают в роли фильтров для задач и обусловливают размещение информационных и управляющих элементов в ходе итеративного процесса проектирования сценариев.

В основе контекста и содержания сценария лежит информация, собранная на этапе исследований и подвергнутая анализу на этапе моделирования. Подобно участвующим в импровизации актерам, проектировщики проигрывают роли персонажей, выступающих героями этих сценариев (Verplank, et al, 1993). Такой процесс приводит к немедленному синтезу структуры и поведения продукта (как правило, на доске), которые позже прорабатываются более детально. Наконец, на протяжении всего процесса проектирования персонажи и сценарии при-

меняются для проверки разумности допущений и пригодности выдвигаемых идей.

Разновидности сценариев

На различных этапах целеориентированного проектирования используются три типа сценариев, основанных на персонажах, причем на каждой последующей стадии особенностям интерфейса уделяется больше внимания, чем на предыдущей. Первый тип сценариев – **контекстные сценарии** – используется для высокоуровневого рассмотрения того, как продукт может наилучшим образом послужить потребностям персонажей. (Раньше мы называли эти сценарии «день из жизни», но в конечном итоге сочли, что этот термин является слишком общим.) Контекстные сценарии создаются до начала проектирования, пишутся с точки зрения персонажа и сосредоточены на человеческих действиях, впечатлениях и желаниях. При разработке именно этого вида сценариев проектировщик располагает наибольшей свободой в представлении идеального опыта пользователя. Более подробно создание сценариев такого типа мы рассмотрим далее в этой главе при обсуждении шага 4 процесса выработки требований.

После того как команда проектировщиков определила функциональные и информационные элементы, а также создала общую инфраструктуру (как это описано в главе 7), необходимо пересмотреть контекстный сценарий. В результате добавления к нему более подробных описаний взаимодействия пользователя с продуктом и применения проектного лексикона он становится **сценарием ключевого пути**. Сценарии ключевого пути фокусируются на наиболее важных моментах взаимодействия, не теряя из виду того, как персонаж пользуется продуктом при достижения своих целей. По мере уточнения образа продукта эти сценарии параллельно с проектированием проходят итерационную доработку.

В ходе всего процесса команда проектировщиков применяет **проверочные сценарии** для тестирования проектных решений в различных ситуациях. Как правило, эти сценарии менее подробны и обычно принимают форму набора вопросов «а что, если?..», касающихся предложенных решений. Более подробно о разработке и применении сценария ключевого пути и проверочных сценариев можно прочитать в главе 7.

Сценарии, основанные на персонажах, и варианты использования

Как сценарии, основанные на персонажах, так и варианты использования (use cases) представляют собой методы описания взаимодействия пользователя с системой. Однако они решают очень разные задачи. Целеориентированные сценарии суть средство для итерационного определения *поведения* продукта с позиции конкретных пользователей (пер-

сонажей). Здесь имеется в виду не только функциональность системы, но также приоритет функций, то, как эти функции выглядят для пользователя и как он взаимодействует посредством них с системой.

Варианты использования, в свою очередь, – это методика, основанная на исчерпывающем описании функциональных требований к системе, часто носящем транзакционный характер и ориентированном на низкоуровневые действия пользователя и соответствующие реакции системы (Wirfs-Brock, 1993). Точное *поведение* системы (*как именно* реагирует система), как правило, не является частью обычного, или *конкретного*, варианта использования; многие предположения относительно формы и поведения проектируемой системы остаются неявными (Constantine and Lockwood, 1999). Варианты использования позволяют провести исчерпывающую каталогизацию пользовательских задач для различных классов пользователей, однако мало или совсем ничего не говорят о том, как эти задачи должны быть представлены пользователю и какие приоритеты они получают в интерфейсе. По нашему опыту, самый серьезный недостаток традиционных вариантов использования как основы для проектирования взаимодействия состоит в том, что все возможные взаимодействия с пользователем считаются одинаково важными и одинаково вероятными. Это и понятно: ведь свое начало варианты использования берут скорее в разработке программного обеспечения, нежели в проектировании взаимодействия. Они могут быть полезны для выявления исключительных ситуаций и для определения степени функциональной завершенности продукта, однако их следует применять лишь на поздних стадиях проверки проектных решений.

Требования: информационное обеспечение проектирования взаимодействия

На стадии выработки требований мы отвечаем на вопросы, начинающиеся со слова *«что»*: что за функции нужны персонажам и что за информация должна быть им доступна, чтобы они могли достичь своих целей. Крайне важно ответить на эти вопросы и добиться консенсуса относительно ответов, прежде чем переходить к тому, *как* продукт выглядит, ведет себя, работает, какое оставляет впечатление. Смешение этих двух вопросов (*что* и *как*) может стать одной из серьезнейших ошибок при проектировании интерактивного продукта. Многие проектировщики испытывают соблазн сразу перейти к активному проектированию и отрисовать возможные решения. Независимо от творческих и профессиональных навыков, которыми вы обладаете, мы настоятельно советуем этого не делать. Вы рискуете попасть в бесконечный цикл итераций. Предлагать решение, не располагая четким согласованным определением проблемы, значит лишить себя способа оценить качество проектных решений. В отсутствие такого метода вам, заинтересованным лицам, а также вашим клиентам придется

прибегать к вкусовщине и интуиции, которые дают печально низкий процент успеха, когда речь идет о таких сложных вещах, как интерактивные продукты.

ПРИНЦИП проектирования

Определите, *что* должен делать продукт, прежде чем проектировать, *каким образом* он это будет делать.

Важно заметить, что наше понятие «требований» отличается от принятого в отрасли искаженного значения данного термина. Во многих организациях, занятых разработкой продуктов, слово «требование» стало синонимом «возможности» или «функции». И хотя между требованиями и функциями существует очевидная связь (которую, как вы увидите в следующей главе, мы существенным образом используем в процессе проектирования), мы предлагаем вам думать о требованиях как о синониме *потребностей*. Иначе говоря, на данном этапе требуется четко определить потребности человека и бизнеса, которые должен удовлетворять продукт.

Другая важная причина, по которой не следует смешивать требования с возможностями, заключается в том, что в распоряжении проектировщика, изобретающего наилучший способ удовлетворить некоторую человеческую потребность, оказывается огромной силы рычаг для создания мощного и привлекательного продукта. Рассмотрим в качестве примера проектирование инструмента для анализа данных, помогающего руководителю лучше понимать состояние своего бизнеса. Если начать сразу с вопроса «*как?*», не разобравшись сначала с вопросом «*что?*», можно предположить, что на выходе этого инструмента должны появляться отчеты. Прийти к такому заключению очень легко: проводя исследование пользователей, вы, вероятно, обратили бы внимание на то, что отчеты являются очень распространенным и популярным решением. Однако обдумав некоторые сценарии и проанализировав действительные требования пользователей, вы, возможно, пришли бы к пониманию того, что руководителю на самом деле нужен способ распознавать исключительные ситуации до того, как возникнут проблемы или будет упущена благоприятная возможность, а также способ уяснять проявляющиеся в анализируемых данных тенденции. Отсюда один шаг до осознания того, что статичные, скучные отчеты – вряд ли лучший вариант удовлетворения таких потребностей. Располагая подобным решением, человек будет вынужден самостоятельно выполнять тяжелую работу по анализу отчетов в поисках значимых данных, указывающих на исключительные ситуации и тенденции. Более качественными решениями были бы такие, которые предоставляют пользователю отчеты, основанные на замеченных исключениях, или же предлагают мониторинг тенденций в реальном времени.

И последняя причина, заставляющая отделять исходную проблему от способа ее решения: такой подход дает максимальную гибкость в усло-

виях постоянно меняющихся технологических возможностей и ограничений. Четко определив, что нужно пользователю, проектировщики могут приступить к поиску лучших решений в сотрудничестве с технологами, не ставя под удар способность продукта помогать людям в достижении их целей. Если вы работаете в подобном стиле, возникновение проблем при реализации не повлияет на определение продукта. Кроме того, становится возможным долгосрочное планирование технологического развития, необходимое для реализации более тонких механизмов удовлетворения потребностей пользователей.

Как уже вкратце говорилось, у требований может быть несколько источников. Имеющийся опыт персонажей и ментальные модели часто формируют некоторые базовые ожидания в отношении продукта. Большую часть требований пользователей мы получаем, анализируя сценарии идеального использования, а требования бизнеса и технические требования извлекаем из интервью с заинтересованными лицами. Наш целеориентированный процесс определения требований к продукту описан далее.

Выработка требований с использованием персонажей и сценариев

Как мы вкратце уже говорили в главе 1, процесс перехода от достоверных моделей к интерфейсным решениям в действительности состоит из двух основных этапов. Этап **выработки требований** дает ответы на общие вопросы о сущности и задачах продукта, а этап **формирования инфраструктуры** отвечает на вопрос о том, как ведет себя продукт и каким образом его структура соответствует целям пользователя. В этом разделе мы подробно обсудим этап выработки требований, а в главе 7 – формирование инфраструктуры. Описываемые методы опираются на методологию сценариев, основанных на персонажах, созданную в компании Cooper Робертом Рейманом, Ким Гудвин, Дейвом Кронином (Dave Cronin), Уэйном Гринвудом (Wayne Greenwood) и Лэйн Хэлли (Lane Halley).

Процесс выработки требований состоит из следующих пяти шагов (подробно описанных в оставшейся части главы):

1. Постановка задач и определение образа продукта.
2. Мозговой штурм.
3. Выявление ожиданий персонажей.
4. Разработка контекстных сценариев.
5. Выявление требований.

Хотя хронологически эти шаги выполняются примерно в таком порядке, они образуют итерационный процесс. Проектировщику следует ожидать, что шаги с третьего по пятый придется выполнить несколько раз, прежде чем требования станут устойчивыми. Это необходимая

часть всего процесса, и здесь не стоит срезать углы. Далее представлены подробные описания каждого из пяти шагов.

Шаг 1: постановка задачи и определение образа продукта

Прежде чем приступить к генерации идей, проектировщику важно получить надежное основание для процесса проектирования. Целеориентированный подход направлен на создание исчерпывающего определения продукта посредством персонажей, сценариев и требований, однако на данном этапе необходимо определить направление, в котором следует двигаться при разработке сценариев и требований. Сейчас у нас уже есть представление о пользователях, на которых ориентирован продукт, однако еще нет достаточного основания для создания продукта, так что остается место для замешательства. Таким основанием должна стать постановка задачи, а также образ будущего продукта – они чрезвычайно полезны, так как помогают достичь согласия между заинтересованными лицами, прежде чем процесс проектирования двинется дальше.

Говоря в общем, **постановка задачи** определяет цель самого проектирования (Newman and Lamming, 1995). Постановка задачи проектирования кратко отражает ситуацию, требующую изменения, как с точки зрения персонажей, так и с точки зрения бизнеса, который создает для этих персонажей продукт. Часто между интересами бизнеса и персонажа существует причинно-следственная связь. К примеру:

> Рейтинг удовлетворенности клиентов компании X низок, а доля на рынке уменьшилась на 10% за последний год, потому что у пользователей нет адекватных инструментов, позволяющих посредством решения задач X, Y и Z достичь цели G.

Увязывание бизнес-вопросов с вопросами юзабилити крайне полезно, когда необходимо убедить заинтересованных лиц в необходимости затрат на проектирование, и позволяет рассматривать этот процесс как с точки зрения пользовательских целей, так и с точки зрения бизнес-целей.

Определение образа продукта – действие, обратное постановке задачи: постановка задачи служит обоснованием и высокоуровневой целью проектирования, а определение образа продукта ставит на первое место потребности пользователей, уже от них переводя вас к тому, как благодаря удовлетворению этих потребностей достигаются бизнес-цели.

> В новой версии продукт X поможет пользователям достичь G, поскольку даст им возможность выполнять X, Y, и Z с большей [точностью, эффективностью и т. п.], при этом избавляя от существующих сейчас проблем A, B и C. Это резко повысит удовлетворенность клиентов компании X и приведет к увеличению присутствия на рынке.

Как постановка задачи, так и определение образа продукта опираются на данные исследований и модели пользователей. Цели и потребности

пользователей должны вытекать из описаний ключевых и второстепенных персонажей, а цели бизнеса нужно извлечь из интервью с заинтересованными лицами.

Постановка задачи и определение образа продукта полезны как при перепроектировании существующего продукта, так и при создании продукта, основанного на новой технологии или нацеленного на неизученную рыночную нишу. Сводка целей пользователей и испытываемых ими сложностей в виде постановки задачи и определения образа продукта помогает достичь взаимопонимания внутри команды и привлечь ее внимание к приоритетам предстоящего проектирования.

Шаг 2: мозговой штурм

Цель мозгового штурма на ранних стадиях процесса выработки требований в некоторой степени карикатурна. К этому моменту команда проектировщиков, вероятно, провела дни или даже недели, исследуя и моделируя пользователей и предметную область. В такой ситуации трудно избежать возникновения определенных предубеждений относительно того, как должно выглядеть решение. Однако в идеальном случае проектировщик создает контекстные сценарии, не имея заранее составленного мнения и полностью сосредотачиваясь на персонажах и их желаниях в отношении взаимодействия с продуктом. Смысл мозгового штурма в этот момент состоит в том, чтобы извлечь такие идеи из головы и, записав, тем самым «отпустить» их, по крайней мере на какое-то время.

Тем самым основная цель штурма – по возможности избавить проектировщиков от предубеждений и дать им возможность открыто и гибко работать над созданием сценариев, задействуя свое воображение и применяя свои аналитические способности для извлечения требований из этих сценариев. Побочная выгода проведения мозгового штурма на этом этапе состоит в том, что мозг проектировщика переключается в «режим поиска решений». На стадиях исследования и моделирования приходится выполнять в основном аналитическую работу, а для поиска неожиданных решений проектирования требуется совсем другой настрой.

Мозговой штурм должен происходить без ограничений, без критики – выкладывайте все те сумасшедшие идеи, которые вы уже обдумывали ранее, а также те, которые не обдумывали, и будьте готовы записать их и убрать на хранение до гораздо более поздней стадии процесса. Далеко не все их них могут оказаться в конечном итоге полезными, однако в них может быть зерно чего-то прекрасного, что отлично впишется в общую структуру продукта, которую вы построите позднее. Карен Хольцблат и Хью Бейер описывают удобную для мозгового штурма методику, которая может быть полезна для «расшевеливания» мозгового штурма, особенно если в команду входят не только проектировщики (Beyer and Holtzblatt, 1998).

Не тратьте слишком много времени на этот шаг процесса выработки требований – нескольких часов должно быть более чем достаточно для того, чтобы вы и другие участники команды излили все свои безумные идеи. Если вы заметили, что идеи начали повторяться или иссякли, значит, пора прервать сеанс.

Шаг 3: выявление ожиданий персонажей

Как уже обсуждалось в главе 2, **ментальная модель** – это внутреннее представление человека о реальности, то, как он думает о чем-либо или как объясняет себе это. Ментальные модели глубоко встроены в сознание и часто являются итогом продолжительного жизненного опыта. Ожидания людей относительно продукта и того, как он работает, в большой степени зависят от ментальных моделей.

Возвращаясь к нашему разговору в главе 2, еще раз подчеркнем важность того, чтобы **модель представления** интерфейса – то, как ведет себя и как выглядит продукт, – максимально точно соответствовала ментальным моделям пользователей, а не отражала модель реализации – то есть внутреннее устройство продукта.

Чтобы этого добиться, необходимо записать ожидания пользователей в формальном виде. Это важный источник требований. Для каждого ключевого или второстепенного персонажа необходимо выявить:

- Взгляды, опыт, устремления, а равно и другие социальные, культурные, физические и когнитивные факторы, влияющие на ожидания персонажа.
- Общие ожидания и желания, которые может иметь персонаж в связи с использованием продукта.
- Ожидаемое или желаемое персонажем поведение продукта.
- Что персонаж думает о базовых единицах информации (скажем, в приложении для электронной почты базовой единицей информации будет сообщение или корреспондент).

Может оказаться, что описания персонажа уже содержат достаточно информации для непосредственного ответа на эти вопросы; однако данные исследований остаются при этом ценным источником – используйте их, чтобы проанализировать, как пользователи определяют и описывают объекты и действия, входящие в состав шаблонов использования, какой язык и грамматику они при этом используют. Вот некоторые вещи, на которые следует обращать внимание:

- Что респонденты упоминают в первую очередь?
- Какие глаголы – слова, обозначающие действия, – они используют?
- Какие промежуточные шаги, задачи или объекты, относящиеся к процессу, они *не* упоминают? (Намек: такие шаги, задачи, объекты могут быть не особенно важны для их ментальных моделей.)

Шаг 4: разработка контекстных сценариев

Любой сценарий – это рассказ о людях и их деятельности, однако из трех типов используемых нами сценариев именно контекстные сценарии более всего похожи на рассказы. Они сконцентрированы вокруг деятельности персонажа, его ментальных моделей и мотивов. **Контекстные сценарии** описывают широкий контекст, в котором проявляются шаблоны использования, и включают информацию о среде использования и об организационных вопросах (в случае с производственными системами (Kuutti, 1995).

Как мы уже говорили, *именно здесь начинается проектирование*. Создавая контекстные сценарии, концентрируйтесь на том, как проектируемый продукт может наилучшим образом помогать персонажам в достижении их целей. Контекстные сценарии устанавливают основные точки соприкосновения каждого ключевого и второстепенного персонажа с проектируемой системой (возможно, и с другими персонажами посредством системы) в течение дня или иного осмысленного промежутка времени.

Контекстные сценарии должны быть достаточно общими и не слишком детализированными. Не следует описывать в контекстных сценариях подробности взаимодействия или особенности продукта – сконцентрируйтесь на высокоуровневых действиях с позиции пользователя. Важно сначала нарисовать общую картину, позволяющую систематически подойти к выявлению требований пользователей. Лишь тогда мы сможем проектировать качественное взаимодействие и удобные интерфейсы.

Контекстные сценарии отвечают на вопросы, подобные этим:

- В какой обстановке будет использоваться продукт?
- Будет ли он использоваться в течение долгого времени?
- Часты ли прерывания в работе персонажа?
- Работает ли с компьютером/устройством более чем один пользователь?
- Какие еще продукты используются вместе с проектируемым?
- Какие основные действия должен выполнить персонаж, чтобы достичь своих целей?
- Каков ожидаемый конечный результат применения продукта?
- Какова допустимая сложность продукта исходя из частоты его использования и навыков персонажа?

Контекстные сценарии *не должны* представлять поведение системы в его текущем виде. Они представляют дивный новый мир целеориентированных продуктов, поэтому, особенно на первых стадиях, сосредотачивайтесь на целях. Не стоит пока беспокоиться о том, *как* в точности будут решаться задачи, – изначально необходимо смотреть на продукт отчасти как на некий волшебный черный ящик.

В большинстве случаев требуется написать более одного контекстного сценария. Это крайне необходимо, когда ключевых персонажей несколько, однако и у одного ключевого персонажа могут оказаться два или более различных контекстов использования продукта.

Необходимо также сказать, что контекстные сценарии являются полностью *текстовыми*. Мы пока говорим не о будущей форме системы, а только о поведении пользователя и системы. Для такого обсуждения наилучшим образом подходит текстовое повествование.

Пример контекстного сценария

Ниже приводится пример первоначального варианта контекстного сценария для ключевого персонажа. Продукт объединяет в себе смартфон и сопутствующую услугу оператора. Персонажа зовут Вивьен Стронг, она – агент по продаже недвижимости из Индианаполиса. Цели Вивьен – достичь равновесия между рабочей и семейной жизнью, успешно завершать сделки, добиться того, чтобы каждый ее клиент чувствовал себя *единственным*.

Контекстный сценарий для Вивьен:

1. Готовясь к выходу с утра, Вивьен при помощи смартфона проверяет электронную почту. Смартфон быстро подключается и обладает достаточно большим экраном, так что это удобнее, чем загружать компьютер. Ведь Вивьен еще надо быстренько сделать дочери Алисе бутерброд в школу.
2. Вивьен видит письмо от последнего клиента, Фрэнка, который хотел бы днем посмотреть дом. Контакт Фрэнка уже есть внутри устройства, поэтому Вивьен может позвонить Фрэнку при помощи единственного действия непосредственно с экрана электронного письма.
3. Разговаривая с Фрэнком, Вивьен включает громкую связь, чтобы иметь возможность в ходе разговора смотреть на экран. Она изучает назначенные встречи, чтобы понять, в какое время она свободна. Когда она создает новую запись о встрече, смартфон автоматически отмечает ее как встречу с Фрэнком, потому что знает, с кем она сейчас разговаривает. Заканчивая беседу, она быстро вносит адрес дома в запись о встрече.
4. Отправив Алису в школу, Вивьен направляется в агентство недвижимости, чтобы собрать документы, которые требуются для другой встречи. Ее смартфон уже синхронизировал новые встречи с Outlook, так что остальные сотрудники офиса знают, где она будет днем.
5. День летит быстро, и Вивьен несколько опаздывает на встречу. Направляясь к дому, который хочет смотреть Фрэнк, она получает уведомление от смартфона, что встреча состоится через пятнадцать минут. Открыв смартфон, она видит не только запись о встрече, но и список всех документов, относящихся к Фрэнку, включая электронные письма, заметки, голосовые сообщения и информацию о звонках на номер Фрэнка. Вивьен нажимает кнопку вызова,

и смартфон автоматически связывает ее с Фрэнком, поскольку знает о скорой встрече с ним. Вивьен сообщает Фрэнку, что будет на месте через двадцать минут.

6. Вивьен знает адрес дома, но она не до конца представляет себе, где именно он находится. Она останавливается у тротуара и нажимает на адрес, который ввела в запись о встрече. Смартфон автоматически загружает указания о маршруте до дома, а также миниатюрную карту, на которой показано текущее положение Вивьен относительно пункта назначения.
7. Вивьен вовремя приезжает к дому и начинает показывать его Фрэнку. Она слышит, как в сумочке звонит смартфон. Обычно во время встречи смартфон автоматически перенаправляет звонки на номер голосовой почты, но Алиса знает код, позволяющий обойти это ограничение. Смартфон знает, что звонит Алиса, и поэтому включает особую мелодию.
8. Вивьен принимает звонок и выясняет, что Алиса опоздала на автобус и ее нужно забрать из школы. Вивьен звонит мужу, чтобы выяснить, сможет ли он это сделать, однако попадает в голосовую почту – вероятно, муж находится за пределами действия сети. Она сообщает мужу, что она на встрече с клиентом, и спрашивает, сможет ли он забрать Алису. Через пять минут смартфон издает короткий звук, по которому Вивьен узнает, что это муж; она видит, что он прислал ей короткое сообщение: «Алису заберу, удачи со сделкой!»

Обратите внимание, что описание в сценарии ведется на достаточно высоком уровне, не затрагивая подробности интерфейсов и технологий. Безусловно, созданные сценарии не должны выходить за пределы технологических возможностей, однако на данном этапе детали реального продукта еще не важны. Мы стремимся оставить пространство для новаторских решений – отступить назад никогда не поздно. В конечном итоге мы хотим получить *оптимальный*, но по-прежнему реалистичный сценарий взаимодействия. Обратите внимание, что сценарий тесно увязывает деятельность с целями Вивьен и стремится исключить как можно большее число задач.

Игра в волшебство

Полезный на ранних стадиях разработки сценариев прием – *представить, что интерфейс волшебный*. Если у вашего персонажа есть цели, а продукт волшебным образом этим целям соответствует, насколько простым может быть взаимодействие? Такие размышления помогают проектировщикам выйти за рамки привычного. Волшебные решения, конечно, невозможны, однако поиск творческих путей такой технической реализации взаимодействия, которое было бы максимально близко к волшебному (с точки зрения персонажей), составляет сущность высококлассного проектирования взаимодействия. Продукты, которые позволяют достигать целей с минимальным количеством преград

и навязчивых излишних действий, представляются пользователям почти чудом. Некоторые варианты взаимодействия в приведенном выше сценарии могут показаться несколько волшебными, однако все они находятся в сфере досягаемости для современных технологий. Волшебство состоит не столько в технологии, сколько в ориентированном на цели поведении.

На ранних стадиях проектирования считайте интерфейс волшебным.

Шаг 5: выявление требований

Получив первый удовлетворяющий вас набросок контекстного сценария, вы можете приступить к его анализу с целью извлечения потребностей персонажей – требований. Эти **требования** могут включать в себя *объекты*, *действия* и *контексты* (Shneiderman, 1998). Помните: мы предпочитаем не отождествлять требования с возможностями или задачами. Таким образом, из приведенного выше сценария можно определить, например, такую потребность:

> Звонок (действие) человеку (объект) непосредственно из записи о встрече (контекст).

Этот формат достаточно хорошо работает; если извлекать потребности в таком формате вам покажется неудобным, попробуйте разделить их на группы информационных, функциональных и контекстных требований, описанные в следующих подразделах.

Информационные требования

Потребности персонажа в данных – это объекты и информация, которые должна представлять система. Если говорить в терминах приведенной выше семантики, бывает полезно считать информационные требования объектами и прилагательными, связанными с этими объектами. Например, учетные записи, люди, документы, сообщения, песни, изображения, а также их свойства, такие как состояние, дата, размер, автор, тема.

Функциональные требования

Функциональные требования – это операции или действия, которые должны выполняться с объектами системы и которые, как правило, реализуются в виде интерфейсных элементов управления. Функциональные требования можно считать *действиями* продукта. Кроме того, функциональные требования определяют места или контейнеры, с помощью которых объекты или данные отображаются пользователю. (Очевидно, что это не самостоятельные действия, да и не действия вовсе, однако присутствие контейнеров обычно подразумевается действиями.)

Прочие требования

Поиграв с волшебством, важно составить четкое представление о реалистичных требованиях бизнеса и технологий, для которых выполняется проектирование (хотя мы надеемся, что у проектировщиков есть определенное влияние на выбор технологии в том случае, если этот выбор непосредственно связан с целями пользователей).

- **Требования бизнеса** могут включать в себя сроки разработки, стандарты, структуры ценообразования и бизнес-модели.
- **Требования бренда и опыта пользователей** отражают характеристики опыта, который в идеальном случае пользователи и клиенты связывали бы с вашим продуктом, компанией или организацией.
- **Технические требования** могут включать в себя ограничения по весу, размеру, форм-фактору, свойствам дисплея, энергопотреблению, а также по выбору программной платформы.
- **Требования покупателей и партнеров** могут включать в себя простоту установки, обслуживания, настройки, стоимость поддержки и лицензионные соглашения.

Теперь у вас есть приблизительное и творческое описание того, как продукт поможет пользователям достигать своих целей. Это описание включает в себя контекстные сценарии, а также сокращенный перечень потребностей и требований, извлеченных из данных исследований, пользовательских моделей и сценариев. Теперь вы готовы глубже погрузиться в детали поведения продукта и начать думать о том, как будет представлен продукт и его функции. Вы готовы к формированию инфраструктуры взаимодействия.

7

От требований к пользовательскому интерфейсу: общая инфраструктура и детализация

В прошлой главе мы рассказали о первой части процесса проектирования – о разработке **сценариев**, помогающих придумывать идеальное для пользователей взаимодействие, а также о том, как формулировать **требования** на основе этих сценариев и других источников. Теперь мы готовы приступить к **проектированию**.

Общая инфраструктура пользовательского интерфейса

Однако заниматься деталями пока еще рано – мы остановимся на более высоком уровне и озаботимся общей структурой пользовательского интерфейса и соответствующего поведения продукта. Мы называем эту фазу целеориентированного процесса построением общей инфраструктуры пользовательского интерфейса. Если бы мы проектировали дом, в этот момент нас интересовало бы, какие комнаты нужны в доме, как их следует расположить друг относительно друга и каковы примерно должны быть площади этих комнат. Нас не занимали бы точные размеры комнат, а также детали, вроде дверных ручек, водопроводных кранов и кухонных столешниц.

Общая инфраструктура пользовательского интерфейса определяет структуру опыта пользователя в целом – от расположения функциональных элементов на экране до интерактивного взаимодействия и его организующих принципов, а также визуальный язык, используемый для представления данных, концепций, функциональности и отличительных признаков бренда. Наш опыт показывает, что форму и пове-

дение нужно проектировать как тандем: инфраструктура интерфейса включает в себя инфраструктуру взаимодействия, визуальную инфраструктуру, а иногда и физическую инфраструктуру (инфраструктуру взаимодействия с физическим устройством). На этой стадии проекта сценарии и требования служат проектировщикам сырьем для создания грубых набросков экранов и вариантов поведения, составляющих **инфраструктуру взаимодействия**. Одновременно с этим графические дизайнеры занимаются исследованием визуального языка с целью создания **визуальной инфраструктуры,** которая, как правило, принимает форму детального макета типовых экранов, а промышленные дизайнеры анализируют язык формы, чтобы выработать грубую физическую модель и **физическую инфраструктуру**. О каждом из трех процессов мы поговорим в этой главе.

Мы обнаружили, что если при проектировании сложного поведения и взаимодействия сразу сфокусироваться на пиксельной прорисовке, дизайне элементов интерфейса и конкретных аспектах взаимодействия, это может помешать эффективному проектированию исчерпывающей инфраструктуры, охватывающей все варианты поведения продукта. Используя подход «сверху вниз», мы можем начать с общей картины и создавать решения, не вдаваясь в детали, чтобы сохранить фокус разработки на самом главном – обеспечении достижения целей и удовлетворении требований персонажей.

Пересмотр сделанного – нормальное явление в мире проектирования. Как правило, процесс материализации и презентации решений проектирования помогает проектировщикам и заинтересованным лицам совершенствовать видение продукта и достигать более глубокого понимания того, как продукт мог бы наилучшим образом служить потребностям человека. Трюк, стало быть, в том, чтобы материализованное решение содержало тот минимум деталей, который позволит спровоцировать живой мыслительный процесс. Нет смысла тратить слишком много времени или сил на создание макетов, которые все равно будут отброшены или модифицированы. Мы обнаружили, что раскадровки с эскизами в сочетании со сценариями-рассказами – весьма эффективный способ исследовать и обсуждать решения проектирования, не неся излишних расходов и не порождая ненужной инерции.

Исследования в области восприятия архитектурных чертежей и визуализаций говорят в пользу такого подхода. Результаты изучения реакции людей на различные изображения, созданные в CAD-системах, приводят к заключению, что карандашные наброски стимулируют обсуждение предложенного дизайн-проекта, а также дают ясное понимание того, что представленная работа еще не закончена (Schumann et al., 1996). Кэролин Снайдер (Carolyn Snyder) подробно описывает данный эффект в своей работе «Paper Prototyping» (Snyder, 2003), где речь идет о ценности слабо детализированных представлений с точки зрения получения реакции от пользователей. Мы считаем, что юзабилити-тести-

рование и отзывы пользователей наиболее полезны на стадии детального проектирования, но определенно существуют случаи, когда эта деятельность приносит пользу уже на стадии создания инфраструктуры. (Более подробно юзабилити-тестирование обсуждается в конце главы.)

Создание инфраструктуры взаимодействия

Общая инфраструктура взаимодействия задает не только высокоуровневую компоновку экранов, но также рабочий процесс, поведение и организацию продукта. Следующие шесть шагов составляют процесс создания общей инфраструктуры взаимодействия:

1. Определение форм-фактора, типа приложения и способов управления.
2. Определение функциональных и информационных элементов.
3. Определение функциональных групп и иерархических связей между ними.
4. Макетирование общей инфраструктуры взаимодействия.
5. Создание ключевых сценариев.
6. Выполнение проверочных сценариев для верификации решений.

Мы разбили процесс на последовательные шаги, однако работа здесь выполняется не линейно, а требует последовательных итераций. В частности, порядок шагов с 3 по 5-й зависит от стиля мышления проектировщика (более подробно это обсуждается позже). Итак, вот описание перечисленных шагов.

Шаг 1: определение форм-фактора, типа приложения и способов управления

Первый шаг в создании общей инфраструктуры – определение **форм-фактора** проектируемого продукта. Будет ли это веб-приложение, используемое на компьютерах с высоким разрешением дисплея? Или это телефон, который должен быть маленьким, легким, иметь низкое разрешение экрана, изображение на котором должно быть одинаково четким как в темноте, так и в солнечный день? Или это киоск, который должен быть достаточно прочным, чтобы выжить в общественном месте, обслужив тысячи невнимательных неподготовленных пользователей? Каковы ограничения, накладываемые каждой из этих платформ на результат проектирования? Любой из этих факторов очевидным образом влияет на пользовательский интерфейс продукта, и ответы на заданные вопросы становятся фундаментом для всех последующих усилий по проектированию. Если ответ неочевиден, обратитесь к своим персонажам и сценариям, чтобы лучше понять идеальный контекст и среду использования продукта. В том случае, когда продукт требует проектирования еще и аппаратной части, необходимо заняться промышленным дизайном. Позже в этой главе мы обсудим, как координировать проектирование взаимодействия с промышленным дизайном.

Определяя форму продукта, следует также выявить базовый **тип интерфейса** продукта и **способы управления** системой. Тип интерфейса продукта определяется тем, как много внимания пользователь будет уделять взаимодействию с продуктом и как продукт своим поведением будет отвечать вниманию со стороны пользователя. Решение по этому вопросу следует основывать на контекстах и окружении использования, упомянутых в контекстных сценариях (см. главу 6). Понятие типа интерфейса мы более подробно рассмотрим в главе 9.

Способ управления определяет, каким образом пользователь взаимодействует с продуктом. Этот способ в некоторой степени диктуется форм-фактором и типом приложения, а также предпочтениями, взглядами и навыками персонажа. Вариантов здесь множество: клавиатура, мышь, панель с клавишами, клавиатура типа thumb-board (клавиатура для больших пальцев обеих рук), сенсорный дисплей, голосовое управление, джойстик, пульт дистанционного управления, специализированные кнопки и прочее, и прочее. Какое сочетание способов подходит вашим ключевым и второстепенным персонажам? Если для продукта уместно сочетание нескольких способов управления (так обычно бывает с веб-сайтами или компьютерными приложениями, рассчитанными на мышь и клавиатуру), выберите для продукта один *ключевой* способ управления.

Шаг 2: определение функциональных и информационных элементов

Функциональные и информационные элементы – это зримые представления функций и данных, доступные пользователю посредством интерфейса. Это конкретные проявления функциональных и информационных потребностей, выявленных на стадии выработки требований. В то время как требования намеренно описывались в общих терминах с позиции персонажей, функциональные и информационные элементы описываются на языке представления этих объектов и действий в интерфейсе. Важно отметить, что каждый такой элемент должен создаваться на основе конкретного требования, выявленного ранее. Тем самым мы добиваемся, чтобы каждый аспект проектируемого продукта имел определенное назначение и соответствовал конкретному аспекту существующего сценария использования или бизнес-цели.

Информационные элементы – это, как правило, фундаментальные объекты интерактивных продуктов. Такие объекты, будь это фотографии, сообщения электронной почты, учетные карточки клиентов или заказов, являются основными единицами, на которые могут ссылаться, реагировать, с которыми могут работать пользователи продукта. В идеале эти объекты должны соответствовать ментальным моделям персонажей. На данном этапе крайне важно создать исчерпывающий каталог информационных объектов, поскольку функциональность продукта часто определяется именно в терминах этих объектов. В поле зрения

проектировщика попадают также значимые атрибуты объектов (к примеру, отправитель сообщения электронной почты или дата создания фотографии), но они на данном этапе представляют меньший интерес. Главное здесь – составить представление о том, какое количество атрибутов интересует персонажей.

Полезно рассмотреть взаимоотношения между информационными элементами. Иногда одни информационные объекты включают в себя другие, в других случаях их объединяют ассоциативные связи. Примерами подобных связей могут служить фотография в альбоме, песня в списке для проигрывания, отдельный счет внутри клиентской записи.

Функциональные элементы – это операции, которые могут выполняться над информационными объектами и представляющими эти объекты элементами интерфейса. В большинстве случаев функциональные элементы представляют собой инструменты, работающие с информационными элементами, а также контейнеры, содержащие информационные элементы. Трансляция функциональных требований в детальные функциональные элементы – та точка, где проектирование начинает обретать конкретность. Контекстный сценарий был способом дать целостное общее представление об опыте, который мы создаем для пользователей; но именно сейчас мы начинаем превращать этот опыт в реальность.

Часто бывает, что одно требование превращается в целый набор элементов интерфейса. Скажем, Вивьен, нашему персонажу проекта интерфейса для смартфона из главы 6, необходимо звонить людям, записанным в телефонной книге. Вот некоторые функциональные элементы, обеспечивающие ее такой возможностью:

- Голосовая активация (голосовые данные, привязанные к контакту из телефонной книги)
- Программируемые кнопки быстрого набора
- Выбор человека из записной книжки
- Выбор на основе заголовка сообщения электронной почты, записи о встрече или заметки
- Аавтоматическое предоставление кнопки вызова в подходящих контекстах (к примеру, при уведомлении о приближающейся встрече)

Повторим: очень важно вернуться к контекстным сценариям, целям персонажей и их ментальным моделям, чтобы убедиться, что ваши решения подходят к рассматриваемым ситуациям. Именно на данном этапе процесса принципы и шаблоны проектирования начинают приносить пользу, способствуя отысканию эффективных решений и избавляя от необходимости изобретать колесо. Здесь проектировщик должен проявить свои творческие способности и применить логику проектирования. Каждое выявленное требование имеет обычно несколько вариантов решения. Какое из возможных решений с наибольшей вероятностью:

- Позволит пользователям эффективно достигать целей?
- Будет лучше других соответствовать нашим принципам проектирования?
- Окажется в рамках бюджета и технологических возможностей?
- Будет лучше других соответствовать прочим требованиям?

Продукт как человеческое существо

Как вы уже знаете из главы 6, существует мощный способ представить идеальный опыт пользователя (который необходимо отразить в контекстных сценариях концептуального уровня) – притвориться, что инструмент, продукт или система – волшебные. Точно так же, притворившись, что система – это *человек*, мы получим мощный инструмент для структурирования деталей уровня взаимодействия. Подробно мы обсудим этот простой принцип в главе 12, а в двух словах – взаимодействие с цифровой системой должно по тону и приносимой пользе напоминать общение с вежливым и внимательным человеком (Cooper, 1999). Определяя варианты взаимодействия и поведения наряду с функциональными элементами и группами, спросите себя: что сделал бы человек, который рад помочь? На что похоже тщательно продуманное и взвешенное взаимодействие? Гуманно ли продукт поступает с ключевым персонажем? Какими способами программа может информировать пользователя, не мешая ему при этом работать? Как она может свести к минимуму усилия персонажа в достижении его целей?

К примеру, мобильный телефон, ведущий себя, как вдумчивый человек, знает, что после завершения разговора с абонентом, отсутствующим в телефонной книжке, возможно, потребуется записать туда его телефонный номер, и потому предоставляет простой и очевидный способ это сделать. Невнимательный к человеку телефон вынуждает пользователя записывать номер на тыльной стороне ладони, после чего лезть в телефонную книгу и создавать новую запись.

Применяйте принципы и шаблоны

Для перевода требований в форму функциональных элементов (а также для группировки этих элементов и детального исследования поведения с помощью сценариев и раскадровок) предельно важным является использование общих принципов взаимодействия и конкретных шаблонов взаимодействия. За этими инструментами стоят годы опыта проектирования взаимодействия. Отказываться от такого знания – значит терять время на проблемы, решения которых уже найдены. Кроме того, отклонение от стандартных шаблонов проектирования может привести к созданию продукта, который заставит пользователей осваивать с нуля все идиомы взаимодействия вместо того, чтобы дать им узнаваемое поведение, знакомое им по другим продуктам и позволяющее использовать прежний опыт (понятие шаблона проектирования мы обсудим в главе 8). Разумеется, иногда уместно изобретать новые

решения известных проблем, но, как мы увидим в главе 14, лучше все же придерживаться стандартов, если только у вас нет достаточно веских причин для обратного.

Сценарии естественным образом подводят нас к проектированию взаимодействия «сверху вниз». Через них итерационно прогоняется инфраструктура интерфейса, последовательно обрастающая деталями, начиная с главных экранов в целом и заканчивая крохотными субпанелями и диалоговыми окнами. Принципы и шаблоны обеспечивают необходимый баланс, предлагая возможность двигаться «снизу вверх». Они могут быть использованы для организации элементов на всех уровнях проектирования. В главе 8 мы подробно поговорим о применении принципов и шаблонов и их типах, а в главах второй и третьей частей книги содержится огромное количество полезных принципов взаимодействия, подходящих для данного этапа проектирования.

Шаг 3: определение функциональных групп и иерархических связей между ними

Создав хороший перечень высокоуровневых функциональных и информационных элементов данных, вы можете начать группировать их в функциональные единицы и выстраивать иерархически (Shneiderman, 1998). Поскольку эти элементы решают конкретные задачи, идея состоит в том, чтобы группировать элементы по принципу наилучшего соответствия рабочему процессу персонажа (см. главу 10) как в пределах одной задачи, так и в рамках взаимосвязанных задач. Вот некоторые вопросы, требующие внимания:

- Какие элементы займут много экранного пространства, а какие нет?
- Какие из элементов являются контейнерами для других элементов?
- Как следует расположить контейнеры, чтобы оптимизировать рабочий процесс?
- Какие элементы используются совместно, а какие нет?
- В какой последовательности используются связанные элементы?
- Какие шаблоны и принципы взаимодействия здесь уместны?
- Как влияют на организацию элементов ментальные модели персонажей?

На этом шаге важно распределить данные и функции по высокоуровневым контейнерам, таким как экраны, фреймы и панели. Эта группировка предварительна и может меняться по мере эволюционирования проекта (особенно в ходе макетирования интерфейса), но все равно полезна, поскольку ускоряет процесс создания первых набросков.

Изучите, какие ключевые состояния или экраны (мы будем называть их **представлениями**) требуются в продукте. Исходные контекстные сценарии подскажут, какие здесь возможны варианты. Если известно,

что у пользователя несколько конечных целей и потребностей с непересекающимися информационными и функциональными элементами, может оказаться полезным определить различные представления для достижения этих целей. С другой стороны, если обнаружилась группа родственных потребностей (к примеру, чтобы назначить встречу, персонажу требуется одновременный доступ к календарю и списку контактов), возможно, есть смысл определить представление, сводящее соответствующие элементы интерфейса воедино.

Группируя функциональные и информационные элементы, обращайте внимание на то, как они должны располагаться исходя из технологической платформы продукта, размера экрана, форм-фактора и способов управления. Контейнеры для объектов, используемых совместно или сопоставляемых в ходе работы, должны располагаться рядом. Объекты, представляющие шаги в процессе, в общем случае должны располагаться друг за другом в последовательности их использования. Применение принципов и шаблонов проектирования взаимодействия исключительно полезно в этих вопросах; в третьей части книги содержатся многочисленные принципы, способные помочь на данном этапе работы.

Шаг 4: макетирование общей инфраструктуры взаимодействия

Теперь мы готовы рисовать наброски интерфейса. Поначалу данный вид визуализации интерфейса должен быть крайне простым. В нашей студии мы часто называем этот момент «фазой прямоугольников», поскольку эскизы начинаются с разделения каждого представления на прямоугольные области, соответствующие панелям, элементам управления (скажем, панелям инструментов) и другим высокоуровневым контейнерам (рис. 7.1). Давайте прямоугольникам названия, показывайте, каким образом одна группа элементов влияет на другие.

Разумно сделать несколько набросков, отражающих различные варианты расположения высокоуровневых контейнеров внутри интерфейса. Начинать следует с самой схематичной визуализации: каждую функциональную группу или контейнер представляет прямоугольник с названием, общую картину дополняют описания связей различных областей (рис. 7.1).

Не пропускайте этап изучения инфраструктуры в целом, не позволяйте деталям отдельных частей интерфейса отвлекать вас (хотя попытки *представить* содержимое каждого из контейнеров помогут решить, как расположить элементы и сколько пространства под них отвести). Для проектирования на уровне отдельных интерфейсных элементов будет еще уйма времени, а попытка заняться ими слишком рано связана с риском утратить при движении вперед логичность интерфейса. Уровень абстрактности «фазы прямоугольников» позволяет с легкостью исследовать массу способов представления информации и функ-

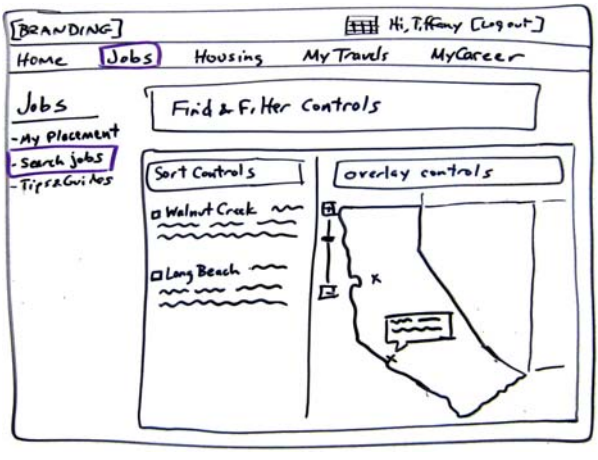


Рис. 7.1. *Пример ранних макетов компоновки: работа студии Cooper для Cross Country TravCorp – интернет-портала для мобильных медсестер. Наброски экранной компоновки должны быть простыми – поначалу достаточно прямоугольников, названий и простых описаний связей между функциональными областями. Можно делать визуальные намеки на детали более низкого уровня, чтобы дать представление о содержимом областей, однако не попадайте в ловушку – не переходите на этом шаге к проектированию деталей*

циональности и при необходимости вносить радикальные изменения. Часто оказывается полезным попробовать несколько вариантов расположения областей и выполнить проверочные сценарии (см. шаг 6 ниже) – и только тогда остановиться на лучшем решении. Трата слишком больших сил и времени на мелкие детали на ранней стадии проектирования затрудняет проектировщикам смену курса в сторону более правильного решения. Выбросить результаты работы и попробовать иной подход легче, когда затраты невелики.

Создание макета общей компоновки – процесс итерационный, и эту работу лучше выполнять небольшой сплоченной группой из графического дизайнера, промдизайнера и одного-двух проектировщиков взаимодействия (в идеале – проектировщика взаимодействия и «коммуникатора» – человека, который думает в терминах текстового описания продукта). По нашему опыту простая белая доска – лучший инструмент для создания первоначальных набросков. Она способствует совместной работе, обсуждениям и, разумеется, позволяет быстро все стереть и нарисовать заново. В таком варианте цифровой фотоаппарат – простое средство моментальной фиксации идей для последующего их рассмотрения.

Когда макеты достигают разумного уровня детализации, можно переходить к использованию компьютерных инструментов рисования. У каждого инструмента есть достоинства и недостатки, но чаще всего для ри-

сования высокоуровневых эскизов интерфейса применяются такие продукты, как Adobe Fireworks, Adobe Illustrator, Microsoft Visio, Microsoft PowerPoint и OmniGraffle от Omni Group. Главное здесь – найти наиболее удобный для вас инструмент, позволяющий быстро работать на высоком уровне, не вдаваясь в детали. Мы находим полезным применять для макетов инфраструктуры такой стиль, который подчеркивает схематичность предложенных решений (вспомним, что грубые наброски лучше всего стимулируют обсуждение проектных решений). Крайне важно также иметь возможность легко получить визуализацию набора последовательно связанных экранов, отражающую поведение продукта при выполнении ключевого сценария (система «фреймов»[1] в Fireworks делает этот инструмент особенно удобным для решения данной задачи).

Шаг 5: создание ключевых сценариев

Ключевой сценарий описывает взаимодействие персонажа с системой в терминах лексикона инфраструктуры взаимодействия. Он отражает магистральные пути внутри интерфейса, используемые персонажем чаще всего (например, ежедневно). Ключевые сценарии сосредоточены на задачах. Скажем, в случае приложения для работы с электронной почтой ключевые действия – это просмотр и создание новых сообщений, а не настройка нового почтового сервера.

Ключевые сценарии, как правило, являются результатом развития контекстных сценариев, но целенаправленно описывают взаимодействие персонажа с различными функциональными и информационными элементами, составляющими общую инфраструктуру взаимодействия. По мере детализации общей инфраструктуры взаимодействия мы дополняем ключевые сценарии таким образом, чтобы они предоставляли больше детальной информации о действиях пользователя и реакциях продукта.

В отличие от контекстных сценариев, которые сосредоточены на целях, ключевые сценарии больше сосредоточены на задачах, намеки на которые или описания которых содержатся в контекстных сценариях. Это не означает, что цели можно игнорировать. Цели и потребности персонажа служат постоянным лакмусом процесса проектирования, отсекающим ненужные задачи и собирающим в единую систему необходимые. Однако **ключевые сценарии** должны с исчерпывающими подробностью и точностью описывать поведение продукта во всех существенных вариантах взаимодействия и документировать все существенные маршруты.

---

[1] В последней версии Adobe Fireworks CS4 фреймы были переименованы в «состояния» – удачное решение с точки зрения применения данного элемента. – *Примеч. науч. ред.*

Раскадровка

Применение набросков с низким уровнем детализации в сочетании с повествовательным ключевым сценарием позволяет создать живое описание того, как предложенное проектное решение помогает персонажам достигать их целей. Техника **раскадровок** заимствована из кинематографа и мультипликации, где сходный процесс применяется для планирования и оценки идей, избавляя от затрат и усилий, связанных со съемкой на пленку. Каждое взаимодействие между пользователем и продуктом можно описать одним или несколькими слайдами. Переход между слайдами позволяет проверить связность и ход взаимодействия (рис. 7.2).

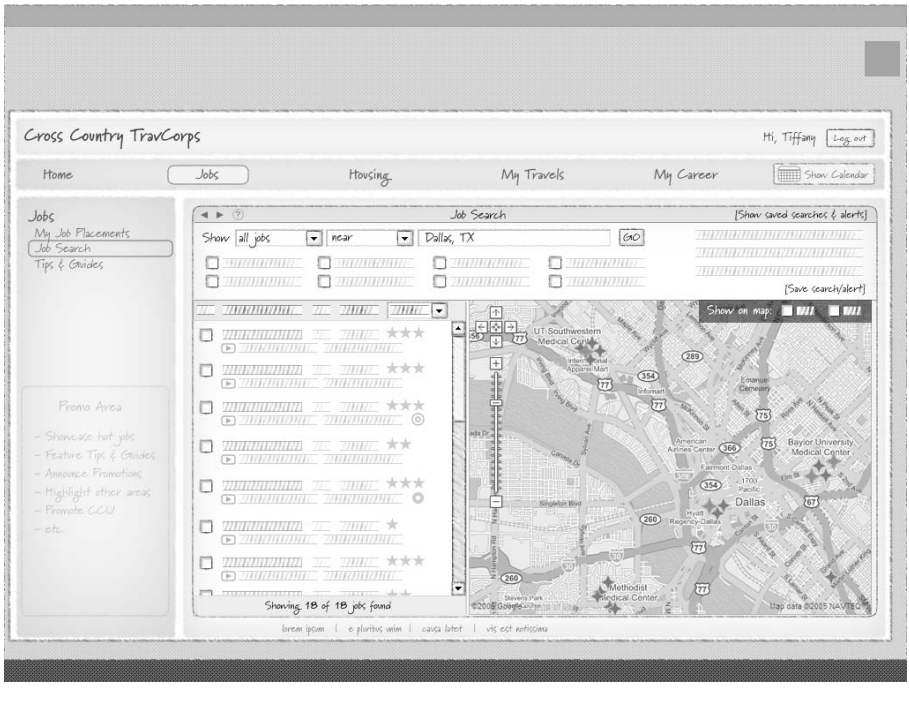


Рис. 7.2. *Пример более подробного варианта компоновки веб-приложения Cross Country TravCorps, связанного с поиском работы*

Цикличность и вариативность процесса

Творческая деятельность человека – процесс чаще нелинейный, поэтому о шагах создания инфраструктуры взаимодействия не следует думать как о простой линейной последовательности. Естественно делать шаги не только вперед, но и назад, а также повторять все шаги по нескольку раз, пока не будет получено качественное решение. В зависимости от образа мышления проектировщика можно выбирать разные

способы выполнения шагов с 3 по 5. Какой-то из способов окажется удобнее прочих.

Проектировщики с *вербальным* типом мышления могут использовать сценарий как движущую силу процесса и выполнять шаги 3–5 в следующей последовательности:

1. Ключевые сценарии.
2. Словесная группировка.
3. Макетирование.

Проектировщики с *визуальным* типом мышления могут обнаружить, что иллюстрация помогает им разобраться с остальными частями процесса. Им будет легче делать так:

1. Макетирование.
2. Ключевые сценарии.
3. Проверка соответствия группировки сценариям.

Шаг 6: выполнение проверочных сценариев для верификации решений

Создав раскадровки ключевых сценариев и скорректировав инфраструктуру взаимодействия, чтобы сценарии выполнялись гладко, а также убедившись, что направление выбрано верно, вы можете обратить внимание на менее востребованные или менее важные варианты взаимодействия. Эти **проверочные сценарии** обычно не детализируются до такой же степени, как ключевые сценарии. Здесь чаще применяются серии вопросов «Что если...». Задача в том, чтобы «потыкать» в различные проектные решения и внести необходимые коррективы (либо выбросить решения и начать заново). Существует три основных разновидности проверочных сценариев, которые следует создавать в следующем порядке:

- **Варианты ключевых сценариев.** Альтернативные или менее востребованные варианты взаимодействия, ответвляющиеся от ключевых маршрутов в результате принятия персонажем решения. Например, это могут быть распространенные исключения, нечасто применяемые инструменты и представления, а также варианты или дополнительные сценарии, основанные на целях и потребностях второстепенных персонажей. Возвращаясь к сценарию для смартфона из главы 6, если Вивьен решит ответить на сообщение Фрэнка на шаге 2, вместо того чтобы звонить ему, это и будет вариантом ключевого сценария.
- **Обязательные сценарии.** Действия, которые должны выполняться обязательно, но нечасто. В эту категорию могут попадать такие действия, как чистка баз данных, настройка, выполнение других исключительных запросов. Обязательные варианты взаимодействия требуют обучения, поскольку пользователи сталкиваются с ними

редко и могут забывать, где расположена функция или как выполнять связанные с ней задачи. Однако редкое использование означает, что пользователям не потребуются дополнительные идиомы взаимодействия, такие как клавиатурные эквиваленты, равно как и возможности настройки для таких функций. Пример обязательного сценария для смартфона: если был куплен подержанный телефон, требуется удалить всю личную информацию прежнего владельца.

- **Сценарии исключительных ситуаций.** Как следует из названия, такие сценарии описывают нетипичные ситуации, с которыми продукт все же должен справляться, пусть даже такая необходимость и возникает нечасто. Программисты концентрируются на исключительных ситуациях, поскольку те часто служат источником нестабильности систем и ошибок и требуют обычно значительного внимания и усилий для обработки. Исключительные ситуации никогда не должны попадать в центр внимания проектировщика. Проектировщики не могут игнорировать исключительные функции и ситуации, но взаимодействие для таких функций и ситуаций имеет низкий приоритет и обычно уходит в интерфейсе на задний план. От способности *кода* обрабатывать исключительные ситуации может зависеть его успех, однако успех продукта напрямую зависит от способности обеспечивать выполнение ежедневных операций и обязательных сценариев. Вернемся снова к смартфону Вивьен (глава 6): если Вивьен попытается добавить в телефонную книжку двух людей с одинаковыми именами, это и будет сценарий исключительной ситуации. Маловероятно, что она это сделает, но если сделает – телефон должен справиться с такой ситуаций.

Создание визуальной инфраструктуры

Параллельно с инфраструктурой взаимодействия, определяющей общую структуру поведения продукта и *связь формы с поведением*, необходимо прорабатывать графический и промышленный дизайн, чтобы подготовиться к детальному проектированию (исключением являются ситуации, когда вы уже располагаете устоявшимся визуальным стилем). Этот процесс подобен созданию инфраструктуры взаимодействия в том смысле, что решения сначала рассматриваются на высоком уровне и лишь затем поэтапно детализируются.

Создание **визуальной инфраструктуры** происходит обычно следующим образом:

1. Провести исследование визуального языка.
2. Применить выбранный визуальный стиль к экранным архетипам.

Шаг 1: исследование визуального языка

Первый шаг в определении визуальной инфраструктуры – изучить разнообразные варианты оформления посредством **исследования ви-**

зуального языка (рис. 7.3). Предметом исследования являются такие переменные, как цвет, шрифт, оформление панелей, а также общие качественные характеристики и «материальные» свойства интерфейса (например, воспринимается ли интерфейс как стеклянный или как бумажный?).

В ходе исследования перечисленные аспекты должны быть изучены отвлеченно и независимо от результата проектирования взаимодействия, поскольку наша цель в данном случае – оценить тон и уместность выбранных средств взаимодействия в целом. Необходимо избегать ситуаций, когда совершенный графический макет грубого еще по сути своей проектного решения вызовет у заинтересованных лиц отторжение.

Исследование визуального языка должно отталкиваться от эмоциональных целей персонажей, а также связанных с опытом, эмоциями и брендом ключевых слов, найденных на стадии выработки требований. Как правило, созданное компанией руководство по визуальному стилю является хорошей отправной точкой для этой деятельности, но следует заметить, что такие принципы редко затрагивают опыт взаимодействия. «Руководство по визуальному стилю» обычно представляет собой документ, объясняющий, как использовать визуальные и текстовые средства для передачи уникальности бренда.

Чтобы перевести маркетинговое руководство по визуальному стилю в осмысленный дизайн интерактивного продукта или веб-сайта, часто требуется проделать большую работу. Создавая визуальный стиль, важно учитывать также особенности среды и склонности персонажей. Экраны, с которыми человек будет работать при ярком освещении или на некотором расстоянии, требуют более контрастных и более насыщенных цветов. Пожилым и близоруким людям требуется более крупный и более читаемый шрифт.

При первом обсуждении с заинтересованными лицами мы обычно показываем от трех до пяти различных вариантов графического дизайна. С проектированием взаимодействия дела обстоят иначе – обычно демонстрируется один оптимальный вариант интерфейса. Что же касается визуальной части, может существовать целый ряд стилей, сопрягающихся с ключевыми словами и целями. И разумеется, «красота – в глазах смотрящего». Мы обнаружили, что зачастую мнение заинтересованных лиц относительно цветов, которые следует применять в интерфейсе, попросту непредсказуемо.

Часто полезно создать один или два крайних варианта, в которых интерфейс имеет перекос в ту или иную сторону. Так проще продемонстрировать различия в подходах, что помогает заинтересованным лицам выбрать направление движения. Позже в ходе проектирования еще будет возможность выправить перекошенный визуальный стиль. При этом все представленные заинтересованным лицам варианты должны быть разумными и подходящими. Если вам не нравится один из вари-

Рис. 7.3. *Исследование визуального языка используется для изучения разнообразных визуальных стилей абстрактно и до некоторой степени без учета результатов проектирования взаимодействия. Тем самым мы получаем возможность обсуждать язык графического дизайна, не отвлекаясь на аспекты взаимодействия. Разумеется, в конечном итоге графический дизайн и проектирование взаимодействия встречаются и начинают шагать в ногу*

антов, именно его выберут заинтересованные лица – это почти неписаное правило.

ПРИНЦИП проектирования

Никогда не демонстрируйте вариант, который не нравится вам самим, потому что именно этот подход может понравиться заинтересованному лицу.

Когда создан набор качественных визуальных стилей, отражающих эмоциональные цели персонажей и выражающих ключевые слова, связанные с брендом и эмоциями, приходит время демонстрации этого набора заинтересованным лицам. Делается это для сбора отзывов. Важно представлять стили в контексте целей и ключевых слов и аргументировать каждый из вариантов дизайна и его относительные выгоды. Мы просим заинтересованных лиц сразу описать свои самые первые эмоциональные реакции, а затем уже обсуждаем все с позиций логики. К концу презентации мы обычно принимаем консолидированное решение и фиксируем положительные аспекты разных визуальных стилей. Обычно требуется провести еще одну итерацию исследования визуального языка, прежде чем переходить к следующему шагу.

Шаг 2: применение выбранного стиля

Следующий шаг заключается в применении одного или двух выбранных стилей к типовым экранам. Здесь требуется скоординированная работа проектировщиков и дизайнеров, поэтому данный шаг выполняется ближе к финалу работы над общей инфраструктурой проектирования, когда графический дизайн начал стабилизироваться, а инфраструктура взаимодействия стала настолько подробной, что можно продемонстрировать визуальный стиль. Данная операция приводит к детализации визуального стиля, который теперь отражает ключевые информационные и поведенческие аспекты интерфейса. Делая дизайн более конкретным, вы можете лучше оценить жизнеспособность предложенного решения, и для этого не требуется обновление многочисленных экранов после каждого мелкого изменения. Кроме того, так легче получать отзывы заинтересованных лиц.

Создание физической инфраструктуры

Физическая инфраструктура создается во многом так же, как визуальная инфраструктура, но из-за того, что форм-фактор и способ управления существенным образом влияют как на промышленный дизайн, так и на графический дизайн, полезным оказывается раннее сотрудничество с целью выявления возможных проблем.

Формирование физической инфраструктуры обычно выполняется следующим образом:

1. Совместно с проектировщиками взаимодействия обсудить форм-фактор и способы управления.

2. Создать грубые прототипы.
3. Выполнить исследование языка формы.

Шаг 1: совместно с проектировщиками взаимодействия обсудить форм-фактор и способы управления

Если проектируемый продукт требует создания специализированной аппаратной части (как в случае сотового телефона или медицинского прибора), важно, чтобы проектировщики взаимодействия и промышленные дизайнеры выбрали первоначальную физическую форму и способы управления. В ходе работы над визуальной инфраструктурой продукт, несомненно, обретет более интересное физическое выражение, но именно в этот момент пора принимать решение о габаритах и общей форме продукта, размерах экрана (если таковой имеется), количестве и ориентации функциональных и программируемых кнопок, о том, требуется ли сенсорный ввод, клавиатура, распознавание голоса и так далее. Такое сотрудничество начинается обычно с пары дней, проведенных рядом с белой доской вместе со сжатым набором сценариев.

Принимая решения, следует помнить, в частности, об эмоциональных целях персонажа (см. главу 5), взглядах, привычках, факторах среды, а также о ключевых словах, связанных с опытом и брендом, исследованиях рынка, стоимости производства и желаемой розничной стоимости продукта. Стоимость шарнира может привести к превышению бюджета по аппаратной части, а внутренние компоненты (такие как аккумуляторы) способны оказывать значительное влияние на форму, так что ранние проверки реалистичности вариантов продукта, проведенные совместно с инженерами-механиками и электриками, обязательны.

Пользовательский опыт является целостным и определяется сочетанием физической формы и интерактивного поведения продукта. Эти два аспекта следует проектировать согласованно, а в соответствии со старой поговоркой современной архитектуры – форма следует за функцией. Промышленный дизайн должен принимать во внимание потребности взаимодействия, однако сложность и стоимость производства в свою очередь служат ограничивающими факторами при проектировании взаимодействия.

Пользовательский опыт целостен – форму и поведение продукта следует проектировать согласованно.

Шаг 2: создать грубые прототипы

Часто бывает так, что даже после определения первоначальной формы и способов ввода у промышленных дизайнеров остается масса направлений работы. К примеру, проектируя офисные телефоны и медицинские приборы, мы часто сталкивались с вопросом о том, следует ли

сделать угол наклона экрана фиксированным или регулируемым, а если регулируемым, то каким образом. Промдизайнеры готовят эскизы и создают грубые прототипы из пеноблоков и прочих материалов. Во многих случаях прототипы демонстрируются заинтересованным лицам, поскольку каждый из прототипов связан с определенными затратами и соображениями эргономики.

Шаг 3: выполнить исследование языка формы

Как и в случае с описанным выше изучением визуального стиля, необходимо изучить разнообразные физические стили. В отличие от исследования визуального языка здесь речь идет не об абстракциях, а о конкретных вариантах внешнего представления с учетом выбранного форм-фактора и способа управления. Анализируются форма, размеры, материалы, цвет, отделка.

Исследование языка формы следует производить исходя из целей персонажа, его взглядов, привычек, ключевых слов, связанных с опытом, факторов среды, производственных ограничений и ограничений цены. Как правило, требуется провести несколько итераций исследования, чтобы получить желанное для пользователей и в то же время осуществимое решение.

Детализация формы и поведения

Сформировав целостную и последовательную общую инфраструктуру взаимодействия, проектировщики замечают, что оставшиеся части системы начинают постепенно вставать на свои места: каждая итерация проработки ключевых сценариев добавляет подробности, укрепляющие целостность продукта. Теперь можно плавно переходить к стадии **детализации**, где проект окончательно обретает конкретные формы.

На этой стадии принципы и шаблоны сохраняют свою важность и используются для проработки деталей формы и поведения продукта. Полезные на стадии детализации принципы описаны во второй и третьей частях книги. Кроме того, важно, чтобы к этапу детализации с самого начала были подключены разработчики: теперь, когда проект обрел целостный концептуальный фундамент и принципы взаимодействия, вклад разработчиков в создание законченной спецификации, воплощающей замысел и одновременно технически осуществимой, оказывается неоценимым.

На этапе детализации происходит перевод раскадровок в полноценные экраны, отражающие пользовательский интерфейс с точностью до отдельных пикселов (рис. 7.4).

В основе процесса детализации лежат те же шаги, посредством которых мы создавали общую инфраструктуру проектирования, но детализация на этот раз выше (хотя, конечно же, нет необходимости возвращаться к форм-фактору и способам управления, если только не

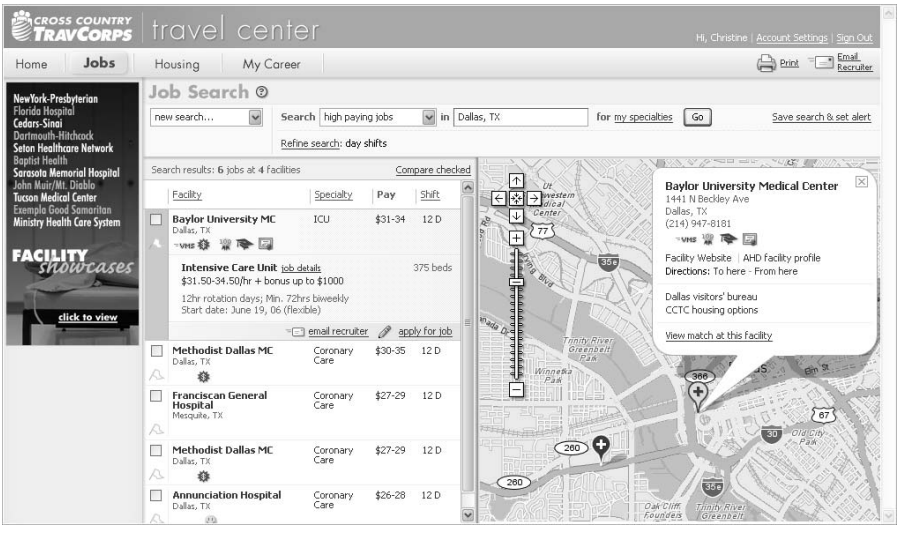


Рис. 7.4. *Полноценные экраны для Cross Country TravCorps, основанные на иллюстрациях, созданных в программе Framework (рис. 7.2). Обратите внимание, что в макете интерфейса произошли небольшие естественные изменения, связанные с реалиями пиксельной графики и экранного разрешения. Дизайнеры и проектировщики взаимодействия на этой стадии в тесном сотрудничестве добиваются того, чтобы все изменения в дизайне подчеркивали те или иные варианты поведения продукта и соответствовали целям ключевых персонажей*

возникли непредвиденные сложности, связанные с производством или стоимостью аппаратной части). Выполнив шаги со 2 по 6 для представлений и панелей в процессе уточнения визуального и промышленного дизайна, примените сценарии для работы над более мелкими составляющими продукта.

Проработайте все возможные представления и диалоговые окна. На этапе детализации дизайнерам следует создать и поддерживать в актуальном состоянии руководство по визуальному стилю. Программисты, используя такое руководство, могут корректно применять элементы дизайна, создавая низкоприоритетные части интерфейса, на которые у проектировщиков обычно не хватает времени и ресурсов. Одновременно с этим промдизайнеры совместно с инженерами работают над финальным «конструктором» и его сборкой.

Конечные результаты проектирования могут быть самыми разнообразными, но, как правило, мы создаем спецификации формы и поведения продукта в печатном виде. Этот документ включает эскизы экранов с пояснениями, достаточно детальными для того, чтобы программисты могли писать код, а также подробные раскадровки, иллюстрирующие поведение продукта в динамике. Может оказаться полезным

создать еще и интерактивный прототип на основе технологии HTML или Flash – он дополнит документацию и более доходчиво проиллюстрирует некоторые нюансы взаимодействия. При этом следует помнить, что одних лишь прототипов обычно недостаточно, чтобы передать неявные шаблоны, принципы и обоснования решений, – а их жизненно необходимо передать программистам. Независимо от выбранного формата документации команда проектировщиков должна работать в тесном сотрудничестве с разработчиками вплоть до завершения реализации. Требуется бдительность, чтобы гарантировать, что результаты проектирования точно и без искажений превратятся в законченный продукт.

Проверка результата проектирования и юзабилити-тестирование

В ходе проектирования взаимодействия зачастую бывает желательно оценить, насколько качественными являются те решения, которые появились на свет. Для этого требуется выйти за пределы персонажей и проверочных сценариев и предложить решения реальным пользователям. Делать это следует тогда, когда решение обрело достаточную детализацию, чтобы пользователи могли реагировать на него как на что-то вполне конкретное, но при этом есть еще достаточно времени, чтобы внести исправления исходя из результатов тестирования.

По опыту авторов, юзабилити-тесты и сеансы, направленные на получение отзывов пользователей, хорошо подходят для выявления крупных проблем с инфраструктурой взаимодействия и для улучшения таких вещей, как надписи на кнопках, порядок и приоритет действий. Они также жизненно важны для настройки таких аспектов поведения, как скорость прокрутки экрана в качестве реакции на поворот аппаратной ручки. К сожалению, сложно создать тест, оценивающий что-либо, кроме простоты освоения решения новичками. Существует ряд методов для оценки удобства продукта в использовании середняком или экспертом, но такие методы отнимают много времени и дают в лучшем случае не слишком точные результаты.

Есть множество способов проверить интерфейс на реальных пользователях – от неформальных сеансов обратной связи, когда вы поясняете идеи, показываете рисунки и выслушиваете соображения пользователя, до более строгого *юзабилити-тестирования*, когда пользователям предлагается решить определенный заранее набор задач. Каждый подход имеет свои преимущества. Более неформальные методы могут применяться спонтанно и требуют меньшей подготовки. Минус такого подхода в том, что проектировщик часто оказывается виновен в «подсказках свидетелю», если объясняет в побуждающей манере. В целом мы находим, что этот подход приемлем для технической аудитории, которая способна домысливать немногочисленные рисунки, представ-

ляющие интерфейс продукта. Он может быть достойной альтернативой юзабилити-тесту, если у команды проектировщиков нет времени для подготовки к формальному юзабилити-тестированию.

Если же времени достаточно, мы отдаем предпочтение более формальному юзабилити-тестированию. Юзабилити-тесты определяют, насколько хорошо найденное решение позволяет пользователям решать свои задачи. Если область тестирования достаточно широка, вы узнаете также, насколько хорошо решение помогает пользователям в достижении конечных целей.

Чтобы не было недоразумений, уточним, что юзабилити-тестирование по сути своей есть средство *анализа*, а не *синтеза*. Оно не служит альтернативой проектированию взаимодействия и никогда не будет являться источником великолепных идей, приводящих к созданию привлекательного продукта. Скорее это метод оценки эффективности существующих идей и инструмент для их оттачивания.

Кроме того, юзабилити-тестирование – это не исследование пользовательской аудитории. Некоторые практики считают, что «тестирование» может включать исследования (интервью, анализ задач) и даже творческие упражнения вроде «проектирования с привлечением пользователей». Однако это – попытка свести в один вид деятельности различные потребности и шаги процесса проектирования.

Изучение пользовательской аудитории должно проводиться *до* генерации идей, а юзабилити-тестирование *после*. Более того, мы обнаружили, что когда проектные ограничения заставляют нас выбирать между этнографическими исследованиями и юзабилити-тестированием, то время, потраченное на исследования, дает нам больше возможностей для создания привлекательного продукта. Точно так же, располагая ограниченными временем и бюджетом, мы выяснили, что время, потраченное на проектирование, привносит в процесс большую ценность, нежели тестирование. Лучше проводить больше времени, принимая взвешенные, основанные на серьезном исследовательском фундаменте решения в области проектирования, чем тестировать полуготовые решения, не пользуясь преимуществами прозрачных и удобных моделей пользователей, их потребностей и целей.

Когда тестировать: полное и промежуточное тестирование

В своей книге «Usability Engineering» (Nielsen, 1993) Якоб Нильсен различает *полное тестирование*, когда тестируются завершенные продукты, и *промежуточное тестирование*, проводимое в ходе проектирования как часть итерационного процесса. Это важное различие.

Полное тестирование применяется для сравнения продуктов, для выявления проблем перед перепроектированием, для расследования причин возвратов продукта и определения источника запросов на обучение

и поддержку. Общие исследования, как правило, проводятся независимыми профессионалами и подробно документируются. В некоторых случаях, особенно при конкурентном анализе, общие исследования строятся таким образом, чтобы получить количественные данные, которые можно проверить на статистическую значимость.

К сожалению, полное тестирование часто используется в составе процесса обеспечения качества – ближе к завершению разработки. В этот момент уже обычно слишком поздно вносить осмысленные изменения в проектные решения: поезд ушел. Оценку пользовательского интерфейса следует проводить до того, как начнется создание кода (или, по крайней мере, достаточно рано, чтобы было время изменить реализацию сообразно внесенным дополнениям). Однако если требуется убедить заинтересованных лиц или программистов в том, что проблема юзабилити в имеющемся продукте *явно* присутствует, то ничто не заменит наблюдения за реальными пользователями, сражающимися с программным продуктом.

Промежуточное тестирование предназначено как раз для этого. Эти быстрые качественные тесты проводятся в ходе проектирования, обычно на этапе детализации. Продуманное и контролируемое промежуточное тестирование открывает окно в разум пользователя, позволяя проектировщикам видеть, как их целевая аудитория реагирует на информацию и инструменты, предоставленные им для решения задач.

Хотя у полного тестирования, безусловно, есть своя область применения, оно относится к сфере информационной поддержки планирования жизненного цикла продукта. Оно позволяет проводить «антикризисные проверки» в ходе разработки, но потери в деньгах, времени, боевом духе команды могут на данном этапе дорого обойтись. Промежуточное тестирование проводится во благо проектирования в ходе процесса проектирования.

Проведение промежуточного юзабилити-тестирования

Существует много мнений относительно того, как правильно проводить юзабилити-тестирование и интерпретировать его результаты. К сожалению, мы обнаружили, что многие из подходов либо метят на место проектирования, либо имеют перекос в сторону количественных метрик и дают бесполезные данные вроде «времени выполнения задачи». Работа Кэролин Снайдер «Paper Prototyping» (Snyder, 2003) – хороший справочник по методам юзабилити-тестирования, причем, как мы убедились, совместимый с методами целеориентированного проектирования. В ней описаны не все методы тестирования и не затронуты отношения между тестированием и проектированием, но достаточно подробно рассмотрены основы и даются некоторые относительно простые методики для юзабилити-тестирования.

Вот вкратце важные составляющие успешного промежуточного юзабилити-тестирования:

- Проводите тестирование достаточно поздно, когда уже существует конкретное решение проектирования, но и достаточно рано, чтобы можно было успеть скорректировать проект и реализацию.
- Тестируйте задачи и аспекты опыта пользователя, связанные с конкретным продуктом.
- Набирайте участников из аудитории целевых пользователей, используя персонажей в качестве фильтра.
- Четко ставьте перед участниками задачи и просите при их решении размышлять вслух.
- Дайте участникам возможность напрямую взаимодействовать с низкотехнологичным прототипом (исключением являются случаи, когда тестируется специализированное оборудование, а бумажный прототип не способен отразить нюансы взаимодействия).
- Управляйте ходом сеансов, чтобы выявить проблемы и изучить причины их возникновения.
- Минимизируйте предвзятость, введя в эксперимент модератора, который ранее не принимал участия в проекте.
- Сосредоточьтесь на поведении и образе мысли участников.
- По завершении тестов проведите разбор (дебрифинг) вместе с наблюдателями, чтобы выявить причины наблюдавшихся проблем.
- Вовлекайте в процесс исследования проектировщиков.

Вовлеченность проектировщика в юзабилити-исследования

Недопонимание между проектировщиком и пользователем – распространенная причина проблем юзабилити. Персонажи помогают проектировщикам понять цели пользователей, их потребности и их точку зрения, формируя фундамент эффективных коммуникаций. Юзабилити-исследование, открывая дополнительное окно в мышление пользователя, позволяет проектировщикам увидеть, как доносятся их вербальные, визуальные и поведенческие решения, узнать намерения пользователей при взаимодействии со спроектированными ожидаемыми назначениями.

Проектировщики (или, говоря шире, люди, принимающие решения по проектированию) – основные потребители результатов юзабилити-исследований. Редкие проектировщики способны модерировать сеанс тестирования, сохраняя нейтралитет, однако их вовлечение в планирование исследований и прямое наблюдение за сеансами тестирования, а также участие в анализе и решении проблем крайне важно для

успеха исследования. Мы обнаружили, что проектировщиков важно привлекать к следующим видам деятельности:

- Планирование исследований – для формулирования важных с точки зрения проектирования вопросов
- Определение критериев отбора участников с помощью персонажей и их атрибутов
- Использование сценариев для выбора тестовых задач
- Наблюдение за сеансами тестирования
- Совместный анализ результатов исследований

II

Проектирование облика и поведения

Глава 8. Создание качественного интерфейса: принципы и шаблоны
Глава 9. Техническая платформа и тип интерфейса
Глава 10. Оркестровка и состояние потока
Глава 11. Оптимизация налогообложения
Глава 12. Проектирование хорошего поведения
Глава 13. Метафоры, идиомы, ожидаемое назначение
Глава 14. Визуальный дизайн интерфейсов

8

Создание качественного интерфейса: принципы и шаблоны

В последних четырех главах мы говорили о том, в какой последовательности следует принимать решения, чтобы можно было задумать и спроектировать желанный и эффективный продукт. Но как принимать эти решения? Какие факторы влияют на успех продукта? Как уже говорилось, мерилом качества проектирования как процесса служит степень, в которой дизайн отвечает целям и потребностям пользователей (при этом не принося в жертву цели бизнеса и принимая в расчет технические возможности). Но есть ли у проектирования четко различимые свойства, благодаря которым оно становится успешным? Можем ли мы вывести общие решения для сходных проблем? Существуют ли универсальные признаки, которыми должно обладать проектирование, чтобы стать «качественным»?

Ответы на эти вопросы непосредственно связаны с применением принципов и шаблонов проектирования взаимодействия. **Принципы** проектирования – это рекомендации по проектированию полезных и желанных продуктов, систем и услуг, а также рекомендации по успешному и этичному проектированию. **Шаблоны** проектирования – это типовые обобщенные решения конкретных классов проблем проектирования.

Принципы проектирования взаимодействия

Принципы проектирования взаимодействия – это обобщенные рекомендации, ориентированные на особенности поведения, формы и содержания продукта. Они поддерживают проектирование такого поведения продуктов, которое служит потребностям и целям пользователей и вызывает у них положительные эмоции при использовании этих продуктов. По сути, эти принципы представляют собой набор правил, ос-

нованных на ценностях, носителями которых мы являемся как проектировщики, и на нашем опыте, связанном с претворением этих ценностей в жизнь. В основе наших ценностей – мысль о том, что технология должна служить разуму и творчеству человека (а не наоборот) и что опыт общения человека с технологией должен структурироваться сообразно возможностям человеческого восприятия, познания и движения.

Принципы применяются на всем протяжении процесса проектирования, помогая нам преобразовывать задачи и требования, возникающие в ходе разработке сценариев, в формализованные структуры и поведенческие реакции интерфейса.

Принципы действуют на различных уровнях детализации

Принципы проектирования действуют на нескольких уровнях детализации – от общей практики проектирования взаимодействия до конкретики интерфейса. Границы между уровнями, мягко говоря, размыты, однако принципы проектирования взаимодействия можно в целом разбить на следующие категории.

- **Ценности проектирования** – императивы эффективной и этичной практики проектирования. Эти принципы, о которых мы поговорим далее в этой главе, служат отправной точкой для принципов более низкого уровня из категорий, перечисленных ниже.
- **Концептуальные принципы** помогают определять *сущность продукта* и его место в более широком контексте использования, который требуется пользователям. Принципы концептуального уровня описываются в главах 3, 9 и 10.
- **Поведенческие принципы** описывают, как *продукт должен себя вести* – в целом и в конкретных ситуациях. Поведенческие принципы описываются в главах с 8 по 20.
- **Интерфейсные принципы** описывают эффективные стратегии визуальной коммуникации поведенческих и информационных аспектов интерфейса. К этому уровню проектирования относятся принципы из глав 13 и 14, а также некоторая информация из глав второй и третьей частей книги.

Большинство принципов проектирования взаимодействия и визуального дизайна не привязаны к конкретной платформе, хотя некоторые платформы, например мобильные устройства и системы реального времени, требуют особых соображений, связанных с такими ограничениями, как размер экрана, способы ввода или контекст применения.

Поведенческие принципы и интерфейсные принципы сокращают трудозатраты

Одно из главных назначений принципов – оптимизировать опыт пользователя, взаимодействующего с системой. В случае офисных инстру-

ментов и других продуктов, не ориентированных на развлечения, такая оптимизация опыта означает *минимизацию трудозатрат*. Сокращать следует такие виды работ:

- **Когнитивная работа** – понимание поведения продукта, а также текста и организующих структур.
- **Мнемоническая работа** – запоминание поведения продукта, векторов команд, паролей, названий и расположения объектов данных и элементов управления, а также других связей между объектами.
- **Работа зрения** – поиск стартовой точки на экране, поиск одного объекта среди многих, расшифровка визуальной планировки, выявление различий между элементами интерфейса, имеющими цветовую кодировку.
- **Физическая работа** – число нажатий на клавиши, перемещения мыши, использование жестов (щелчок, перетаскивание, двойной щелчок), переключение между режимами ввода, количество щелчков для осуществления навигации.

Большинство описанных в этой книге принципов направлены на минимизацию работы и в то же время на обеспечение более качественной обратной связи и лучшего информирования пользователя в рамках определенной ситуации.

Следует также заметить, что некоторые развлекательные продукты (в частности, игры) способны завоевывать внимание пользователей, требуя от них выполнения разумного объема определенной работы и вознаграждая их за эту работу. Вспомните помешательство на тамагочи в конце 90-х годов прошлого века: у людей вырабатывалась зависимость от манипуляций по уходу за карманными виртуальными любимцами. Разумеется, слишком большие объемы работы и слишком незначительные вознаграждения способны превратить игру в неприятное занятие. Проектирование взаимодействий подобного рода требует тонкого подхода.

Ценности проектирования

Принципы – это правила, ведущие к действиям и обычно опирающиеся на ряд ценностей и убеждений. Приведенный ниже набор ценностей создали Роберт Рейман, Хью Дабберли (Hugh Dubberly), Ким Гудвин, Дэвид Фор (David Fore) и Джонатан Корман (Jonathan Korman). Он применим к любой дисциплине проектирования, которая служит потребностям человека.

Задача проектировщиков взаимодействия – создавать такие проектные решения, которые:

- **Этичны** [*тактичны, заботливы*]:

  не причиняют вреда;

  улучшают положение человека.

- **Целенаправленны** [*полезны, применимы*]:
  помогают пользователям решать их задачи и достигать целей;
  учитывают контексты и возможности пользователей.
- **Прагматичны** [*жизнеспособны, осуществимы*]:
  помогают организации, внедряющей ваши проектные решения, достигать своих целей;
  учитывают требования бизнеса и технические требования.
- **Элегантны** [*эффективны, искусны, вызывают эмоции*]:
  представляют собой простые, но полноценные решения;
  обладают внутренней (самоочевидной, понятной) целостностью;
  учитывают и пробуждают эмоции и познавательные процессы.

Рассмотрим эти ценности подробнее.

Проектирование этичного взаимодействия

С этическими вопросами проектировщики взаимодействия сталкиваются, когда их просят спроектировать систему, оказывающую серьезное влияние на жизнь людей. Это может быть непосредственное воздействие на пользователей системы или косвенное влияние на людей, жизнь которых каким-либо образом оказывается затронута системой. Для проектировщиков взаимодействия этот вопрос может быть особенно животрепещущим, поскольку результаты их работы, в отличие от результатов работы графических дизайнеров, – не просто убедительная передача определенной стратегии продвижения продукта. От их работы в действительности зависит реализация этой стратегии, или собственно создание продукта. Говоря коротко, интерактивные продукты *обладают возможностью изменять мир*, и проектировщики должны стремиться к тому, чтобы это были изменения *к лучшему*. Относительно легко спроектировать систему, которая хорошо относится к пользователям, однако вычислить, какое косвенное влияние система оказывает на других людей, гораздо сложнее.

Не навреди

Продукты не должны никому причинять вреда – или, принимая во внимание сложность реального мира, должны *минимизировать ущерб*. Вот некоторые разновидности вреда, в причинении которого может принимать участие интерактивная система:

- **межличностный** вред (потеря достоинства, оскорбления, унижение);
- **психологический** вред (замешательство, дискомфорт, раздражение, давление, скука);
- **физический** вред (боль, травмы, лишения, смерть, угрозы безопасности);
- **экологический** вред (загрязнение, сокращение числа биологических видов);

- **социальный и общественный** вред (эксплуатация, создание или поддержание несправедливости).

Предотвращение двух первых разновидностей вреда требует глубокого понимания аудитории пользователей, а также убежденности заинтересованных лиц в том, что такие вопросы могут решаться в рамках проекта. Многие из идей, содержащихся во второй и третьей частях этой книги, способны помочь проектировщикам в создании решений, которые оказывают поддержку человеческому разуму и эмоциям. Предотвращение физического вреда требует глубокого понимания эргономических принципов и соответствующего использования элементов интерфейса с целью минимизации работы. Подробная информация по этой теме содержится в третьей части книги. Очевидно, что последние два пункта к большинству продуктов не относятся, однако читатель наверняка может придумать такие примеры, когда они будут иметь значение, например система управления нефтяной платформой или система электронного голосования.

Улучшай положение человека

Чтобы результатом проектирования стало действительно этичное взаимодействие, мало просто не вредить – требуется способствовать. Вот некоторые улучшения, которым в широком смысле могут содействовать интерактивные системы:

- **улучшение понимания** (индивидуального, социального, культурного);
- **повышение эффективности/действенности** отдельных личностей и групп;
- **совершенствование коммуникаций** – как между отдельными личностями, так и между группами людей;
- **снижение социокультурной напряженности** между личностями и группами;
- **умножение справедливости** (финансовой, социальной, правовой);
- **сглаживание культурных противоречий** путем стимулирования общественного согласия.

Проектировщикам, начинающим работу над новым проектом, необходимо в глубине души осознавать эти глобальные вопросы. Следует всегда предоставлять людям возможность творить добро, даже если это немного выходит за рамки проекта.

Проектирование целенаправленного взаимодействия

Основная тема этой книги – целенаправленное проектирование, основанное на понимании целей и мотивов пользователей. Целеориентированный процесс, описанный в главах первой части, как минимум, должен способствовать целенаправленному проектированию. Целенаправленность – это не только понимание целей пользователей, но и осозна-

ние их ограничений. В этом смысле персонажи служат качественной меркой, поскольку шаблоны поведения, которые вы сможете наблюдать в ходе исследований и при создании персонажей, дадут вам хорошее представление о сильных *и* слабых сторонах пользователей. Целеориентированное проектирование помогает проектировщикам создавать продукты, которые поддерживают пользователей в том, в чем они слабы, и делают их более производительными в том, в чем они сильны.

Проектирование прагматичного взаимодействия

Проектные спецификации, собирающие пыль на полках, бесполезны: чтобы иметь ценность, проектирование должно воплощаться в продукте. Будучи созданным, продукт должен увидеть свет. А увидев свет, он должен приносить прибыль создателям. Крайне важно, чтобы цели бизнеса и технические требования учитывались в ходе проектирования. Это не означает, что проектировщики должны принимать как должное все, что говорят заинтересованные лица и разработчики, – необходим живой диалог, в котором участвуют представители бизнеса, инженерная часть команды, проектировщики, – диалог, позволяющий договориться о том, какие области определения продукта заданы жестко, а какие являются гибкими. Разработчики часто заявляют, что воплощение спроектированного *невозможно*, а в действительности имеют в виду, что это *невозможно в рамках существующего плана-графика*. Маркетинговые организации могут создавать основанные на сводных статистических данных бизнес-планы, не понимая полностью, как поведут себя отдельные пользователи и клиенты. Проектировщики, собравшие подробные, качественные данные о пользователях, могут обладать собственным уникальным взглядом на бизнес-модель. Проектирование оказывается наиболее эффективным, когда существует взаимное доверие и уважение между проектировщиками, бизнесменами и инженерами.

Проектирование элегантного взаимодействия

Элегантность в словаре определяется как «грациозность и сдержанная красота стиля» или «научная точность, аккуратность и простота». Авторы считают, что элегантность в проектировании – или, по меньшей мере, в проектировании взаимодействия – включает оба этих идеала.

Создавай простые, но полноценные решения

Один из классических элементов качественного дизайна – это *экономия формы*: достижение большего меньшими затратами. В проектировании интерфейсов этот принцип экономии гласит, что интерфейс должен содержать лишь те экраны и элементы, которые требуются для решения поставленной задачи. Это касается и поведения: следует дать пользователю простой набор инструментов, позволяющий ему добиваться великолепных результатов. В визуальном дизайне подобная экономия означает использование минимального числа визуальных

отличий для четкой передачи нужного смысла. В хорошем проектировании «меньше» означает «больше», и проектировщики должны стремиться разрешать проблемы проектирования с минимальными добавлениями к форме и поведению в соответствии с ментальными моделями персонажей. Такой подход хорошо знаком программистам, которые соглашаются, что лучшие алгоритмы коротки и понятны.

Известный турист и основатель компании Patagonia, производящей одежду для туристов, Ивон Шунар (Yvon Chounard) очень метко процитировал французского писателя и авиатора Антуана де Сент-Экзюпери: «абсолютно во всем совершенство достигается не тогда, когда уже нечего добавить, но когда уже ничего нельзя отнять».

Добивайся внутренней целостности

Качественно спроектированный продукт оставляет ощущение единого целого, в котором все составные части сбалансированы и гармоничны. Некачественно спроектированный продукт или продукт, вообще не проектировавшийся, обычно выглядит и создает впечатление, будто его наспех сколотили из выбранных случайным образом разрозненных кусков. Часто такой продукт – результат конструирования на основе модели реализации, когда различные команды разработчиков трудятся над различными модулями интерфейса, не общаясь между собой, или когда программная и аппаратная части проектируются раздельно. Это полная противоположность тому, к чему мы стремимся. Целеориентированный процесс проектирования, в котором концепция продукта зарождается сразу и полностью на высшем уровне с последующей поэтапной детализацией, дает идеальную среду для создания внутренне целостных проектов. Применение сценариев как точки отсчета проектирования и тестирования решений гарантирует, что решения будут пронизаны единой нитью повествования.

Учитывай и пробуждай эмоции и познавательные процессы

Многие проектировщики традиционной школы часто говорят о *желаниях* и важности желаний при проектировании коммуникаций и продуктов. Они не ошибаются, однако авторам кажется, что, делая такой сильный упор на одну (хотя и сложную) эмоцию, они временами теряют из виду полную картину.

Желание – ограниченная эмоция, если речь идет о продукте, у которого есть определенное назначение – особенно если этот продукт используется в корпоративной среде или же его назначение сугубо техническое либо узкоспециальное. Вряд ли стоит пытаться сделать так, чтобы оператор, работающий с прибором радиотерапии, начал желать эту систему. Мы хотим, напротив, чтобы он проявлял осторожность и, возможно, испытывал почтение к опасной энергии, подчиненной системе. Так что мы как проектировщики делаем все возможное, чтобы этот человек сосредоточился на пациенте и лечении. Поэтому вместо того, что можно назвать *желанием*, мы предлагаем элегантность (в смысле

грациозности), которая означает, что пользователь получает стимулы и поддержку – как когнитивные, так и эмоциональные – независимо от контекста, в котором он находится.

В оставшихся главах книги мы расскажем о том, что именно считаем самыми важными принципами в проектировании взаимодействия и интерфейса. Есть, несомненно, и другие, с которыми вы познакомитесь в своей работе, но описанных нами принципов для начала должно быть достаточно. В главах первой части мы описали процесс и идеи, лежащие в основе практики целеориентированного проектирования. В последующих главах мы подвергаем переосмыслению многие моменты проектирования, чтобы дать вам понимание, которое поможет преобразовать приобретенные знания в замечательные продукты в любой предметной области.

Шаблоны проектирования взаимодействия

Шаблоны проектирования суть решения целых классов проблем проектирования, возникающие путем выявления и обобщения ценных проектных находок. Эта деятельность по формализации знания и фиксации наилучших решений в области проектирования служит многим важным целям:

- сократить время и усилия, затрачиваемые на проектирование в новых проектах;
- повысить качество проектных решений;
- способствовать улучшению коммуникации между проектировщиками и программистами;
- повысить профессиональный уровень проектировщиков.

Хотя аспекты применения шаблонов, связанные с обучением проектированию и совершенствованием процесса, очень важны, шаблоны проектирования взаимодействия в первую очередь ценны потому, что служат представлением оптимальных или близких к оптимуму взаимодействий пользователя с теми сферами деятельности, на которые эти шаблоны ориентированы.

Архитектурные шаблоны и проектирование взаимодействия

Идея выявления шаблонов проектирования взаимодействия уходит корнями к замыслу Кристофера Александера, который первым описал шаблоны архитектурного проектирования в своих новаторских работах «A Pattern Language» (Alexander, 1977) и «The Timeless Way of Building» (Alexander, 1979). Определив жесткий набор архитектурных особенностей, Александер стремился выразить ту суть архитектурного проектирования, которая создает ощущение благополучия у людей, находящихся в зданиях.

Именно этот человеческий элемент замысла Александера так хорошо резонирует с потребностями проектировщиков взаимодействия, и именно фокус на человеческом аспекте каждого шаблона отличает шаблоны проектирования взаимодействия (и шаблоны архитектурного проектирования) от инженерных шаблонов, единственная забота которых – эффективное повторное использование и стандартизация кода.

Отдельное важное свойство, отличающее шаблоны проектирования взаимодействия от шаблонов архитектурного проектирования, – это их направленность не только на структуру и организацию элементов, но также на динамику поведения и изменения элементов вследствие деятельности пользователя. Существует соблазн рассматривать это отличие просто как изменение во времени, однако такие изменения интересны тем, что происходят в ответ на деятельность человека и зависят от текущего состояния приложения. Это отличает их от предопределенных переходов, которые можно обнаружить в механических продуктах, средствах массового вещания и фильмах (у которых собственный набор шаблонов проектирования).

Определение шаблонов проектирования взаимодействия и их использование

Шаблоны всегда применяются в рамках некоторого контекста и конструируются так, чтобы быть применимыми в типичных ситуациях, которые имеют схожий контекст использования, схожие ограничения и условия. Описывая шаблон, важно четко задать ситуацию, в которой применимо решение, дать один или несколько конкретных примеров, перечислить абстрактные признаки, характерные для всех примеров, а также рассуждения, объясняющие, почему решение является хорошим.

Чтобы приносить пользу, набор шаблонов должен быть организован осмысленным с точки зрения контекстов применения образом. Такой набор обычно называют *библиотекой*, или *каталогом шаблонов*, а если набор четко определен и описан, является достаточно полным и покрывает все решения в предметной области, его называют *языком шаблонов* (хотя, учитывая скорость развития во всех тех областях, где существуют цифровые продукты, кажется маловероятным, что подобный язык сумеет стабилизироваться в обозримом будущем).

Шаблоны проектирования *не являются* рецептами или готовыми решениями. В своей книге «Designing Interfaces»[1] – объемистом и полезном собрании шаблонов проектирования взаимодействия – Дженифер Тидвелл (Jenifer Tidwell) предостерегает нас: «[Шаблоны] – это не го-

[1] Дж. Тидвелл «Разработка пользовательских интерфейсов». – Пер. с англ. – СПб: Питер, 2008.

товые к употреблению компоненты; каждая реализация шаблона немного отличается от всех других» (Tidwell, 2006).

Мир проектирования программного обеспечения до некоторой степени заворожен идеей, что исчерпывающий каталог шаблонов позволит даже начинающим проектировщикам быстро и легко собирать целостные решения проектирования (если есть четкое представление о потребностях пользователя). И хотя мы сталкивались с определенными подтверждениями этой идеи, когда речь шла об опытных проектировщиках взаимодействия, чисто механическое использование шаблонов, без понимания контекста их применения, попросту невозможно. Как метко указывает Кристофер Александер, архитектурные шаблоны суть противоположность типовым домам, поскольку контекст на сто процентов определяет форму проявления шаблона в реальном мире. Крайне важно, в какой среде будет использован шаблон, важны другие шаблоны, которые входят в его состав, примыкают к нему или окружают его. То же верно и для шаблонов проектирования взаимодействия. Суть каждого шаблона – в отношениях между представленными объектами, а также между представленными объектами и целями пользователя. (Это одна из причин того, что общее руководство по стилю неспособно заменить ориентированное на контекст проектное решение.) Точная реализация шаблона наверняка будет отличаться в каждом конкретном случае, и определяющие ее объекты, естественно, будут разными в разных предметных областях, однако отношения между объектами останутся по существу такими же.

Типы шаблонов проектирования взаимодействия

Подобно большинству прочих шаблонов проектирования, шаблоны проектирования взаимодействия можно выстроить в иерархию, простирающуюся от концептуального уровня до уровня отдельных элементов интерфейса. Подобно принципам их можно применять на различных уровнях инфраструктуры интерфейса (причем различия между уровнями точно так же могут быть весьма размытыми):

- Шаблоны **позиционирования** могут применяться на концептуальном уровне и помогают определить тип продукта в отношении к пользователю. Пример такого шаблона – *временный тип*, который означает, что использование продукта скоротечно и требуется только в составе деятельности, направленной на достижение более глобальной цели. Понятие позиционирования и наиболее важные шаблоны типов мы подробно опишем в главе 9.
- **Структурные** шаблоны решают проблемы, связанные с управлением отображением информации и функциональных элементов на экране. Они включают виды, панели и группы элементов, вкратце обсуждавшиеся в главе 7.
- **Поведенческие** шаблоны решают широкий спектр проблем, относящихся к конкретным взаимодействиям с теми или иными функцио-

нальными элементами или элементами данных. В эту категорию попадает поведение отдельных компонентов интерфейса, и многие из этих низкоуровневых шаблонов описываются в третьей части книги.

Структурные шаблоны являются, по всей видимости, наименее документированными, однако при этом они распространены повсеместно. Один из наиболее широко применяемых высокоуровневых структурных шаблонов хорошо прослеживается в Microsoft Outlook: навигационная панель слева, обзорная панель справа вверху, панель подробностей справа внизу (рис. 8.1).

Данный шаблон является оптимальным для полноэкранных приложений, в которых пользователю требуется работать с разнообразными объектами, объединять объекты в группы, а также просматривать содержимое или свойства отдельных объектов или документов. Он позволяет выполнять все эти операции в рамках одного экрана, не открывая дополнительных окон. Многие программы для электронной почты используют этот шаблон, а его вариации встречаются в многочисленных программах для создания и управления информацией, где часто требуется быстрый доступ и управление объектами многих типов.

Создание собственного каталога шаблонов – один из важнейших аспектов обучения проектировщика взаимодействия. Изучая лучшие образцы работы других специалистов, мы коллективно совершенствуем идиомы взаимодействия во благо пользователей, а проделанная работа позволяет нам сосредоточить усилия на решении новых проблем, избавляя от необходимости изобретать колесо.

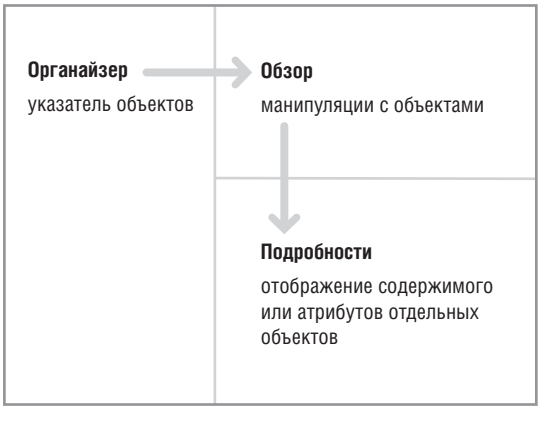


Рис. 8.1. *Основной структурный шаблон Microsoft Outlook часто копируется при разработке программ, причем в самых различных предметных областях. Левая вертикальная панель отвечает за навигацию и управляет содержимым обзорной панели справа вверху. Выбор элемента из обзорной панели приводит к заполнению панели справа внизу подробностями или содержимым документа*

9

Техническая платформа и тип интерфейса

Как вы помните из главы 7, первый вопрос на старте проектирования интерактивного продукта звучит следующим образом: «Какой тип интерфейса и какую техническую платформу следует выбрать?» **Платформой** можно считать сочетание аппаратных и программных средств, позволяющих продукту функционировать – как в плане взаимодействия с пользователем, так и на уровне внутренних механизмов.

Несомненно, вы знакомы с некоторыми распространенными платформами для интерактивных продуктов, включая приложения для настольных компьютеров, веб-сайты и веб-приложения, киоски, автомобильные системы, портативные устройства (фотоаппараты, телефоны, КПК), домашние развлекательные комплексы (игровые консоли, телевизионные тюнеры, музыкальные центры) и профессиональные устройства (медицинские и научные приборы). Глядя на этот список, вы можете отметить, что понятие «платформа» не имеет четкого определения. Это скорее сокращение для описания ряда важных особенностей продукта, таких как физическая форма, размер и разрешение дисплея, способы ввода и подключения к сети, операционная система и возможности для работы с данными.

Все эти факторы существенным образом влияют на способ проектирования, конструирования и использования продукта. Выбор платформы – это поиск баланса между наилучшей поддержкой потребностей и контекста персонажей с одной стороны и ограничениями и задачами бизнеса, а также технологическими возможностями – с другой.

Тип интерфейса определяет поведенческую сущность продукта – то, как он предъявляет себя пользователю. Тип интерфейса – это способ описать то, как много внимания пользователь будет уделять взаимодействию с продуктом и каким образом продукт будет реагировать на это внимание. Как и все прочие проектные решения, выбор типа ин-

терфейса должен опираться на понимание вероятных контекстов и среды применения продукта.

Тип интерфейса

Большинство людей демонстрируют поведение, типичное для их профессии. Солдат насторожен и бдителен, сборщик налогов скучен и безразличен ко всему, актер ярок и заметен в любом обществе, человек из обслуживающего персонала бодр и услужлив. Подобно людям продукты предъявляют себя пользователю в какой-то определенной доминирующей манере.

Программа может быть смелой или скромной, красочной или тусклой, но эти ее качества должны определяться конкретным целеориентированным предназначением. Стиль поведения программы не должен зависеть от персональных предпочтений ее проектировщика или программиста. То, как программа предъявляет себя, формирует отношение пользователя к ней, а это, в свою очередь, сильно влияет на удобство использования программного продукта. Программа, внешнее представление и поведение которой конфликтуют с ее назначением, раздражает и выглядит неуместно, как волосы в чашке с чаем или клоун на свадьбе.

Внешнее представление и поведение продукта должны соответствовать способу его использования, а не личным вкусам проектировщиков. С точки зрения позиционирования программы ее внешнее представление и поведение *не* являются только лишь эстетической категорией – это категория поведенческая. Тип интерфейса вашей программы – часть ее поведенческого фундамента, а все эстетические решения должны находиться в гармонии с ним.

Тип интерфейса определяет многие важные ориентиры для всех остальных проектировочных решений, однако он не позволяет поделить мир на черное и белое. Как человек способен вести себя по-разному в зависимости от контекста, так и некоторые продукты могут демонстрировать признаки различных типов. Читая электронную почту с устройства BlackBerry во время поездки на поезде, пользователь может сосредоточить свое внимание на взаимодействии с устройством (и ожидать сопоставимой по качеству взаимности), но тот же самый человек уже не сможет уделять так много внимания интерфейсу устройства, когда пытается найти адрес, спеша на встречу. Другой пример: текстовый процессор обычно следует оптимизировать для вдумчивого, увлеченного и частого использования, но некоторые встроенные инструменты, например мастер создания таблиц, используются нечасто и помалу. В подобных случаях следует не только определять доминирующий тип интерфейса продукта в целом, но и думать о самостоятельных типах интерфейса для отдельных возможностей продукта и контекстов применения.

Техническая платформа и тип интерфейса тесно связаны: различные аппаратные платформы благоприятствуют приложениям различных

поведенческих типов. Приложение, работающее на мобильном телефоне, очевидно, должно разрабатываться с учетом иной разновидности пользовательского внимания, чем образовательная программа для игровой приставки.

В этой главе мы обсудим уместные типы интерфейса и другие соображения проектирования для ряда платформ – персонального компьютера, Всемирной паутины, киосков, портативных устройств и бытовых устройств.

Проектирование настольных приложений

Термин «настольные приложения» мы используем в качестве обобщающего для программ, работающих на современном персональном компьютере. Вообще говоря, проектирование взаимодействия уходит корнями именно в настольные приложения. Разумеется, ситуации, в которых проектировщикам приходилось бороться с проблемами при реализации сложного поведения, возникали на самых разных технических платформах, однако именно персональные компьютеры принесли это сложное поведение на каждый рабочий стол. Поэтому в основе многих затронутых в данной книге тем лежит появившееся у нас понимание того, что необходимо настольным приложениям, чтобы эффективно служить потребностям человека. В новейшей истории это понимание распространилось на среду Всемирной паутины, на большие устройства и мобильные устройства, а также на прочие системы реального времени, как станет видно далее в этой главе.

Определяя платформу продукта, очевидно, необходимо выйти за пределы термина «рабочий стол», чтобы выбрать подходящую продукту операционную систему, систему управления базами данных, а также технологию пользовательского интерфейса. Оценка каждого из перечисленных аспектов настольных приложений выходит за рамки нашей книги, и все же принимаемые в этой области решения крайне важно анализировать с точки зрения поддержки потребностей пользователей. Более того, поскольку все виды проектирования, по сути, есть диалог с материалом, важно также понимать ограничения и возможности, связанные с каждой из этих ключевых технологий.

К сожалению, решения, связанные с выбором платформы – особенно в том, что касается аппаратной части, – во многих организациях до сих пор принимаются до того, как к работе оказывается привлечен проектировщик. Важно донести до руководства, что выбор платформы будет более эффективным, если его сделать после завершения работы проектировщиков.

ПРИНЦИП проектирования

Решения о выборе технической платформы следует соотносить с работой по проектированию взаимодействия.

Интерфейс настольных приложений можно отнести к одному из трех типов: **монопольный**, **временный** и **фоновый**. Поскольку каждая из категорий имеет свой набор поведенческих атрибутов, категория определяет индивидуальный тип взаимодействия с пользователем. Важно, что эти категории дают проектировщику отправную точку при проектировании интерфейса. Например, программа, являющаяся монопольной, не будет удобной, если не ведет себя соответственно этому статусу.

Монопольный тип

К приложениям **монопольного типа** относятся программы, полностью завладевающие вниманием пользователей на длительные периоды времени. Монопольное приложение предлагает пользователям большой набор тесно связанных функций и возможностей, а пользователи обычно разворачивают такое приложение на весь экран и работают с ним непрерывно. Вот характерные примеры приложений такого типа: текстовые процессоры, электронные таблицы, программы для работы с электронной почтой. Многие приложения для вертикальных рынков также являются монопольными, поскольку нередко остаются на экране в течение долгого времени, а взаимодействие с ними может быть очень сложным и запутанным. Пользователи монопольных программ часто оказываются в состоянии потока. Окно монопольной программы обычно *развернуто на весь экран* (состояния окон мы обсудим в главе 20). Например, трудно представить работу с Microsoft Outlook в окне 7,5×10 сантиметров. При таком размере невозможно использовать эту программу по прямому назначению, т. е. создавать и просматривать сообщения электронной почты и информацию о встречах (рис. 9.1).

Для продуктов с монопольным интерфейсом характерна непрерывная работа в течение длительных отрезков времени. В процессе работы пользователя монопольный продукт является его основным инструментом и преобладает над остальными. Например, приложение PowerPoint занимает полный экран все время, которое пользователь работает над презентацией, от начала и до конца. Даже если в процесс вовлечены другие программы, PowerPoint сохраняет свою монопольную сущность.

Пользователями монопольных приложений, как правило, являются середняки

Люди обычно уделяют много времени и внимания работе с монопольными приложениями и часто кровно заинтересованы в том, чтобы одолеть кривую обучения и стать пользователями-середняками, как говорилось в главе 3. Каждый пользователь проходит стадию новичка, но эта стадия коротка *по сравнению с общим временем, в течение которого он работает* с данным продуктом. Конечно, новичку приходится преодолевать первоначальный подъем кривой обучения, но с точки зрения суммарного времени работы с приложением в будущем время, потраченное на знакомство с программой, невелико.

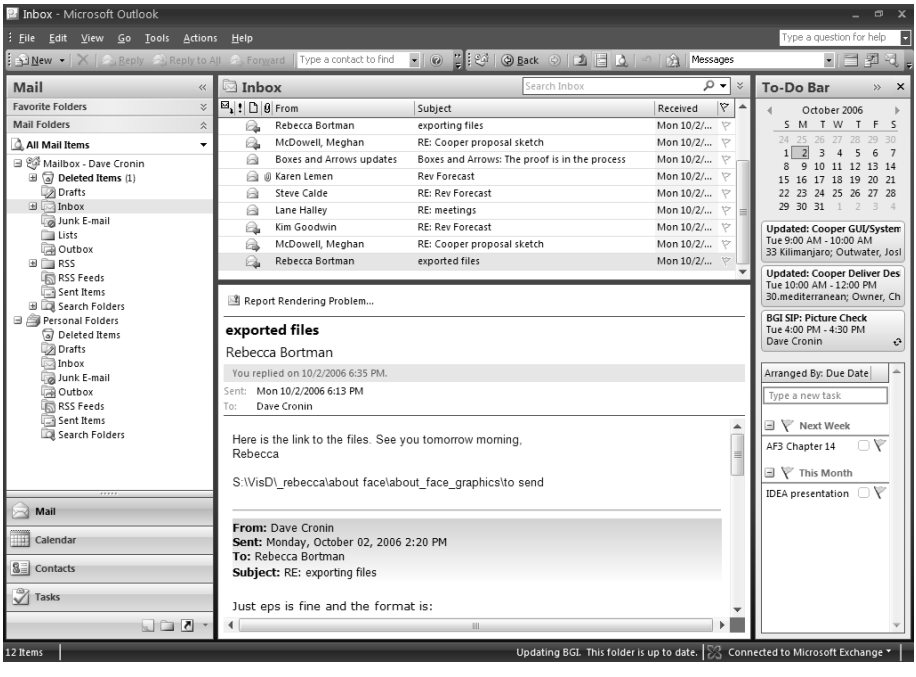


Рис. 9.1. *Microsoft Outlook является классическим примером приложения с монопольным интерфейсом. Эта программа подолгу находится на экране, непрерывно взаимодействуя с пользователем, а ее многочисленные навигационные и информационные панели хорошо себя чувствуют тогда, когда занимают весь экран*

Для проектировщика это часто означает, что программа должна быть нацелена на использование вечными середняками, а не начинающими пользователями и не специалистами. Здесь равно неуместно как жертвовать производительностью и мощностью ради неуклюжей идиомы, облегчающей обучение, так и давать пользователю лишь сложные, хотя и мощные инструменты. Впрочем, если вы найдете более простые идиомы, работающие не в ущерб взаимодействию с пользователем среднего уровня, то это будет оптимальный вариант. В любом случае подготовка пользователя, на которого ориентирована оптимизация, определяется выбором ключевого персонажа и вашим пониманием его взглядов, склонностей и контекстов использования.

Между новичками и середняками существует значительная прослойка пользователей, прибегающих к услугам монопольных приложений лишь время от времени. Их нельзя игнорировать. Конечно, успех приложения-монополиста по-прежнему предопределяют середняки, которые часто его используют, – однако лишь до тех пор, пока не появится конкурирующее приложение, удовлетворяющее *одновременно* и середняков, и новичков. Хороший пример – WordStar, древний текстовый

процессор. Он доминировал на рынке текстовых процессоров в конце 70-х и в начале 80-х годов, поскольку исключительно хорошо отвечал потребностям середняков, хотя новичкам и тем, кто пользовался им от случая к случаю, работать с ним было крайне трудно. Компания WordStar Corporation процветала, пока один из ее конкурентов не предложил программу, столь же мощную и удобную для середняков и одновременно более понятную для менее опытных пользователей. Редактор WordStar оказался неконкурентоспособным и быстро сошел со сцены.

Не жалейте места на экране

Поскольку взаимодействие пользователя с монопольным приложением занимает почти весь сеанс работы за компьютером, программа может смело затребовать все доступное пространство экрана. Никакая другая программа не станет с ней конкурировать, так что не растрачивайте экранное пространство понапрасну, но и не стесняйтесь – берите столько, сколько нужно. Если требуется предоставить пользователю четыре панели инструментов, создавайте четыре. Для программы, позиционируемой в другой категории, это было бы чересчур, но приложение-монополист – в своем праве.

В большинстве случаев окно монопольного приложения развернуто на весь экран. В отсутствие явных инструкций от пользователя ваше самостоятельное приложение должно по умолчанию занимать весь экран. Программа должна позволять пользователю изменять размер окна и оставаться работоспособной при любом его размере, но по умолчанию интерфейс должен быть ориентирован на полный экран, а не на другие, более редкие варианты.

ПРИНЦИП проектирования

Оптимизируйте монопольные приложения для работы на полном экране.

Используйте строгий визуальный стиль

Поскольку пользователь смотрит на монопольное приложение в течение продолжительного времени, позаботьтесь о том, чтобы приглушить цвета и текстуру визуальной части. Придерживайтесь консервативной цветовой палитры. Крупные и яркие элементы управления способны привлечь новичков, но через пару недель ежедневной работы они станут казаться кричащими. Крохотные точки или легкие цветовые акценты в конечном счете будут эффективнее крупных клякс и к тому же позволят вам плотнее упаковать управляющие элементы.

ПРИНЦИП проектирования

Интерфейс монопольного приложения должен придерживаться консервативного визуального стиля.

Пользователь будет подолгу смотреть на одни и те же палитры, меню и панели инструментов и вследствие значительного стажа работы привыкнет к расположению элементов интерфейса. Это дает проектировщику возможность достичь высокой информативности при минимальном «расходе» пикселов. Панели инструментов и соответствующие элементы управления могут иметь уменьшенные размеры. Служебные элементы, например разделители экрана, линейки и полосы прокрутки, можно располагать плотнее и делать более узкими.

Обогащенная обратная связь

Монопольные приложения – отличная платформа создания сред с обогащенной обратной связью для пользователей. Вы можете повышать продуктивность, расширяя интерфейс дополнительными информационными блоками. Строка состояния у нижнего края экрана, концы полос прокрутки, обычно занятые ползунками, строка заголовка и другие пыльные углы видимой области программы могут быть заполнены индикаторами ее состояния, состояния обрабатываемых данных, состояния системы и другими подсказками, повышающими производительность труда пользователя. Однако будьте осмотрительны: обогащая визуальную обратную связь, следите за тем, чтобы не получить в результате безнадежно замусоренный интерфейс.

Новичок не заметит эти артефакты и тем более не поймет их, потому что они не будут бросаться в глаза. Но через некоторое время постоянной работы он станет обращать на них внимание, интересоваться их смыслом и изучать их методом «тыка». В этот момент пользователь захочет приложить определенные усилия, чтобы узнать больше. Если вы предоставите ему простой способ выяснить назначение этих элементов, он не просто станет более опытным пользователем – он станет более довольным пользователем, а его власть над программой будет расти вместе с его уровнем знаний о ней. Добавление таких дополнительных индикаторов к существующему интерфейсу подобно добавлению разнообразных ингредиентов к мясному бульону: блюдо в целом становится лучше. Идея **обогащенной визуальной немодальной обратной связи** подробно обсуждается в главе 25.

Обогащенные средства ввода

Аналогичным образом монопольные приложения выигрывают от обогащенных средств ввода. Для каждого часто используемого аспекта приложения необходимо обеспечить несколько способов управления. Непосредственное манипулирование, диалоговые окна, клавиатурные сокращения – все в этом случае будет уместным. При использовании идиом непосредственного манипулирования вы можете предъявлять более жесткие требования к точности моторики пользователя. Чувствительные области на экране могут иметь размер 2×2 пиксела, потому что вы вправе исходить из предположения, что пользователь

удобно устроился в рабочем кресле, его рука устойчиво расположена на столе и мышь четко перемещается по эластичному коврику.

В монопольных приложениях следует применять обогащенные средства ввода.

Вы можете зайти еще дальше – разместить элементы интерфейса в углах и на границах окна программы. В кабине самолета наиболее часто используемые элементы управления расположены прямо перед пилотом, а те, которыми он пользуется от случая к случаю или в нештатной ситуации, находятся на подлокотниках, над головой или на боковых панелях. В редакторе Word наиболее часто применяемые функции размещены на двух главных панелях инструментов (рис. 9.2). Часто вызываемые, но изменяющие внешний вид экрана функции закреплены за маленькими кнопками слева от горизонтальной полосы прокрутки внизу экрана. Эти кнопки полностью меняют представление документа: Обычный, Разметка страницы, Структура. Новички используют

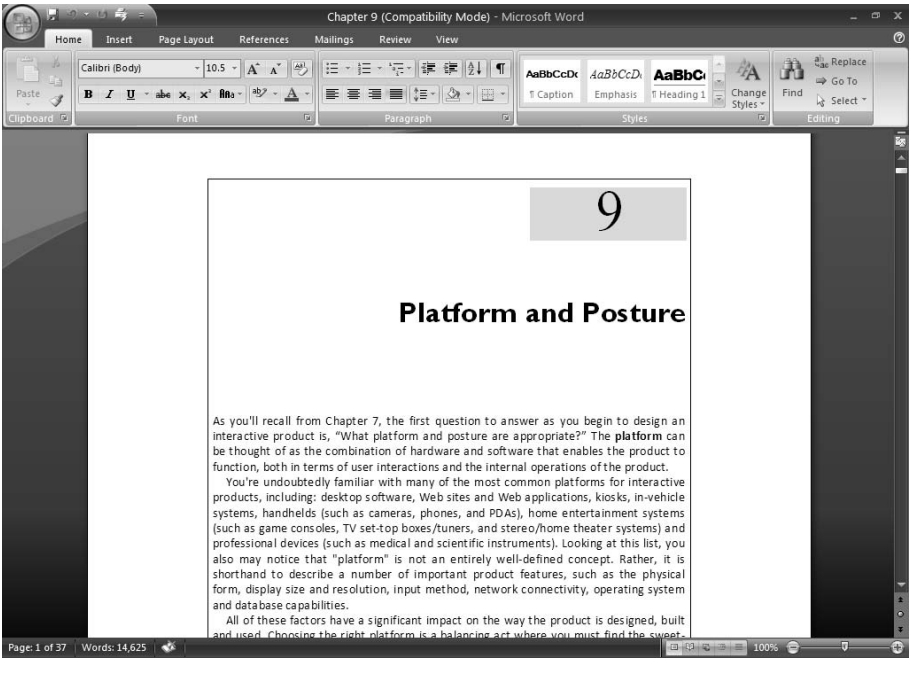


Рис. 9.2. *В редакторе Microsoft Word элементы управления размещены в верхней и нижней части окна. Элементы, расположенные внизу, обособлены потому, что вызывают значительное изменение внешнего представления документа*

их редко, а случайный вызов этих функций может смутить неопытного пользователя. Будучи размещенными внизу экрана, эти кнопки становятся практически невидимыми для новичков. Их обособление тонко и безмолвно намекает, что пользоваться ими следует с осторожностью. Более опытные и уверенные в своих действиях пользователи заметят эти элементы интерфейса и зададутся вопросом об их назначении. Они могут поэкспериментировать с этими элементами, когда будут морально готовы к последствиям. Это очень точная и полезная увязка размещения элементов с их использованием.

Приложения, ориентированные на работу с документами

Аксиома о том, что монопольное приложение должно работать в полный экран, верна и для окон документов, открытых приложением. Дочерние окна с документами всегда должны быть развернуты, если пользователь явным образом не потребовал иного или ему не понадобилось для выполнения определенной задачи увидеть одновременно несколько документов.

Разворачивайте документы в монопольных приложениях на полный экран.

Многие монопольные приложения ориентированы на работу с документами (их основное назначение состоит в создании и просмотре документов, содержащих разнообразные данные). В результате легко начать считать, что окно приложения – это и есть окно документа. Однако это не всегда так. Если приложение работает с документом, но выполняет лишь одну простую функцию, скажем сканирует изображение, оно не является монопольным, и ему не следует монополизировать внимание пользователя. Такие «приложения одной функции» имеют собственный тип – временный.

Временный тип

Продукт **временного** типа приходит и уходит, предлагая одну функцию и ограниченный набор связанных с этой функцией элементов управления. Приложение вызывается при необходимости, делает свою работу и быстро исчезает, позволяя пользователю продолжить прерванную деятельность (как правило, в окне монопольного приложения).

Определяющей характеристикой временного приложения является его преходящая сущность. Поскольку оно не находится на экране в течение больших периодов времени, у пользователя нет возможности привыкнуть к нему. Следовательно, интерфейс программы должен быть недвусмысленным и представлять элементы управления четко и ясно, исключая ошибки или путаницу. Интерфейс должен сообщать о своих функциях. Здесь нет места красивым, но неоднозначным пиктограммам или изображениям. Как раз *здесь* кнопки должны быть больши-

ми, а надписи на них – ясными, набранными крупным и хорошо читаемым шрифтом.

Временные приложения должны быть простыми, понятными, четкими.

Хотя временная программа, несомненно, может быть единственной запущенной программой на рабочем столе, она, как правило, играет роль вспомогательной при монопольном приложении. Типичным примером сценария работы с временным приложением является вызов Проводника для поиска и открытия файла в то время, когда вы уже редактируете другой файл в редакторе Word. Другой пример – регулировка громкости динамиков компьютера. Поскольку временная программа отбирает место на экране у монопольной, она должна проявлять уважение и не требовать пространства больше, чем необходимо. Если монопольное приложение имеет право вырыть котлован и заложить фундамент, то временное просто ставит палатку на выходные. Оно не может занять все пространство на экране и все время пользователя. Это автомобиль-такси в мире программного обеспечения.

Если вся компьютерная система в целом играет роль временного приложения в физическом мире атомов, не обязательно минимизировать число потребляемых пикселов и объем привлекаемого внимания. Таким свойством обладают, к примеру, мониторы слежения в производственных условиях и цифровые видеосистемы в операционной. Здесь весь экран компьютера используется лишь временами, тогда как пользователь монопольно занят механической деятельностью. В подобных случаях крайне важно, чтобы информация была внятной и легко воспринималась с расстояния в несколько метров, а это, очевидно, требует более смелого применения цвета и более щедрого распределения экранного пространства (рис. 9.3).

Рис. 9.3. *Хорошие примеры временных приложений – Yahoo! Widgets и iTunes. Обращение к ним и взаимодействие с ними происходит в течение краткого промежутка времени, после чего внимание пользователя возвращается к деятельности, связанной с приложением-монополистом. Применение насыщенной объемной визуализации придает этим приложениям достаточную визуальную привлекательность*

Яркие и понятные

Хотя временная программа должна экономно использовать экранное пространство, ее элементы управления могут быть пропорционально больше, чем элементы управления монопольного приложения. Перегруженный графикой интерфейс монопольного приложения приестся пользователю через пару недель, но временная программа находится на экране не так долго, чтобы надоесть пользователю. Наоборот, броская графика поможет пользователю быстрее сориентироваться, когда программа появится на экране.

Инструкции по работе с временными программами должны быть встроены в их интерфейс. Весьма вероятно, что пользователь видит окно такой программы лишь раз в месяц, так что вполне способен забыть смысл ее элементов управления. Так, вместо надписи «Настройка» на кнопке была бы уместна надпись «Настройка пользовательских предпочтений», а сама кнопка, соответственно, должна быть крупнее. Конструкция «действие – объект» делает интерфейс более понятным, а результаты использования кнопки – более предсказуемыми. По аналогичным причинам во временной программе недопустимы сокращения. Обратная связь должна быть конкретной и ясной, чтобы избежать путаницы. Скажем, пользователь должен без труда понимать, что принтер занят, а длительность только что записанного аудиоролика составляет пять секунд.

Стремитесь к простоте

После того как пользователь вызвал временную программу, вся информация и все инструменты должны быть предоставлены ему непосредственно в единственном окне этой программы. Удерживайте внимание пользователя на этом окне и не распыляйте основную функцию программы по служебным и диалоговым окнам. Если вы ловите себя на том, что создаете диалоговое окно или второе представление для временной программы, это верный признак того, что ваше решение требует пересмотра.

Временное приложение следует ограничивать одним окном и одним представлением.

Во временных программах неуместны узкие полосы прокрутки и возня с мышью. Требования к точности моторики пользователя должны быть минимальными. Лучшее решение – простые кнопки для простых функций. Непосредственное манипулирование тоже может быть эффективным решением, но объекты манипулирования должны быть достаточно крупными, чтобы было легко находить их и взаимодействовать с ними. Клавиатурные сокращения также желательны, но они должны быть простыми, а все важные функции должны быть напрямую доступны в интерфейсе.

Конечно, для временных приложений возможны исключения из правила «одна функция», но они редки. Если приложение выполняет более одной функции, интерфейс должен отражать это визуально и недвусмысленно, предоставляя мгновенный доступ ко всем функциям, но не обрастая при этом дополнительными окнами или диалогами. Одно из таких приложений – Art Directors Toolkit, созданный Code Line Communications. Оно выполняет ряд вычислительных функций, востребованных пользователями приложений для графического дизайна (рис. 9.4)

Помните, что временная программа, скорее всего, будет вызвана для поддержки некоторых аспектов монопольного приложения (как в случае с Art Directors Toolkit на рис. 9.4). Это означает, что временное приложение, расположившись на экране поверх монопольного приложения, может заслонить ту самую информацию, ради которой было вызвано. Отсюда следует, что окно временной программы должно быть перемещаемым, то есть иметь заголовок или иной очевидный способ перетаскивания.

Очень важно, чтобы управление временным приложением требовало минимальных накладных расходов. Пользователь хочет лишь выполнить конкретную функцию и двигаться дальше. Было бы крайне неразумно заставлять его совершать при этом какие-то непродуктивные действия, связанные с управлением окном приложения.

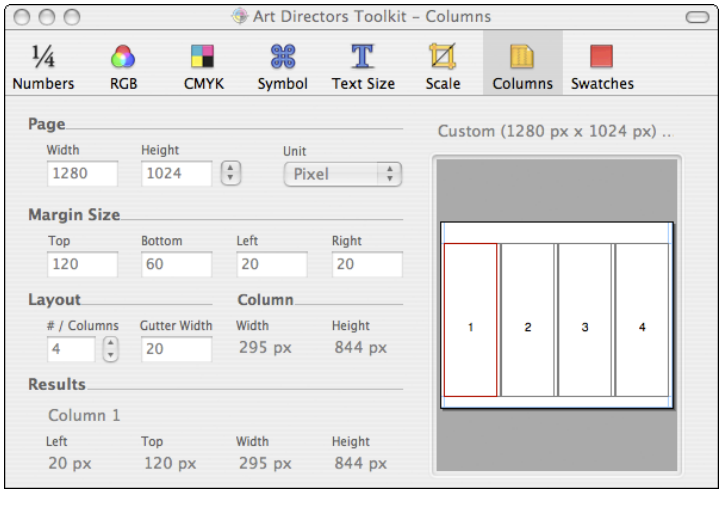


Рис. 9.4. *Art Directors Toolkit от Code Line Communications – еще один пример временного приложения. Данная программа предоставляет пользователю ряд самостоятельных функций, таких как вычисление специальных размеров для сетки верстки. Эти функции спроектированы с целью поддержки монопольных приложений верстки, вроде Adobe InDesign. Имея несколько различных функций, это приложение структурирует их посредством вкладок, так что все функции доступны в любой момент*

Сохранение пользовательского выбора

Наиболее уместный способ помочь пользователю как с временными, так и с монопольными приложениями – наделить приложение собственной памятью. Если временная программа запомнит свое состояние при последнем использовании, велика вероятность, что и в следующий раз положение и размеры ее окна окажутся подходящими. Практически всегда такое решение лучше любых значений по умолчанию. Какие бы форму и положение ни придал пользователь программе, она должна появляться именно в таком виде при следующем вызове. Конечно, то же самое справедливо и в отношении ее логических настроек.

Временное приложение должно восстанавливать предыдущее положение и предыдущую конфигурацию.

Читатели уже наверняка поняли, что практически все диалоговые окна по сути своей являются временными приложениями. Нетрудно заметить, что все предшествующие рекомендации, относящиеся к временным приложениям, в равной степени применимы и к проектированию диалоговых окон (диалоговые окна подробно обсуждаются в главах 24 и 25).

Фоновый тип

Программы, которые в нормальном состоянии не взаимодействуют с пользователем, позиционируются как **фоновые**. Они работают в фоновом режиме, невидимые и неслышные, и выполняют задачи, которые, возможно, важны, но не требуют вмешательства пользователя. Драйвер принтера или подключение к сети – вот два отличных примера.

Как можно догадаться, любое обсуждение интерфейса фоновой программы будет по естественным причинам кратким. Тогда как временное приложение управляет выполнением функции, фоновые приложения обычно управляют процессами. Сердцебиение – это не функция, которая требует сознательного контроля, но процесс, автономно происходящий в фоновом режиме. Подобно процессам, регулирующим сердцебиение, фоновые приложения остаются обычно совершенно незаметными, добросовестно выполняя свое предназначение, пока включен компьютер. Однако, в отличие от сердца, их требуется время от времени устанавливать и удалять, а также настраивать в связи с изменениями обстоятельств. Именно в такие моменты возникает необходимость в общении фоновых приложений с пользователем. Взаимодействие между пользователем и фоновой программой является по природе своей исключительно временным, так что здесь действуют все правила для временных приложений.

Следование принципам проектирования временных приложений, а именно обеспечение информирования пользователей о назначении приложения и его возможностях, а также информирование о смысле

выбранных значений, в ситуации с фоновыми приложениями становится еще более критичным. Во многих случаях пользователь и не подозревает о существовании фоновой программы. С учетом этого факта становится очевидно, что сообщения от таких программ могут сбить пользователя с толку, не будучи предъявленными в соответствующем контексте. Поскольку многие из этих программ выполняют таинственные функции (как, например, драйвер принтера или концентратор соединений), поступающие от них сообщения должны быть такими, чтобы пользователи не испытывали растерянности или недоумения.

Вопрос, ответ на который считается очевидным, когда речь идет о приложениях других типов, становится принципиальным для фоновых программ: если программа невидима, как вызвать на экран ее окно в тех редких случаях, когда в нем возникает необходимость? Один из наиболее распространенных подходов в системе Windows – представить фоновую программу пиктограммой в системной области уведомлений.

Размещение перед глазами пользователя пиктограммы, которой он, может быть, никогда не воспользуется, является оскорблением не меньшим, чем наклеивание рекламы на ветровое стекло автомобиля. Пиктограммы фоновых программ постоянно должны быть перед глазами, только если предоставляют полезную информацию о состоянии этих программ. Microsoft решила эту проблему в Windows XP: пиктограммы фоновых программ скрываются, если пользователь не обращается к ним достаточно активно (рис. 9.5).

Эффективным подходом к настройке фоновых программ, применяемым как в Mac OS, так и в Windows, являются **панели управления**. Это программы временного типа, обеспечивающие единую точку входа для настройки служб. Важно также обеспечить прямой непротиворечивый доступ к фоновым приложениям в любой момент, когда возникает проблема, мешающая пользователю решать свои задачи. (Разумеется, здесь тоже действует стандартное правило: не прерывайте работу пользователя без необходимости.) Например, если пиктограмма в области уведомлений указывает на проблему с принтером, щелчок по этой пиктограмме должен давать доступ к механизму для исправления ситуации.

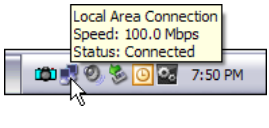


Рис. 9.5. *Область уведомлений на панели задач в Windows XP. Указатель мыши наведен на пиктограмму, представляющую фоновый процесс, который отслеживает подключение к сети. Эта пиктограмма немодально отображает статус и, если доступа к сети нет, меняет свой внешний вид. Если навести указатель мыши на пиктограмму, можно получить дополнительную информацию, а щелчок правой кнопкой мыши дает доступ к различным функциям, связанным с сетевым подключением*

Проектирование в среде Всемирной паутины

Развитие Всемирной паутины стало для проектировщиков взаимодействия одновременно благом и проклятием. Возможно, впервые со времени изобретения графических пользовательских интерфейсов люди, принимающие решения, начали понимать подход к проектированию, ориентированный на пользователя, и способствовать его использованию. С другой стороны, ограничения и сложности реализации взаимодействия в веб-среде – естественный результат ее развития – отбросили проектировщиков на несколько лет в прошлое. Проектировщики веб-приложений только сейчас начинают пользоваться преимуществами многих привычных идиом взаимодействия (таких как перетаскивание), бывших в ходу за годы до появления первых веб-сайтов.

На заре бума Всемирной паутины в отрасль пришло множество выпускников школ графического дизайна, традиционных дизайнеров и молодых энтузиастов, которые видели в веб-среде захватывающие и сулящие выгоду возможности для коммуникации посредством новых форм интерактивного визуального взаимодействия. Наибольшие сложности возникали с обходом жестких ограничений формата (изначально созданного для организации совместного доступа к научным работам и сопутствующим иллюстративным материалам), даже когда требовался лишь базовый уровень интерактивности и визуального структурирования.

Уже в те времена авторы веб-сайтов признавали, что источником новой проблемы проектирования являются гиперссылки в документах – проектирование, организация и структурирование содержания. Предложенный Питером Морвилем (Peter Morville) термин видимость[1] (findability) метко описывает эту проблему. Новое поколение проектировщиков, назвавшее себя *информационными архитекторами*, создало дисциплину и практику решения проблем невизуального проектирования, касающихся логической структуры и связей внутри информационного наполнения.

Некоторые из современных приложений, работающих в броузере (часто называемых приложениями *Web 2.0* – термин, предложенный Тимом О'Рейли (Tim O'Reilly)), стирают различия между настольными приложениями и веб-приложениями и даже предлагают возможность создавать новые идиомы взаимодействия, наилучшим образом поддерживающие тех людей, для которых мы проектируем. С появлением так называемых насыщенных интернет-приложений (использующих такие технологии, как AJAX, Macromedia Flash, Java и ActiveX) проектирование интернет-приложений стало требовать гораздо большего внимания к тонким аспектам *поведения* продукта, чем в случае с преж-

[1] Этому вопросу посвящена книга Питера Морвиля «Тотальная видимость. Как наши находки меняют нас». – Пер. с англ. – СПб: Символ-Плюс, 2008. – *Примеч. науч. ред.*

ними простыми веб-сайтами. И хотя видимость остается важным вопросом, ее могут затмить классические проблемы взаимодействия, известные по настольным приложениям.

Появившаяся возможность создавать сложное поведение в броузере предъявляет к качеству проектирования взаимодействия такие же требования, как разработка самостоятельных программных приложений. Одного только внимания к внешнему виду сайта со стороны дизайнера и внимания к структуре со стороны информационного архитектора теперь уже недостаточно для того, чтобы на новом витке развития Всемирной паутины создавать эффективный и привлекательный для пользователей опыт взаимодействия. Прежде чем заняться различными типами поведения приложений в веб-среде, мы обсудим виды продуктов и услуг, предлагаемых обычно через веб-броузер: информационные веб-сайты, сервисные веб-сайты и веб-приложения. Границы между продуктами и услугами различных видов, конечно же, могут быть размытыми. Считайте, что они представляют шкалу, на которой можно найти любой веб-сайт или веб-приложение.

Информационные веб-сайты

Изначально веб-броузеры задумывались как средство просмотра опубликованных и связанных документов без обращения к неудобным протоколам вроде FTP (File Transfer Protocol), Gopher или Archie. Как следствие, изначально веб-среду составляли исключительно коллекции документов (или *страниц*) такого рода, известные как **веб-сайты.** Мы по-прежнему используем этот термин для описания информационных служб Всемирной паутины, взаимодействие с которыми сводится к поиску информации и переходу по гиперссылкам. Такие веб-сайты легко придумать: набор страниц или документов, имеющих последовательную или ступенчатую иерархию, модель навигации для перехода с одной страницы на другую, а также функция поиска, обеспечивающая целеориентированный доступ к конкретным страницам. Существует множество простых веб-сайтов – персональных, созданных для нужд маркетинга и технической поддержки, а также информационных для интрасетей. В случае таких сайтов основные вопросы проектирования связаны с дизайном, визуальной композицией, элементами навигации и структурой (информационной архитектурой). Веб-сайты, как правило, не демонстрируют сложного поведения (то есть такого поведения, где результат взаимодействия с пользователем зависит от состояния приложения), а потому им редко требуется внимание проектировщиков взаимодействия.

Поскольку предмет этой книги – проектирование взаимодействия, мы не станем обсуждать многочисленные аспекты проектирования веб-сайтов, которые подробно описаны в имеющейся литературе. Ясное и доступное изложение важных базовых элементов сайтостроительства дается, в частности, в книгах «The Art and Science of Web Design»

(Veen, 2000) Джеффри Вина (Jeffrey Veen), «Don't Make Me Think!»[1] (Krug, 2000) Стива Круга (Steve Krug) и «Information Architecture»[2] (Rosenfeld and Morville, 1998) Луиса Розенфельда (Louis Rosenfeld) и Питера Морвиля. Еще один отличный источник информации – веб-сайт Якоба Нильсена *useit.com*.

Типы информационных веб-сайтов

Чисто информационные сайты, которые не требуют сложных транзакций для реализации постраничной навигации и ограниченного поиска, должны балансировать между необходимостью вывода важной информации с достаточной плотностью и необходимостью предоставления новичку и нечастому гостю возможностей быстро учиться и ориентироваться на сайте. Отсюда проистекает конфликт между монопольными и временными признаками информационных сайтов. То, какой тип перевесит, сильно зависит от целевых персонажей сайта и от шаблонов их поведения. Приходят ли эти пользователи изредка или однократно – или же это постоянные посетители, заглядывающие еженедельно или ежедневно, чтобы просмотреть содержимое сайта?

Частота обновления содержимого сайта до некоторой степени предопределяет это поведение. Информационные сайты с постоянно обновляемой информацией, естественно, привлекут больше постоянных посетителей, чем сайты, где содержимое обновляется раз в месяц. Редко обновляемые сайты используются скорее как источники справочного материала (если информация на них не является животрепещущей), чем как ресурсы, к которым пользователь обращается постоянно. Следовательно, они должны демонстрировать поведение, типичное для временного, а не монопольного приложения. Более того, такой сайт может подстраиваться под пользователя в зависимости от частоты его посещений, проявляя признаки монопольного типа.

Признаки монопольного типа

Подробное представление информации легче реализовать в монопольном режиме. Предполагая, что пользователь развернет окно на весь экран, проектировщики могут воспользоваться преимуществами доступного пространства, чтобы ясно представить информацию, а также средства навигации и подсказки, по которым смогут ориентироваться пользователи.

Единственной проблемой при позиционировании веб-сайта как монопольного приложения является выбор оптимального разрешения экрана. (В меньшей степени та же проблема преследует настольные при-

---

[1] С. Круг «Веб-дизайн: книга Стива Круга или не заставляйте меня думать!», 2-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2008.

[2] Л. Розенфельд, П. Морвиль «Информационная архитектура в Интернете», 2-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2005.

ложения, хотя создателям таких приложений легче диктовать используемое разрешение.) Разработчикам веб-сайта приходится уже на ранних стадиях проекта принимать решение о том, какое наименьшее разрешение они готовы поддерживать. Технически возможно использовать адаптивную («резиновую») верстку для гибкого отображения содержания страниц при самых разнообразных размерах окон, однако все равно необходима оптимизация под распространенные экранные размеры и наименьшее допустимое для ключевого персонажа разрешение. Качественные исследования помогут с прояснением этого вопроса: какое количество людей, похожих на персонажей сайта, работает с разрешением 800×600?

Признаки временного типа

Чем реже ключевые персонажи посещают сайт, тем более сильным должен быть его уклон в сторону временного типа. Для информационного сайта это означает простоту и прозрачность навигации и ориентирования.

Сайты, используемые от случая к случаю как справочные ресурсы, должны позволять пользователю ставить закладки на любой странице, чтобы он смог быстро вернуться на то же место впоследствии.

Если материал сайта обновляется с периодичностью от одного раза в неделю до одного раза в месяц, пользователи, вероятнее всего, будут посещать его нерегулярно, и, следовательно, система навигации должна быть предельно ясной. Если сайт сохраняет информацию о действиях пользователя с помощью cookie-файлов или серверных методов и предоставляет информацию с учетом интересов пользователя, это окажет серьезную навигационную поддержку тем, кто посещает сайт относительно редко (здесь мы исходим из предположения, что при каждом последующем посещении пользователя интересует похожая информация).

Сервисные веб-сайты

Некоторые веб-сайты выходят за рамки обычных сайтов и предлагают сервисные функции, позволяющие посетителям не только получать информацию. Классические примеры сервисных веб-сайтов – интернет-магазины и сайты финансовых служб. Как правило, они структурируются аналогично информационным веб-сайтам, но помимо информации на страницах размещены еще и функциональные элементы, обладающие более сложным поведением. В интернет-магазине эта группа элементов представлена корзиной, функциями обработки заказа и возможностью сохранять профиль пользователя. На сайтах некоторых магазинов имеются и более сложные интерактивные инструменты, такие как конфигураторы, позволяющие пользователям выбирать и уточнять параметры своих приобретений.

Чтобы реализовать адекватное поведение функционально обогащенных элементов управления, в процессе проектирования сервисных

веб-сайтов необходимо уделять пристальное внимание как архитектурной организации страниц, так и проектированию взаимодействия. Разумеется, обе задачи, равно как и эффективная передача атрибутов бренда (а это бывает часто важно, учитывая коммерческую природу большинства сервисных сайтов), нуждаются в поддержке со стороны графического дизайна.

Типы сервисных веб-сайтов

Подобно информационным веб-сайтам интернет-магазины, интернет-банки, инвестиционные сайты, порталы и прочие сайты должны удерживать равновесие между монопольной и временной линиями поведения. И действительно, многие сервисные сайты содержат значительные объемы информации – к примеру, онлайн-покупатели любят изучать и сравнивать продукты. В рамках этой деятельности пользователи часто уделяют значительное внимание одному сайту, но в некоторых случаях (как при выборе наилучшего предложения) они просматривают несколько сайтов подряд. Для таких сайтов крайне важна простота навигации, а равно наличие доступа к сопутствующей информации и эффективность транзакций.

Поисковые машины и порталы вроде Google и Yahoo! представляют собой особую разновидность сервисного сайта; их назначение – обеспечивать навигацию по другим веб-сайтам, а также доступ к подборкам новостей и информации из многочисленных источников. Очевидно, что поиск и переход на сайты из результатов поиска – деятельность, носящая временный характер, однако для агрегации информации на портале вроде Yahoo! при реализации иногда требуется использовать монопольный подход. Мимолетность общения пользователей с временными функциями сервисных сайтов делает особенно важной минимизацию действий, связанных с навигацией. Возникает соблазн разбить информацию и функции на несколько страниц, чтобы снизить время загрузки и визуальную сложность (похвальное стремление), однако следует избегать путаницы и помнить, что аудитория устает щелкать мышью. В важном юзабилити-исследовании 2001 года, проведенном User Interface Engineering и посвященном тому, как пользователи воспринимают время загрузки страниц сайтов электронной коммерции, таких как Amazon.com и REI.com, выяснилось, что *восприятие пользователем продолжительности загрузки страницы в большей степени зависит от того, достигает ли пользователь своих целей, нежели от реальной продолжительности загрузки* (Perfetti and Landesman, 2001).

Веб-приложения

Веб-приложения насыщены взаимодействием и демонстрируют сложное поведение – они во многом похожи на серьезные настольные приложения. И хотя некоторые веб-приложения сохраняют постраничную модель навигации, их страницы скорее похожи на *представления*, чем на веб-документы. Хотя многие подобные приложения по-

прежнему ограничены архаичной моделью запрос/ответ (которая требует, чтобы пользователь вручную «отправлял» каждое изменение состояния), существующая технология уже поддерживает асинхронный обмен с сервером и локальное кэширование данных, что позволяет приложению в броузере вести себя во многом подобно сетевому настольному приложению.

Вот некоторые примеры веб-приложений:

- Программы для предприятий, начиная со старомодных интерфейсов SAP, реализованных в броузере, и заканчивая современными средствами командной работы, такими как Salesforce.com и Basecamp компании 37signals.
- Средства персональных публикаций, включая приложения для блогов (MoveableType от SixApart), приложения для публикации фотографий (Flickr) и, конечно же, вездесущие wiki-системы.
- Рабочие инструменты, такие как работающий в броузере текстовый процессор Writely и электронные таблицы Google Spreadsheets.

Такие веб-приложения во многом могут предъявляться пользователям как настольные приложения – просто работающие внутри броузера. Возникающие при этом шероховатости не будут значительными, если взаимодействия тщательно спроектированы с учетом технологических ограничений. И хотя задача проектирования и создания насыщенного и оперативного взаимодействия, работающего в различных броузерах, определенно может быть сложной, Всемирная паутина как платформа замечательно подходит для создания инструментов, способствующих и оказывающих поддержку совместной работе. Помимо прочего веб-броузер является еще и хорошим средством предоставления нечасто используемой функциональности, поскольку не требует установки исполняемых файлов на компьютер пользователя. И, разумеется, веб-приложение дает пользователям возможность работать со своей информацией и нужными функциями в любом месте, где есть доступ к сети Интернет. Это не всегда уместно или необходимо, однако, учитывая современную мобильность сотрудников компаний и растущую популярность удаленной работы, такая возможность может представлять собой определенную ценность, позволяя людям получать доступ к одним и тем же инструментам и функциям с различных компьютеров.

Существует несколько распространенных мифов о веб-приложениях, и об этих мифах стоит упомянуть. Во-первых, создавать такие приложения не проще и не быстрее. По нашему опыту создание приложения, работающего в броузере, требует такого же или даже большего объема планирования, разработки и программирования, как в случае применения стандартных технологий для персональных компьютеров. Во-вторых, веб-приложения не являются по умолчанию более удобными или более понятными. Принято считать, что раз большое число пользователей знакомо с некоторыми веб-взаимодействиями (в силу своего опыта взаимодействия с интернет-магазинами и инфор-

мационными веб-сайтами), эти пользователи смогут применить свои навыки и быстрее разобраться с приложением, работающим в броузере. И хотя в этом утверждении может быть небольшая доля истины, факты свидетельствуют, что проектирование удовлетворительного опыта взаимодействия для любой платформы требует серьезного анализа и тяжелой работы. Подача приложения через броузер определенно не позволяет обойтись без этого.

Типы веб-приложений

Веб-приложения, как и настольные приложения, могут быть монопольными или временными, однако поскольку веб-приложениями мы называем продукты со сложной и богатой функциональностью, то они по определению тяготеют к монопольному типу.

Монопольные веб-приложения стремятся доставлять информацию и функциональность так, чтобы наилучшим образом поддерживать сложную деятельность человека. Часто это требует создания функционально насыщенных высокоинтерактивных пользовательских интерфейсов. Хорошим примером подобного веб-приложения служит Flickr – интернет-служба публикации фотографий, которая предоставляет такие возможности, как сортировка изображений посредством перетаскивания (drag-and-drop) и прямое редактирование текстовых меток и аннотаций (рис. 9.6). Многочисленные корпоративные программы, работающие в броузере, также являются монопольными веб-приложениями.

К проектированию монопольных веб-приложений лучше подходить так же, как к проектированию настольных приложений, забыв об информационных и сервисных веб-сайтах с постраничной навигацией. Проектировщикам также требуется четкое понимание технических ограничений среды и пределов возможного в рамках имеющихся у разработчика времени и бюджета. Подобно монопольным настольным приложениям большинство монопольных веб-приложений должны быть полноэкранными, плотно заполненными информационными и функциональными элементами и должны использовать специализированные панели или другие экранные области для группировки родственных функций и объектов. У пользователей должно возникать ощущение среды, а не постоянных переходов от страницы к странице или с места на место. Следует минимизировать видимую перерисовку экрана (на обычных веб-сайтах практически каждое действие требует полного обновления страницы).

Преимущество позиционирования монопольных веб-приложений как настольных приложений, а не наборов веб-страниц, состоит в том, что проектировщики получают возможность вырваться из страничной модели взаимодействия в броузере и работать со сложными вариантами поведения, столь необходимыми веб-приложениям. Веб-сайты – хороший способ получать необходимую информацию, они подобны лифтам, удобно доставляющим людей на нужные этажи. Но вы же не пытаетесь работать в лифтах? Так и пользователям не будет удобно вы-

Рис. 9.6. *Инструмент Organize на сайте Flickr позволяет пользователям создавать наборы фотографий и менять свойства фотографий пакетно – и все в одном месте, не открывая для этого бесчисленные веб-страницы*

полнять в броузере реальную работу, требующую транзакций и интенсивного взаимодействия, если используется страничная модель.

Одно из преимуществ подачи важной для корпораций функциональности с помощью броузера заключается в том, что в случае грамотной реализации такой подход позволяет пользователям в любой момент работать с редко требуемой информацией и функциональностью, не прибегая к установке на компьютер всех инструментов, которые только могут понадобиться. Рутинная задача, выполняемая лишь раз в месяц, или внезапно возникшая необходимость сгенерировать отчет – именно такая работа хороша для **временных веб-приложений**.

Проектируя временные веб-приложения, как и любые другие временные приложения, крайне важно обеспечить прозрачную навигацию и ориентирование. Помните также, что временное приложение одного пользователя может быть монопольным приложением другого. Проанализируйте как следует, насколько совместимы наборы потребностей этих двух пользователей – часто бывает, что корпоративное веб-приложение обслуживает широкий спектр персонажей, так что для доступа к одной и той же информации требуется целый ряд пользовательских интерфейсов.

Интернет-приложения

В результате непрерывного развития сети Интернет родились две удивительных особенности: мгновенный доступ к невероятным объемам информации и простота организации совместной работы. В основе этих особенностей лежит среда World Wide Web, но это не означает, что для их использования требуется веб-броузер.

Другой отличный подход – отказаться от броузера полностью и создавать *интернет-приложения*. Разработка приложения на стандартной для персонального компьютера платформе, такой как .NET или Java/Swing, с использованием стандартных протоколов сети Интернет обеспечивает богатые, четкие, сложные взаимодействия, сохраняя возможность обращаться к данным Всемирной паутины. Развитие протоколов доставки данных, таких как RSS, и интерфейсов прикладного программирования веб-приложений позволяет продуктам извлекать из веб-среды всю ту же информацию, которая доступна броузеру, но быть гораздо более приятными для пользователей благодаря возможностям, доступным только «родным» для персонального компьютера приложениям.

Хороший пример такого приложения – Apple iTunes. Программа позволяет покупать и загружать музыку и видео, получать информацию о компакт-дисках, делать музыку доступной через сеть Интернет – и все это посредством пользовательского интерфейса, оптимизированного для подобных действий способом, который вряд ли возможно реализовать в веб-броузере.

Другой пример такого подхода – системы архивации и передачи изображений, используемые радиологами для просмотра снимков пациентов, в частности снимков, полученных магнитно-резонансным сканированием. Такие системы позволяют радиологам быстро работать с сотнями изображений, масштабировать конкретные снимки и выполнять коррекцию снимков для более точной идентификации различных типов тканей. Очевидно, такого рода взаимодействие плохо подходит для реализации в броузере. При этом возможность просматривать изображения из любого места может оказаться очень полезной для радиолога, если он, скажем, находясь в крупном исследовательском медицинском центре, консультирует деревенскую больницу, где нет специалистов, способных диагностировать определенные заболевания. Поэтому во многих системах архивации изображений применяются интернет-протоколы, обеспечивающие возможности удаленного просмотра и совместной работы.

Интранет-приложения

Интранет (и родственное явление – экстранет) обычно представляет собой помесь веб-сайта и веб-приложения. **Интранет** – это закрытый вариант веб-среды, доступный только сотрудникам компании (а также

ее партнерам, клиентам и поставщикам – в случае экстранета) и обычно содержащий значительное количество информационных страниц, посвященных компании, ее отделениям и деятельности отделений, а также более функциональные составляющие – от модулей учета рабочего времени и подготовки к командировкам до систем поддержки снабжения и бюджетирования. Проектирование информационной части интранета требует, чтобы информационные архитекторы создали качественную организующую структуру, а проектирование функциональной части требует, чтобы проектировщики взаимодействия спроектировали важные варианты поведения.

Прочие платформы

Программа, работающая на персональном компьютере, располагает роскошью получать внимание пользователей, когда ей это необходимо. Проектирование взаимодействия для продуктов, работающих в мобильных и публичных контекстах, требует особого внимания, так как окружающая эти продукты обстановка реального мира полна событий и шумов. **Портативные устройства, киоски и прочие встроенные системы** (в телевизорах, микроволновых печах, автомобильных приборных панелях, фотоаппаратах, банкоматах, лабораторном оборудовании) – уникальные платформы, каждая из которых имеет свои возможности и ограничения. Бездумно наделяя устройства и приборы цифровыми мозгами, мы рискуем создать вещи, которые будут вести себя скорее как компьютеры, чем как удобные продукты, которые нравятся пользователям.

Общие принципы проектирования

Хотя встроенные системы (физические устройства с интегрированными программными системами) и могут предлагать типичное для программ взаимодействие с пользователем, они обладают уникальными особенностями, которые отличают их от систем, работающих в настольном компьютере. При проектировании любой встроенной системы, будь то интеллектуальный бытовой прибор, киоск или портативное устройство, не забывайте следующие основополагающие принципы:

- Не думайте о продукте как о компьютере
- Объединяйте проектирование аппаратной и программной частей
- Позвольте контексту определять направление проектирования
- Используя режимы, делайте это взвешенно
- Ограничивайте функциональность
- Выдерживайте баланс между навигацией и плотностью отображаемой информации
- Минимизируйте сложность ввода

Рассмотрим каждый из принципов подробнее.

Не думайте о продукте как о компьютере

Пожалуй, самое важное правило для разработчика встроенных систем заключается в том, что он создает *не* компьютерную систему, хотя в ее интерфейсе, возможно, преобладает дисплей, аналогичный компьютерному. У пользователей могут быть вполне конкретные ожидания относительно возможностей продукта (если это бытовой прибор или иное известное им устройство) либо, наоборот, не слишком большие ожидания (если это киоск в общественном месте). В такой ситуации последним делом будет переносить весь профессиональный багаж (идиомы и терминологию) из мира персональных компьютеров на «простенькое» устройство вроде фотоаппарата или микроволновки. Аналогично этому пользователи научного или иного технического оборудования ожидают получить быстрый и непосредственный доступ к информационным и функциональным элементам тем способом, который им привычен, не продираясь в поисках необходимого сквозь дебри операционной системы или файловой системы.

Программисты (особенно те, кто имеет опыт проектирования для настольных платформ) легко забывают, что, хотя продукт, который они разрабатывают, является программным продуктом, он предназначен не для компьютера, типичными характеристиками которого являются большой цветной экран, высокая производительность, необъятная память, полноценная клавиатура и мышь. У большинства встроенных систем ничего подобного нет. И, что еще важнее, эти продукты используются в совершенно иных контекстах, чем настольные компьютеры.

Идиомы, ставшие привычными на персональных компьютерах, совершенно неуместны для встроенных систем. «Отмена» – неподходящее название кнопки, выключающей микроволновку, а требовать от людей входа в режим «настройки», чтобы изменить температуру нагрева, попросту абсурдно. Чем втискивать компьютерный интерфейс в форм-фактор устройства с маленьким экраном, лучше вникнуть в то, для чего создано устройство и как цифровая технология может быть приложена к нему, чтобы облегчить пользователям обращение с этим устройством.

Объединяйте проектирование аппаратной и программной частей

С точки зрения взаимодействия с пользователем отличительной характеристикой встроенных систем является тесное переплетение аппаратных и программных составляющих интерфейса. В отличие от настольных компьютеров, где все внимание пользователя сосредоточено на большом цветном экране с высоким разрешением, большинство встроенных систем предлагают аппаратные элементы управления, на которые направлено внимание пользователя и которые должны точно соответствовать его целям. Стоимость производства, производительность и форм-фактор накладывают ограничения, в силу которых аппаратные элементы управления и навигации часто используются вместо экран-

ных эквивалентов. Следовательно, они должны быть подстроены под требования программной составляющей интерфейса и в то же время соответствовать требованиям эргономики и целям пользователя.

Таким образом, разрабатывать программные и аппаратные компоненты интерфейса системы (и взаимодействие между ними) следует одновременно и с учетом эргономики и целей пользователя, а также соображений эстетического плана. Многие из лучших, передовых цифровых устройств, существующих сегодня (например, TiVo, iPod), проектировались именно в соответствии с таким целостным подходом – их аппаратная и программная части сочетаются настолько хорошо, что опыт, получаемый пользователями, оказывается привлекательным и эффективным (рис. 9.7). Это редко происходит в ходе стандартного процесса разработки, когда команда конструкторов аппаратной части периодически передает команде разработчиков программной части законченные механические решения и решения промышленного дизайна, готовые к производству, а уж разработчики программ должны подстраиваться под эти решения без учета интересов пользователей.

Рис. 9.7. *Дизайн компании Cooper для интеллектуального телефона, демонстрирующий высокую степень интеграции аппаратных и программных элементов управления. Пользователь может легко настраивать громкость динамика, набирать телефонные номера, управлять воспроизведением голосовых сообщений с помощью аппаратных элементов управления, управлять записной книжкой и входящими звонками, вызывать журнал, голосовую почту и устраивать конференции с помощью сенсорного экрана и управляющего колесика. Интерфейс данного аппарата не перегружен функциями, но упрощает доступ к наиболее востребованным и самым важным из них. Обратите внимание на то, что размеры областей сенсорного экрана, к которым нужно прикасаться, соответствуют размеру пальца, а текстовые подсказки облегчают обращение с аппаратом*

Позвольте контексту указывать направление проектирования

Еще одним принципиальным отличием встроенных систем от настольных приложений является значение, которое имеет для них контекст среды. Хотя вопросы контекста иногда возникают и при разработке настольных приложений, проектировщики могут предполагать, что большинство настольных приложений будут выполняться на стационарном компьютере, расположенном в относительно тихом и уединенном месте. По мере того как портативные компьютеры догоняют по мощности и коммуникационным возможностям настольные системы, это предположение все чаще нарушается, но в целом остается справедливым, поскольку даже с ноутбуком человек старается найти для работы спокойное и малолюдное место.

В отношении многих встроенных систем верно ровно обратное. Эти системы разрабатываются либо для работы на ходу (в портативных устройствах), либо для стационарных применений в общественных местах (в информационных киосках). Даже самые стационарные и уединенные встроенные системы (домашние электроприборы) имеют ярко выраженный контекст. Хозяйка, расставляющая тарелки с горячей едой, пока гости ждут в другой комнате, постоянно отвлекается от элементов управления «интеллектуальной» микроволновки. Навигационные системы в приборных панелях автомобилей не должны иметь программируемых кнопок, изменяющих свое назначение в различных контекстах, поскольку водителю придется отрывать взгляд от дороги, чтобы прочитать подпись. От работника в цехе нельзя требовать, чтобы все его внимание было сосредоточено на расшифровке элементов управления оборудования. В некоторых случаях отвлечение внимания от основной деятельности может представлять угрозу для жизни.

Таким образом, проектирование встроенных систем должно очень четко соответствовать контексту их использования. У портативных устройств этот контекст определяется тем, как и где (физически) пользователь работает с устройством. Как он его держит? Одной рукой или двумя? Где оно находится, когда не используется? Чем еще занимается пользователь во время работы с устройством? В каком окружении оно используется? Там шумно? Светло или темно? Что чувствует пользователь этого устройства, когда работает с ним на глазах множества людей? Некоторые из этих вопросов мы более подробно рассмотрим далее.

Для киосков вопрос контекста сводится к окружению, в котором он находится, и к некоторым социальным моментам. Какую роль играет киоск в данном окружении? Находится ли он на пути людского потока? Является ли он источником справочной информации или же представляет интерес сам по себе? Способна ли архитектура окружения естественным образом направить людей к киоску, когда это требуется? Можно ли оценить, сколько людей будут пользоваться киоском одновременно? Достаточно ли киосков, чтобы удовлетворить запросы поль-

зователей, не создавая длинных очередей? Достаточно ли свободного пространства вокруг киосков, не мешают ли они людскому потоку? Эти и другие вопросы мы также обсудим чуть позже.

Используя режимы, делайте это взвешенно

Как правило, приложения в настольных компьютерах имеют множество рабочих режимов. Программа может находиться в разных состояниях, в которых элементы управления и ввода данных могут иметь различные линии поведения. Хорошей иллюстрацией сказанного являются палитры инструментов в Photoshop: выберите инструмент – и ваши манипуляции с мышью и клавиатурой будут связаны с набором функций выбранного инструмента. Выберите другой инструмент – и реакция устройств ввода на аналогичные манипуляции будет иной.

К сожалению, пользователей очень легко сбить с толку с помощью недостаточно очевидного режима поведения. Поскольку самостоятельные устройства обладают обычно дисплеями меньшего размера и ограниченными механизмами ввода, очень сложно сообщить пользователю, в каком режиме находится продукт, а для смены режима часто требуются сложные навигационные действия. Возьмем для примера мобильные телефоны. Обычно режимы организованы иерархически, и для доступа к конкретному режиму приходится нажимать огромное количество кнопок. Большинство владельцев сотовых телефонов используют только функцию набора номера и телефонную книгу, и многие из них сразу теряются, когда возникает необходимость в доступе к другим функциям. Даже такая важная функция, как выключение звонка, часто выходит за рамки умений среднего пользователя телефона.

При проектировании встроенных систем важно ограничивать число режимов, а переключение между ними в идеальном случае делать естественным следствием изменений контекста. Например, в смартфоне переключение в режим телефона должно происходить, когда поступает звонок, а по окончании разговора устройство должно возвращаться в предыдущий режим. (Ответ на звонок при работе с другими данными – предпочтительная альтернатива.) Если работа в нескольких режимах действительно необходима, они должны быть недвусмысленно представлены в интерфейсе, а путь выхода из каждого режима должен быть столь же ясен. Четыре аппаратных кнопки в большинстве наладонных компьютеров под управлением Palm OS – хороший пример четкого обозначения режимов (рис. 9.8).

Ограничивайте функциональность

Большинство встроенных систем используются в конкретных ситуациях с конкретными целями. Боритесь с искушением превращать их в универсальные компьютеры. Устройства с ограниченным набором функций обслужат пользователей лучше, чем устройства, пытающиеся совместить много несопоставимых функций в одном корпусе. Такие устройства, как КПК под управлением Microsoft Windows Mobile, в по-

Рис. 9.8. *Palm Treo 650 предлагает аппаратные кнопки для переключения между режимами (заметьте, что кнопки календаря и электронной почты расположены по обе стороны от джойстика)*

следнее время претендующие на эмуляцию полноценных настольных систем, рискуют отпугнуть пользователей громоздким интерфейсом, который под завязку набит функциями, включенными в него только потому, что они существуют в настольных системах. Многие из нас полагаются на свои «умные» телефоны (такие как Treo, BlackBerry), но думаю, большинство согласится, что спектр функций этих устройств до некоторой степени снижает их пригодность как телефонов.

Многие устройства способны обмениваться информацией с компьютерами. Имеет смысл подходить к проектированию таких систем, ориентируясь на персональный компьютер: устройство является *спутником* настольного компьютера, его расширением и предоставляет пользователю важнейшие данные и функции в таких контекстах, где использование настольного компьютера невозможно. Сценарии помогут вам определить, какие именно функции действительно необходимы в системах-спутниках.

Выдерживайте баланс между навигацией и плотностью отображаемой информации

Многие самостоятельные устройства имеют ограниченное экранное пространство. Это является следствием стоимости аппаратных составляющих, форм-фактора, соображений мобильности или потребления электроэнергии, но, каковы бы ни были причины этого, проектировщикам следует оптимально использовать доступную технологию вывода информации для удовлетворения потребностей пользователей. При создании встроенных систем с ограниченным экранным пространством ценен каждый пиксел, каждый сегмент и каждый квадратный

миллиметр экрана. Такие ограничения по площади дисплея почти всегда требуют компромисса между ясностью информации и сложностью навигации. Разумно ограничивая набор функций, вы можете до определенной степени смягчить ситуацию, но конфликт между выводом информации и навигацией существует почти всегда.

Вы должны тщательно продумывать вывод на дисплей во встроенных системах, создавая иерархическую информационную структуру. Определите, какая информация наиболее важна для пользователя, и сделайте соответствующую функцию самой заметной в интерфейсе. Затем выясните, какая дополнительная информация еще может поместиться на экране. Постарайтесь избежать переключений между различными группами данных, при которых экран мигает. Например, микроволновая печь с цифровым управлением может переключаться между температурой, до которой следует выполнять нагрев, и текущей температурой. В этом случае очень легко перепутать, где какая температура. Более удачным решением будет вывод конечной и текущей температур при помощи столбцовой диаграммы, показывающей, насколько текущая температура близка к желаемой. На дисплее должно еще оставаться место для отображения состояния соответствующих аппаратных элементов управления, а еще лучше будет, если эти элементы сами будут демонстрировать свое состояние. Для этой цели подходят кнопки со встроенными светодиодами, двух- и многополюсные выключатели, ползунковые регуляторы, рукоятки.

Минимизируйте сложность ввода

Почти все встроенные системы оборудованы упрощенным механизмом ввода, а отнюдь не клавиатурой и устройством графического ввода. Это означает, что ввод любой (особенно текстовой) информации в систему неудобен, выполняется медленно и труден для пользователей. Даже самые изощренные из механизмов ввода – сенсорные экраны, устройства голосового ввода и распознавания рукописного текста, встроенные клавиатуры – неудобны по сравнению с полноразмерной клавиатурой и мышью. Таким образом, ввод должен быть максимально ограничен и упрощен.

Устройство BlackBerry от компании RIM эффективно задействует колесико в качестве основного механизма выбора. Быстрая прокрутка колесика перебирает варианты, а нажатие на него (или на соседнюю кнопку) приводит к выбору текущего пункта. В устройстве используется и встроенная клавиатура для ввода текстовых данных. В Palm Treo, для сравнения, используется сенсорный экран и встроенная клавиатура. Такой подход был бы эффективен, если бы экран Treo позволял активизировать любую функцию прикосновением пальца. Однако большинство элементов управления на экране Palm столь малы, что приходится использовать перо. Это означает, что пользователю приходится переключаться между пером и сенсорным планшетом, и в результате ввод становится неудобным. В поздних устройствах компа-

нии Palm проблема решается расположенным между сенсорным экраном и встроенной клавиатурой «джойстиком» с двумя осями и кнопкой, что позволяет выполнять навигацию и активировать элементы управления на экране, не прикасаясь к экрану (см. рис. 9.8).

В информационных киосках экраны обычно крупнее, но и там тоже следует по возможности отказываться от текстового ввода. Сенсорные экраны, если позволяет размер, могут отображать виртуальные клавиатуры, но каждая клавиша должна быть достаточно велика, чтобы пользователю было сложно опечататься. В сенсорных экранах, кроме того, следует избегать идиом, связанных с перетаскиванием. Простые идиомы прикосновения к объекту легче поддаются контролю и более очевидны для новичков (при наличии ярко выраженного назначения).

Проектирование портативных устройств

Портативные устройства ставят перед проектировщиками взаимодействия особые задачи. Поскольку эти устройства созданы для мобильного использования, они должны быть маленькими, легкими, экономичными с точки зрения энергопотребления, прочными и удобными для манипулирования в ситуациях с отвлекающими факторами. Тесное сотрудничество между проектировщиками взаимодействия, промдизайнерами, программистами и конструкторами аппаратной части в случае портативных устройств особенно важно. Практическое значение имеют размер дисплея, читаемость информации, простота ввода и управления, чувствительность к контексту. В данном разделе подробно обсуждаются перечисленные вопросы и предлагаются подходы к их решению. Ниже перечислены самые полезные принципы проектирования взаимодействия и интерфейсов портативных устройств:

- **Старайтесь интегрировать функциональность так, чтобы навигация была минимальной.** Портативные устройства используются в разнообразных и вполне конкретных контекстах. Исследуя контекстные сценарии, вы получите хорошее представление о том, какие функции необходимы для обеспечения естественного и ориентированного на цели опыта пользователя.

  Большинство подобных устройств перегружено функциями, а это риск оставить недовольной большую часть пользовательской аудитории. Коммуникаторы, вроде Treo, проявляют себя лучше всего, когда основная функциональность направлена на улучшение качества коммуникации. Эти устройства неплохо объединяют телефон с записной книжкой: когда поступает звонок, можно видеть полное имя звонящего, взятое из записной книжки, а одним прикосновением к имени в записной книжке можно набрать соответствующий номер. Однако эта интеграция может стать еще более совершенной. Щелчок по имени в записной книжке может отображать все документы, связанные с указанным человеком и известные устройству: встречи, сообщения электронной почты, телефонные звонки, записи с именем

этого человека, интернет-адреса и т. д. Аналогичным образом щелчок по адресу электронной почты в записной книжке может открывать приложение электронной почты. Некоторые новые приложения для коммуникаторов, такие как Agendus от iambic Inc., начинают использовать подобный подход, интегрируя разрозненные приложения в единый процесс, соответствующий целям пользователей.

- **Думайте о том, как пользователь будет носить и держать устройство.** Чтобы понять, как пользователь будет манипулировать устройством, необходимо создавать физические модели. Эти модели должны, как минимум, отражать размер, форму и трансформации (например, раскладной корпус) устройства. Если модель отражает и вес устройства, она становится еще эффективнее. Проектировщики должны использовать эти модели в контекстных и ключевых сценариях для проверки пригодности выбранных форм-факторов. Подписи к кнопкам очень сильно зависят от контекстов применения. К примеру, кнопкам на устройстве, отслеживающем пересылку срочной почты, подсветка не нужна, в отличие от кнопок мобильного телефона или пульта дистанционного управления телевизором.
- **Уже на ранней стадии проектирования определите, будет ли пользователь работать с устройством одной рукой или обеими.** С помощью сценариев следует выяснить, какие режимы приемлемы для пользователей в различных контекстах. Если устройство разработано преимущественно для операций, выполняемых одной рукой, наличие расширенных функций, требующих манипуляций двумя руками, приемлемо только в том случае, если эти функции используются нечасто. Например, если устройство для инвентаризации позволяет выполнять подсчет одной рукой, но затем требует использования двух рук для пересылки введенных данных, оно спроектировано неудачно, поскольку операция отправки данных является частью стандартного сценария.
- **Выясните, является устройство спутником или самостоятельным прибором.** Большинство портативных устройств лучше всего проектировать как спутники стационарных систем обработки данных. Устройства Palm и Windows Mobile преуспели на рынке как портативные системы, способные к взаимодействию с настольными компьютерами или серверами. Их создатели не пытались скопировать все функции настольной системы, а сосредоточились на доступе к информации и просмотре данных, обеспечив лишь минимальную функциональность средств ввода и редактирования. Последние портативные модели расширяют идею устройства-спутника, привязанного к компьютеру, переходом в сферу беспроводной связи, в результате чего идея таких устройств развивается и становится более практичной. С другой стороны, некоторые устройства (например, обычные сотовые телефоны) разрабатываются как действительно самостоятельные приборы. Многие сотовые телефоны позволяют загружать телефонные номера из компьютера, но большинство их

владельцев игнорируют эту возможность из-за сложности процедуры. Такие самостоятельные устройства оказываются наиболее успешными, когда фокусируются на узком наборе функций, но выполняют их на высочайшем уровне.

- **Избегайте всплывающих и смежных (расположенных бок о бок) окон.** На маленьких дисплеях с низким разрешением для всплывающих окон обычно нет места. Поэтому интерфейсы портативных устройств должны напоминать интерфейсы монопольных приложений в том смысле, что им следует использовать все экранное пространство. Любой ценой избегайте немодальных диалоговых окон, а от модальных окон и сообщений об ошибках избавляйтесь с помощью приемов, обсуждаемых в главах 24 и 25.

Типы интерфейсов портативных устройств

Проектирование для портативных устройств – это упражнение на обход аппаратных ограничений: механизм ввода, размер и разрешение экрана, потребление энергии – вот лишь несколько примеров таких ограничений. Одно из самых важных прозрений для многих проектировщиков в отношении ряда портативных устройств состоит в том, что они часто не являются самостоятельными системами. Портативные устройства – это спутники стационарных систем (как в примере с устройствами-органайзерами Palm и Windows Mobile) и больше используются для просмотра информации, чем для ввода. (И хотя у многих портативных устройств есть встроенные клавиатуры или подключаемые раскладные клавиатуры, они все же малопригодны для взаимодействия, требующего постоянного ввода данных.) Эти устройства-спутники обычно имеют временный тип интерфейса. При типичном взаимодействии пользователь быстро сверяется со своим расписанием на день или добавляет запись в список дел, пока горит красный сигнал светофора.

Интересной разновидностью портативных устройств являются сотовые телефоны. Они *не* устройства-спутники, а основное средство связи. Однако с точки зрения типа интерфейса телефоны также являются временными приложениями. Пользователь начинает звонок мгновенно, после чего сразу забывает об интерфейсе в пользу разговора. Можно сказать, что лучший интерфейс для телефона – невизуальный. Голосовая активация идеально подходит для совершения звонков, а открывание флипа – вероятно, самый удобный способ на них отвечать (можно еще освободить руки, отвечая на звонки при помощи той же самой голосовой активации). Чем меньше внимания требует интерфейс телефона, тем лучше.

Нам часто приходится выслушивать идеи создания монопольных приложений для портативных устройств – скажем, продукта, позволяющего радиологам просматривать медицинские снимки (рентгеновские или с магниторезонансного сканера) на поле для гольфа. Очевидно, что технологиям портативных дисплеев предстоит еще проделать дол-

гий путь, прежде чем такие продукты можно будет выпускать, и пока неясно, насколько хорошо портативные устройства подходят для монопольных приложений (мы в данном случае не рассматриваем развлекательные продукты вроде мобильных игровых устройств и DVD-проигрывателей).

Проектирование киосков

На первый взгляд может показаться, что у киосков много общего с интерфейсом настольных компьютеров: большой цветной экран и довольно мощный процессор. Однако когда речь заходит о взаимодействии с пользователем, сходство заканчивается. В отличие от пользователей монопольного настольного приложения пользователи киоска в лучшем случае имеют с ним дело нечасто, а в типичном – обращаются к нему лишь однажды. Более того, подходя к киоску, человек имеет либо одну конкретную цель, либо вовсе никакой определенной цели. Пользователи киосков обычно не имеют в своем распоряжении клавиатуры и мыши, а если бы и имели, то в большинстве своем не смогли бы применить их эффективно. Наконец, пользователи киосков, как правило, находятся в шумном общественном месте в окружении множества отвлекающих факторов, к тому же они могут воспользоваться киоском целой компанией. Все эти факторы окружения накладывают отпечаток на проектирование пользовательского интерфейса киоска (рис. 9.9).

Рис. 9.9. *GettyGuide – система информационных киосков, спроектированная компанией Cooper совместно с Getty и Triplecode*

Сервисные и исследовательские киоски

Киоски разделяются на две категории – **сервисные** и **исследовательские**. Сервисные киоски предоставляют строго ограниченную транзакцию или услугу. Сюда входят банкоматы, автоматы по продаже билетов в аэропортах, на железнодорожных и автобусных вокзалах, а также в некоторых кинотеатрах. Даже автозаправку или кофе-автомат можно рассматривать как простейший сервисный киоск. Пользователи сервисных киосков имеют вполне конкретные цели: получить наличные, билет, пачку жевательной резинки или какую-то конкретную информацию. Единственное, что их интересует, – по возможности быстро и без проблем достичь своих целей.

Исследовательские же киоски обычно стоят в музеях. Образовательные и развлекательные киоски не находятся в центре внимания посетителей, но предоставляют дополнительную информацию и новые возможности для тех, кто пришел посмотреть экспозицию (хотя есть несколько музеев , таких как, скажем, музей Experience Music Project в Сиэтле, где интерактивные киоски являются основой некоторых экспонатов). Исследовательские киоски отличаются от сервисных тем, что пользователь не ожидает от них чего-либо конкретного. Пользователем может двигать любопытство, или смутное желание развлечься, или стремление повысить свой культурный уровень, но у него, скорее всего, нет определенной конечной цели. (С другой стороны, он может захотеть найти кафе или уборную – и эти цели можно поддерживать наряду с открытыми целями.) В исследовательских киосках привлекать пользователя должен сам акт исследования. Следовательно, интерфейс киоска должен быть не только четким и снабженным легко осваиваемой навигацией, но и эстетически привлекательным, то есть приятным на вид (и, возможно, на слух). Каждое окно должно представлять интерес само по себе, а также стимулировать пользователя к дальнейшему исследованию системы.

Взаимодействие в общественном месте

Сервисные киоски, как правило, не нуждаются в специальных приманках для пользователей. Однако их размещение следует оптимизировать так, чтобы они были заметными и при этом справлялись с наплывом пользователей. Для повышения эффективности киосков используйте специальные внешние указатели. Некоторые сервисные киоски, особенно банкоматы, требуют, чтобы вы учитывали вопросы безопасности. Если местоположение киоска кажется небезопасным (или является таковым), пользователи будут избегать его (или идти на риск). Архитектурное планирование сервисного киоска должно начинаться одновременно с проектированием интерфейса и разработкой конструкции.

Аналогично сервисным киоскам, исследовательские киоски должны быть расположены продуманно и снабжены указателями. Они не должны заслонять главные объекты окружения, но при этом их следует размещать поблизости, чтобы подчеркнуть связь с этими объекта-

ми. Вокруг киоска должно быть достаточно места, поскольку им часто пользуются целые группы людей (например, семья). Особенно трудной задачей является выбор оптимального количества киосков, для чего может потребоваться изучение потока перемещения людей в том месте, где планируется установка. У сервисного киоска люди не задерживаются подолгу и готовы стоять в очереди, поскольку у них есть конкретная цель. Исследовательские киоски, наоборот, располагают пользователя к долгому общению, что делает их непривлекательными для зевак. Потенциальные пользователи не знают, чего ожидать от такого киоска, и у них нет стимулов ждать в очереди. Можно смело предположить, что большинство людей подойдет к исследовательскому киоску, только когда он свободен.

При разработке интерфейса киоска тщательно продумайте звуковое оформление. Для исследовательских киосков естественно использование интенсивной звуковой обратной связи и богатого звукового содержимого, однако уровень громкости не должен отвлекать окружающих от основных объектов, для которых киоск служит лишь дополнением. В сервисных киосках не следует злоупотреблять звуковой обратной связью, однако она может быть полезной, например, для напоминания пользователю, что он должен вынуть банковскую карточку или взять сдачу.

Кроме того, раз киоски располагаются в общественных местах, при проектировании следует учитывать потребности пользователей с физическими недостатками. Более подробно о проектировании доступного взаимодействия мы поговорим в главе 26.

Управление вводом

У многих киосков есть сенсорные экраны или наборы кнопок, соответствующих объектам на экране. В случае киоска с сенсорным экраном действуют те же принципы, что и для сенсорных экранов вообще:

- **Активные экранные объекты должны быть достаточно крупными**, чтобы в них было легко попадать. Они должны быть контрастными, цветными, и их следует разнести так, чтобы избежать случайного прикосновения не к тому объекту. Размер в 20 миллиметров – типичный минимум для пользователей, которые не торопятся и стоят близко к экрану; в тех случаях, когда пользователь торопится и работает вытянутой рукой, объекты следует увеличить. Хорошая быстрая проверка того, что размеры подобраны правильно, выполняется так: распечатайте экран в натуральный размер, испачкайте палец штемпельной краской и пройдитесь по сценариям с разумной скоростью. Если отпечатки ваших пальцев выходят за пределы объектов, размеры объектов, скорее всего, следует увеличить.
- **Избегайте использования виртуальных клавиатур**. У разработчика может возникнуть соблазн создать на сенсорном экране киоска виртуальную клавиатуру для ввода данных. Однако прибегать к тако-

му механизму ввода следует лишь при необходимости организовать ввод коротких текстов. Виртуальная клавиатура не только неудобна для пользователя, но и приводит обычно к заляпыванию экрана отпечатками пальцев.

- **Избегайте перетаскивания**. Овладеть перетаскиванием на сенсорном экране бывает крайне сложно, так что эта идиома не подходит для тех пользователей, которые не тратят много времени на освоение сложных идиом взаимодействия. Прокрутку любого рода тоже следует исключить, кроме тех случаев, когда она абсолютно необходима.

В некоторых киосках вместо дорогих и капризных сенсорных экранов используются аппаратные кнопки, ассоциированные с экранными функциями. Как и в портативных системах, здесь очень важна непротиворечивость этих связей. Аналогичные функции в разных окнах должны быть связаны с одними и теми же кнопками. Кнопки должны быть размещены вблизи экрана и так организованы пространственно, чтобы ассоциирование их с функциями не вызывало сомнений (вопросы ассоциирования более подробно обсуждаются в главе 10). В общем случае при наличии возможности использовать сенсорный экран следует отдавать ему предпочтение перед вариантом с аппаратными кнопками, которые ассоциированы с экранными функциями.

Типы интерфейсов киосков

Из-за крупных дисплеев с полноэкранными приложениями может показаться, что киоски тяготеют к монопольному типу, однако по ряду причин задача так просто не решается. Во-первых, пользователи киосков часто новички (есть ряд исключений, в частности, пользователи банкоматов и автоматов по продаже билетов на общественный транспорт) и в большинстве случаев пользуются киосками нерегулярно. Во-вторых, большинство людей не задерживаются у киоска. Человек выполняет простую операцию или поисковый запрос, получает нужную информацию – и идет дальше. В-третьих, в большинстве киосков применяются сенсорные экраны или шлифованные кнопки рядом с экраном, и оба механизма ввода плохо сочетаются с высокой плотностью данных, которой можно ожидать от монопольного приложения. Наконец, пользователи киосков редко могут удобно сесть перед оптимально расположенным монитором – обычно они стоят в общественном месте, где есть яркое рассеянное освещение и множество отвлекающих факторов.

Такое поведение пользователей и ограничения среды приводят к тому, что большинство киосков как приложения тяготеют к временному типу, имеющему простую навигацию, крупные и яркие, привлекательные интерфейсы с ясным ожидаемым назначением элементов управления и четкой ассоциацией программных функций с аппаратными кнопками (если таковые имеются). Как и при проектировании портативных устройств, следует избегать всплывающих окон и диалогов.

Связанную с ними информацию лучше отображать на одном и том же экране (как принято в монопольных приложениях). Таким образом, киоски балансируют на загадочной границе между двумя наиболее популярными типами настольными приложений.

Поскольку сервисные киоски часто проводят пользователей через последовательность экранов для получения информации, контекстная ориентация и навигация становятся важнее глобальных средств навигации. Помогайте пользователям разобраться не в том, в какой *точке системы* они находятся, но в том, на какой *стадии процесса* они пребывают. Для сервисных киосков важно наличие «аварийных выходов», позволяющих пользователям в любой момент прервать транзакцию и начать сначала.

Киоски следует оптимизировать в расчете на пользователей, не имеющих опыта работы с ними.

Образовательные и **развлекательные киоски** несколько отличаются от **сервисных**, которые тяготеют к приложениям временного типа. В данном случае важнее изучение окружения, в котором расположен киоск, нежели выполнение транзакций или поисковых запросов, а более высокая плотность данных и более сложное взаимодействие и визуальные переходы могут иногда давать положительный эффект. Однако следует внимательно относиться к ограничениям механизмов ввода, иначе пользователь потеряет удобство навигации по интерфейсу.

Проектирование телевизионных интерфейсов

Телевизионные интерфейсы, такие как TiVo и большинство кабельных и спутниковых тюнеров, организуют взаимодействие с пользователем посредством пульта дистанционного управления. Пользователь с пультом обычно находится в другом конце комнаты. Если в пульте не используется радиоканал (а в большинстве пультов применяется однонаправленная передача инфракрасного луча), пользователю приходится направлять пульт в сторону телевизора или тюнера. Все это накладывает определенные ограничения на проектирование эффективного управления телесистемами.

- **Используйте визуальную композицию и дизайн, которые легко читаются с расстояния в несколько метров.** Даже если вы полагаетесь на разрешение экранов HDTV (телевидения высокого разрешения), пользователи не будут сидеть так же близко к телевизору, как к монитору компьютера. Это означает, что текст и прочие элементы навигации следует делать крупными, а визуальное укрупнение, в свою очередь, диктует организацию информации на экране.
- **Навигация по экранам должна быть простой.** Человек не думает о телевизоре как о компьютере, а механизмы навигации, основанные на пультах дистанционного управления, ограничены, поэтому

наилучшим будет такой подход, который легко ассоциируется с контроллером типа «джойстик» (перемещение вдоль двух осей и кнопка). Можно использовать для навигации новаторские решения, вроде колеса прокрутки и других механизмов ввода, однако есть вероятность, что потребуется не только работать с проектируемым устройством, но и поддерживать совместимость с другими дополнительными телевизионными устройствами, поэтому будьте осмотрительны, применяя новые идеи. Кроме этого, простота использования обеспечивается способами организации визуальной навигации, такими как раскраска экранов соответственно функциональным областям и применение визуальных и текстовых подсказок по доступным на каждом экране командам (в этом особенно преуспело устройство TiVo).

- **Не забывайте о необходимости интеграции.** Большинство людей крайне недовольны тем фактом, что для управления домашними развлекательными устройствами, подключенными к телевизору, требуется несколько пультов. Позволяя управлять наиболее востребованными функциями других домашних развлекательных устройств (в идеале – с минимальными трудозатратами на настройку), а не только проектируемого, вы удовлетворите животрепещущую потребность пользователей. Это означает, что пульт управления вашим продуктом или его консоль будут вынуждены передавать команды другим устройствам и, возможно, запоминать некоторые параметры состояния этих устройств. Универсальные пульты дистанционного управления Harmony от Logitech делают и то, и другое, а настройка пультов выполняется через веб-приложение после подключения к компьютеру через порт USB.
- **Дистанционное управление должно быть предельно простым.** Многие пользователи пугаются сложных пультов, так что бóльшая часть функций в пультах домашних развлекательных комплексов попросту не используется. Особенно ярко это проявляется в универсальных пультах, где наблюдается тенденция к перегрузке кнопками – не редкость универсальные пульты с 40, 50 и даже 60 кнопками. Решить эту проблему можно при помощи дисплея на пульте, создающего контекст для кнопок, и тогда кнопок для каждого устройства потребуется гораздо меньше. Реализовать это решение можно посредством сенсорного экрана или многофункциональных кнопок, примыкающих к экрану. У каждого из этих подходов есть недостатки. Так, большинство сенсорных экранов не обеспечивают тактильной обратной связи, поэтому пользователь вынужден отвлекаться от телевизора, чтобы активировать функцию. Многофункциональные кнопки такого недостатка лишены, но увеличивают общее количество кнопок на пульте. Появление дисплея на пульте может спровоцировать разделение навигации или кнопок на «страницы». Существуют случаи, когда это оправдано, однако любое интерфейсное решение, разделяющее внимание пользователя между двумя

дисплеями (телевизора и пульта в данном случае), подвержено риску запутать пользователя и доставить ему неудобство.

- **Сосредотачивайте внимание на целях и деятельности пользователей, а не на функциях продукта.** Эффективное использование большинства домашних развлекательных систем требует от пользователей понимания топологии и состояний системы. Чтобы посмотреть фильм, пользователь должен знать, как включить телевизор, как включить DVD-проигрыватель, как переключить источник сигнала телевизора на канал, к которому подключен DVD-проигрыватель, как включить систему объемного звука и как перевести телевизор в широкоэкранный режим. Чтобы выполнить все эти действия, требуется до трех пультов или до пяти нажатий кнопок на универсальном функционально богатом пульте. Пульты дистанционного управления, вроде Harmony, подходят к задаче иначе. Элементы управления ориентированы на действия пользователя (скажем, «смотреть фильм») и используют предоставленную пользователем информацию о подключении устройств, чтобы выполнить соответствующую последовательность команд, настраивающих эти устройства. И хотя такую систему гораздо сложнее разработать, она, будучи грамотно реализованной, станет для пользователя очевидным благом.

Проектирование автомобильных интерфейсов

В автомобильных интерфейсах, особенно обладающих сложной телематической функциональностью (навигация и развлекательные системы), серьезную проблему порождает требование обеспечить безопасность езды. Сложное или запутанное взаимодействие, требующее слишком большого внимания, создает опасность на дороге, и, чтобы избежать подобных рисков, необходимы значительные усилия по проектированию и юзабилити-тестированию. Серьезность проблемы усугубляется пространственными ограничениями приборной панели, центральной консоли и руля в автомобиле.

- **Минимизируйте то время, когда руки водителя находятся не на руле.** Востребованные элементы управления (проигрывание/пауза, выключение звука, пропуск/поиск дорожки) должны быть на руле (удобно водителю) и на центральной консоли (удобно пассажиру).
- **Используйте единообразную визуальную компоновку всех экранов.** Выдерживая непротиворечивость стилей, вы позволяете водителю не терять ориентацию при переключении контекстов.
- **По возможности применяйте прямое ассоциирование управляющих элементов.** Подписанные кнопки лучше, чем программируемые элементы управления. Сенсорные кнопки с тактильной обратной связью лучше, чем аппаратные кнопки, связанные с экранными подписями, поскольку требуют меньших когнитивных усилий от водителя.

- **Тщательно выбирайте механизмы ввода.** Водителю гораздо удобнее совершать выбор не посредством набора кнопок, а с помощью вращающихся рукояток. Рукоятки не захламляют интерфейс, поскольку их меньше, они выступают, благодаря чему их легче нащупать, и при грамотном проектировании предоставляют возможность элегантно и интуитивно управлять – как грубо, так и точно.
- **Обеспечьте простое и предсказуемое переключение режимов/контекстов.** Система iDrive компании BMW собрала большинство развлекательных функций автомобиля, управление климатом и навигацией в один элемент управления, представляющий собой гибрид рукоятки и джойстика. Он задумывался как упрощающий взаимодействие, но, перегрузив гибрид функциями, BMW создала опасность для пользователей, которым приходится перемещаться по интерфейсу, чтобы переключить контекст и режим. Режимы (переключение с компакт-диска на радио или с управления климатом на навигацию) должны быть доступны напрямую – по нажатию одной кнопки, а расположение кнопок режимов в интерфейсе системы должно быть фиксированным и единообразным.
- **Реализуйте звуковую обратную связь.** Звуковые подтверждения команд уменьшают необходимость отрывать взгляд от дороги. Однако следует проявлять осторожность и не делать такую обратную связь слишком громкой или отвлекающей. Для автомобильных навигационных систем вербальная обратная связь, дающая маршрутные указания, может быть полезна, если инструкции о том, где поворачивать и по какой улице ехать, проговариваются заранее и водитель успевает на них реагировать. Другая возможность – голосовой ввод для управления интерфейсом. Впрочем, в автомобиле обычно шумно, и совсем не очевидно, что выговаривание команды, особенно если ее требуется повторять или корректировать, дает меньшую когнитивную нагрузку, чем нажатие на кнопку. Замечательная функция с точки зрения маркетинга, но нам кажется, что пока неясно, насколько она делает езду на автомобиле более безопасной и удобной.

Проектирование бытовых устройств

Большинство бытовых устройств наделены крайне простыми дисплеями, а для управления состоянием часто применяются физические кнопки и рукоятки. Однако в некоторых случаях крупные приборы (стиральные машины и сушилки) оборудуются цветными жидкокристаллическими сенсорными экранами, позволяющими организовать информационно насыщенный вывод и прямой ввод.

Интерфейсы бытовых устройств, например интерфейсы телефонов, упоминавшиеся в предыдущем разделе, в основном относятся к временному типу. Пользователи этих интерфейсов редко подкованы технически, а потому им требуется самый простой и самый прямолинейный из возможных интерфейсов. Кроме того, они привыкли к физическим

элементам управления. Рукоятки и кнопки (плюс соответствующая тактильная, звуковая и визуальная обратная связь через простейший индикаторный дисплей или даже обычные лампочки) – правильный выбор, если только применение сенсорного экрана не позволяет достичь какой-то невероятной простоты использования. Многие производители бытовых устройств совершают ошибку, добавляя десятки новых – и совершенно ненужных – возможностей в новейшие цифровые модели. Вместо того чтобы облегчить жизнь, «простой» жидкокристаллический сенсорный экран становится запутанным нагромождением крайне неудобных элементов управления.

Еще один довод в пользу применения в бытовых устройствах интерфейсов временного типа – то, что пользователи этих устройств пытаются добиться совершенно конкретных результатов. Как и пользователи сервисных киосков, они не заинтересованы в исследовании интерфейса или в получении дополнительной информации – они просто хотят включить стиральную машину в нормальном цикле или приготовить ужин из полуфабрикатов.

Существует один аспект проектирования бытовых приборов, требующий иного типа интерфейса. Информацию состояния, указывающую, в каком режиме работает стиральная машина или включена ли запись на видеомагнитофоне, следует представлять в виде минималистичной пиктограммы где-нибудь в углу экрана. Если требуется что-то большее, нежели краткая информация о состоянии, для этого будет уместен дополнительный интерфейс.

Проектирование звуковых интерфейсов

Звуковые интерфейсы, применяемые в системах голосовых сообщений и автоматизированных справочных системах, имеют свои особенности. Самым сложным моментом здесь является навигация, поскольку пользователю легко заблудиться в дереве функций, не видя свое положение в иерархии. Часто репутации сильных брендов вредит плохая организация взаимодействия пользователя с деревом иерархического телефонного меню. (Практически все голосовые интерфейсы основаны на древовидных структурах, даже если варианты выбора скрыты за системой распознавания голоса – которая приносит с собой целый ряд других проблем.)

Ниже перечислены некоторые принципы создания звуковых интерфейсов, пригодных для использования.

- **Организуйте и называйте функции в соответствии с ментальными моделями пользователей.** Этот принцип важен в проектировании любго устройства, но важен вдвойне, когда передача функций осуществляется исключительно вербально, не выходя при этом за контекст текущей функции. Обязательно изучите контекстные сценарии, чтобы выявить самые важные и востребованные функции и сделать их наиболее доступными: они должны идти в начале списка.

- **Обязательно объявляйте доступные в данный момент функции.** После любого действия пользователя сообщайте, какие функции доступны прямо сейчас и как их вызвать.
- **Всегда обеспечивайте возможность возврата на один шаг назад и выхода на верхний уровень.** После каждого действия пользователя сообщайте ему, как можно вернуться на один шаг назад в функциональной иерархии (на один узел вверх по дереву) и как перейти на самый верхний уровень дерева.
- **Всегда предоставляйте пользователю возможность поговорить с оператором.** После каждого выполненного действия интерфейс должен предоставлять пользователю возможность при необходимости связаться с оператором. Это особенно актуально там, где у пользователя могут возникнуть проблемы.
- **Дайте пользователю достаточно времени на ответ.** Системы часто требуют от пользователя словесного ответа или ввода информации с телефонной клавиатуры. Вы должны провести тестирование, чтобы определить оптимальное время ожидания. Не забывайте, что клавиатуры телефонов могут быть неудобны для ввода текстовой информации.

В заключение напомним, что выбор шаблонов проектирования, основанных на типе интерфейса, и выбор технической платформы – первоочередные задачи при проектировании интерактивного продукта. По нашему опыту многие некачественно спроектированные продукты пострадали из-за неспособности разработчиков принять сознательное решение по этим вопросам на каком-то этапе разработки. Не стоит сразу погружаться в детали – сделайте шаг назад и выясните, какая технологическая платформа и какой тип интерфейса позволят наилучшим образом удовлетворять потребности пользователей и бизнеса, а также какие последствия это решение повлечет за собой при детализации взаимодействия.

10

Оркестровка и состояние потока

Если мы хотим, чтобы пользователи наших продуктов работали более продуктивно, эффективно и увлеченно, мы должны убедиться, что пользователь пребывает в нужном настрое. В этой главе мы обсудим ментальную эргономику – способы сделать так, чтобы наши продукты оказывали поддержку человеческому интеллекту и способностям, и избежать разрушения того состояния продуктивной концентрации, в котором должны находиться пользователи.

Состояние потока и прозрачность

Люди, способные всецело сосредоточиться на некоторой деятельности, забывают о посторонних проблемах и отвлекающих факторах. Такое состояние называется **потоком**. Это понятие впервые предложил Михай Чиксентмихайи (Mihaly Csikszentmihalyi) в своей книге «Flow: The Psychology of Optimal Experience» (Csikszentmihalyi, 1991).

Том Демарко (Tom DeMarco) и Тимоти Листер (Timothy Lister) в своей книге «Peopleware: Productive Projects and Teams»[1] описывают поток как «состояние глубокого, почти медитативного погружения в работу». Поток нередко вызывает «мягкое чувство эйфории», так что человек, бывает, перестает замечать течение времени. Важно, что в состоянии потока человек может быть исключительно продуктивным, особенно если он занят созидательной работой, например конструированием, дизайном, разработкой или созданием литературных произведений. Итак, подчеркнем очевидное: чтобы сделать людей более продуктивными и счастливыми, нам надлежит проектировать интерактивные продукты, вызывающие и поддерживающие состояние потока, и при-

[1] Т. Демарко, Т. Листер «Человеческий фактор: успешные проекты и команды». – Пер. с англ. – СПб: Символ-Плюс, 2005.

кладывать все возможные усилия, чтобы избежать любого поведения, потенциально способного разрушить поток. Если программа то и дело выбивает пользователя из потока, ему трудно возвращаться в это продуктивное состояние.

Если бы пользователь мог достичь своих целей волшебным образом, без вашего продукта, он бы так и поступил. Аналогично этому, если бы он, пользуясь вашей программой, мог решать свои задачи без всякого интерфейса, он бы обязательно решал их именно так. Взаимодействие с программным продуктом не несет в себе эстетической составляющей (кроме, быть может, взаимодействия с игровыми, развлекательными и информационными программами вроде веб-броузеров). В большинстве случаев это сугубо утилитарная деятельность, которую следует свести к минимуму.

Каким бы замечательным ни был ваш интерфейс, чем его меньше, тем лучше.

Если вы будете прежде всего думать об интерфейсе как таковом, на первый план выйдут побочные эффекты инструментов и технологий, а не цели пользователя. Пользовательский интерфейс – это артефакт, не связанный напрямую с целями, стоящими перед пользователем. Когда вы в следующий раз будете ликовать по поводу замечательно спроектированного взаимодействия, вспомните, что, в конечном счете, лучшим пользовательским интерфейсом в большинстве случаев является полное отсутствие интерфейса.

Чтобы привести пользователя в состояние потока, вы должны сделать его взаимодействие с программным продуктом **прозрачным**. Когда романист пишет хорошо, писательское ремесло остается незамеченным, а читатель воспринимает только историю и ее персонажей с ясностью, которой техника письма не препятствует. Точно так же, когда продукт хорошо взаимодействует с человеком, механика взаимодействия исчезает, оставляя человека один на один с его целями без какой-либо посреднической функции программы. Плох тот писатель, читатели которого обращают внимание на текст, и плох тот проектировщик взаимодействия, чье неуклюжее присутствие ощущается в созданных им программах.

Хорошо оркестрованные пользовательские интерфейсы прозрачны.

Для романиста не может быть «хорошего» предложения в отрыве от повествования. Не существует правил, благодаря которым предложение становится прозрачным. Все зависит от действий главного героя или от воздействия, которое желает оказать на читателя автор. Писатель понимает, что не следует использовать непонятные слова в осо-

бенно тихом и чувственном абзаце, ибо они прозвучат как фальшивые ноты в струнном квартете.

То же верно и для программного обеспечения. Проектировщик взаимодействия должен учить себя слышать фальшивые ноты в **оркестровке** программного взаимодействия. Жизненно важно, чтобы все элементы интерфейса работали скоординированно для достижения единой цели. Когда общение приложения с человеком организовано хорошо, оно становится практически незаметным.

Словарь Webster определяет оркестровку как «гармоничную организацию» – а от интерактивных продуктов разумно ожидать именно такой организации. Гармоничная организация не следует жестким правилам. Невозможно создать указания вроде «Пять кнопок в диалоговом окне – это хорошо» и «Семь кнопок в диалоговом окне – это уже слишком много». Однако легко догадаться, что не стоит создавать диалоговое окно с 35 кнопками. Серьезное затруднение, которое вызывают рассуждения такого рода, состоит в том, что проблема рассматривается *в искусственных условиях*. Подобный анализ не принимает во внимание задачу, которую требуется решить, равно как и то, чем занят в данный момент человек и чего он пытается добиться.

Проектирование гармоничного взаимодействия

Не существует универсальных правил, определяющих гармоничное взаимодействие, как не существует универсальных правил определения гармоничных пауз в музыке, однако мы находим, что перечисленные ниже стратегии направляют проектирование взаимодействия в нужное русло:

1. Следуйте ментальным моделям пользователей.
2. Меньше – лучше.
3. Позволяйте пользователям управлять, не принуждайте их к диалогу.
4. Держите инструменты под рукой.
5. Обеспечивайте немодальную обратную связь.
6. Проектируйте наиболее вероятное, будьте готовы к возможному.
7. Предоставляйте информацию о контексте.
8. Организуйте непосредственное манипулирование и графический ввод.
9. Отображайте состояния объектов и статус приложения.
10. Избегайте ненужных сообщений.
11. Не используйте диалоговые окна, чтобы сообщить, что все нормально.
12. Избегайте чистого листа.
13. Просите прощения, а не разрешения.

14. Отделяйте функции от их настройки.
15. Не задавайте вопросы – предоставляйте выбор.
16. Прячьте рычаги катапультирования.
17. Оптимизируйте скорость реакции; предупреждайте о задержках.

Обсудим эти принципы более подробно.

ПРИНЦИП проектирования

Следуйте ментальным моделям пользователей.

Понятие ментальных моделей было введено в главе 2. Разные пользователи имеют разные ментальные модели определенной деятельности или процесса, но эти модели редко представлены в терминах того, как на самом деле функционируют компьютеры. Каждый пользователь естественным образом формирует в уме образ того, как программа выполняет свою работу. Мозг пытается найти какую-нибудь причинно-следственную связь между событиями и объяснить поведение компьютера.

Например, в медицинской информационной системе медперсонал имеет ментальную модель базы данных о пациентах, основанную на медицинских картах, с которыми они сталкиваются в реальной жизни. Следовательно, имеет смысл реализовать поиск информации о пациенте по имени. Каждый врач наблюдает нескольких больных, так что имеет смысл дополнительно фильтровать пациентов в интерфейсе информационной системы так, чтобы врач мог выбирать пациента из списка, отсортированного по именам. С другой стороны, сотрудников больничной бухгалтерии в первую очередь интересуют неоплаченные счета пациентов. Изначально им все равно, как зовут больных, – важны сроки оплаты счетов (и, возможно, суммы). Значит, для бухгалтерии интерфейс информационной системы должен сортировать записи о пациентах по срокам задержки оплаты счетов и, возможно, по их суммам, а имена пациентов отходят на второй план.

ПРИНЦИП проектирования

Меньше – лучше.

Во многих случаях действует правило «чем больше – тем лучше». В области проектирования взаимодействия справедливо обратное утверждение, и мы должны постоянно бороться за сокращение количества элементов интерфейса без сокращения возможностей продукта. Чтобы добиться этого, мы должны сделать много, пользуясь минимумом средств, и здесь особенно важна тщательная оркестровка. Мы должны координировать и контролировать функциональность программного продукта, не позволяя интерфейсу превращаться в нагромождение диалоговых окон, усыпанных нелогичными и редко используемыми элементами управления.

Легко создать сложный, но не очень мощный интерфейс. Как правило, в подобных продуктах функциональность изолируется в отдельные «гетто», и пользователь может выполнить какую-то одну задачу, не имея доступа к средствам решения смежных задач. Когда в 1995 году вышло первое издание этой книги, проблема была повсеместной. Взять такую простую вещь, как диалоговое окно сохранения файла в Windows-приложении: в этом диалоговом окне не было возможности переименовать или удалить файлы, представленные пользователю. Пользователь был вынужден выполнять эти родственные сохранению файлов задачи обходными путями, что, в конечном счете, требовало от операционной системы *дополнительных интерфейсов*. К счастью, современные операционные системы лучше справляются с такими тонкостями. Поскольку теперь принято предлагать функциональность, соответствующую контексту пользователя, человеку реже приходится перебирать различные области интерфейса, чтобы решать простые распространенные задачи.

Впрочем, у нас впереди еще долгий путь. Мы часто сталкиваемся с программами для бизнеса, в которых каждая функция или возможность «живет» в отдельном диалоге или окне, словно разработчики не задумывались, каким образом людям необходимо сочетать функции для решения определенных задач. Пользователям часто приходится выполнять команду меню, чтобы получить фрагмент информации, затем копировать эту информацию в буфер обмена, после чего в другом окне выполнять еще одну команду – и все это лишь для того, чтобы вставить скопированную информацию в поле ввода. Это не просто неэлегантное и грубое решение – это подход, провоцирующий ошибки и неспособный поддерживать продуктивное разделение труда между человеком и машиной. Как правило, подобные продукты создаются неспециально – это получается в ходе многолетней разработки с принятием решений на ходу или вследствие разработки продукта несколькими изолированными командами, обитающими в отдельных резервациях внутри организации.

Популярная модель телефона Motorola – Razr – хорошо иллюстрирует эту проблему: промдизайн телефона заслуживает всяческих похвал за элегантность результата, а вот программное обеспечение взято от предыдущего поколения телефонов Motorola и, похоже, разрабатывалось несколькими нескоординированными командами. Так, интерфейс ввода текста в записной книжке телефона отличается от интерфейса ввода текста в календаре. По-видимому, каждая из команд разработчиков создала собственное решение, что и дало два различных интерфейса, решающих одну и ту же задачу; все это – пустая трата ресурсов при разработке, а также источник путаницы и затруднений для пользователей продукции Motorola.

Классическая книга Маллета (Mullet) и Сано (Sano) «Designing Visual Interfaces» (Mullet and Sano, 1995) содержит плодотворное обсужде-

ние идеи **элегантности**, которую можно считать новым, простым, экономичным и изящным способом решать задачи проектирования. Поскольку обычно программное обеспечение интерактивных продуктов является невероятно сложным, все более ценными становятся элегантность и простота. Это необходимые свойства технологии, предназначенной служить человеку и его потребностям.

Минималистичный подход к проектированию продуктов неизбежно связан с четким пониманием назначения: чего пытается достичь пользователь, применяя инструмент? Без понимания назначения интерактивные продукты – не более чем неорганизованное нагромождение функций, демонстрирующих работу технологий. Показательным примером того, как глубокое понимание назначения привело к созданию минималистичного пользовательского интерфейса, является поисковая система Google, где есть лишь текстовое поле, две кнопки («Поиск в Google», отображающая перечень результатов, и «Мне повезет!», выполняющая переход к первому сайту из результатов поиска), логотип компании Google и пара гиперссылок на вселенную функций Google (рис. 10.1). Другой хороший пример минималистичного пользовательского интерфейса – iPod Shuffle. Компания Apple, определив со всей тщательностью набор уместных возможностей, служащий конкретным потребностям пользователей, создала превосходный по удобству продукт с одним переключателем, пятью кнопками – и без экрана! Невероятно простой текстовый редактор WriteRoom (Hog Bay Software) вообще не имеет пользовательского интерфейса – только область, в которой можно набирать текст. Текст сохраняется автоматически, избавляя от необходимости иметь дело с файлами.

Важно помнить, что стремление к простоте может зайти слишком далеко – сокращение интерфейса есть поиск баланса, требующий хорошего понимания ментальных моделей пользователей. Упомянутый выше аппаратный интерфейс iPod – пример элегантного и сдержанного проектирования – тем не менее противоречит ожиданиям некоторых пользователей. Если вы жили в мире кассетных дек и проигрывателей для компакт-дисков, вероятно, использование переключателя

Рис. 10.1. *Знаменитый поисковый интерфейс Google – классический пример минималистичного проектирования интерфейса, в котором каждый элемент на экране содержателен и понятен*

Проигрывание/Пауза на устройстве iPod для выключения устройства, а кнопки Меню – для включения устройства покажется вам странным. Это классический случай того, как визуальная простота может приводить к высокому когнитивному сопротивлению. В данном случае идиомы достаточно просты, чтобы их можно было легко заучить, и последствия неправильных действий не очень страшны, так что на успех продукта в целом это не повлияло.

Позволяйте пользователям управлять, не принуждайте их к диалогу.

Многие разработчики, похоже, воображают, что идеальный интерфейс должен иметь форму диалога с пользователем. Однако пользователи в большинстве своем так не думают. Они предпочитают взаимодействовать с программой примерно так же, как с автомобилем: они открывают дверцу и садятся в машину, когда хотят куда-нибудь съездить; они давят на педаль газа, когда хотят, чтобы машина ехала, и на тормоз, чтобы она остановилась; они поворачивают руль, когда хотят повернуть.

Такое идеальное взаимодействие не является диалогом. Скорее, это использование средства передвижения. Когда плотник забивает гвозди, он не вступает в диалог с молотком, а просто стучит по гвоздю. Водитель командует автомобилем, когда хочет изменить его поведение. Водитель ждет от автомобиля и окружения прямого отклика, уместного в контексте: вид из окна, показания приборов, свист ветра и звук шин на дороге, боковые перегрузки и вибрация от неровностей дороги. Есть своя обратная связь и у плотника: он чувствует, как гвоздь уходит в доску, слышит стук молотка и чувствует его тяжесть.

Водитель явно не ожидает, что автомобиль вступит с ним в переговоры с помощью диалогового окна, а плотник тем более не оценит появление на молотке диалогового окна (вроде того, что изображено на рис. 10.2).

Одной из причин, по которым программный продукт нередко раздражает пользователей, является то, что он ведет себя не так, как автомо-

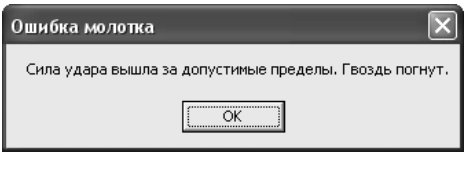


Рис. 10.2. *Из того, что программисты привыкли к подобным сообщениям, вовсе не следует, что это справедливо в отношении представителей других профессий. Никто не хочет, чтобы компьютер его ругал. Если мы заставляем его сделать глупость, то мы вправе ожидать, что он эту глупость сделает. Конечно, наши инструменты должны защищать нас от непоправимых ошибок, но защищать и ругать – не одно и то же*

биль или молоток. Он часто имеет наглость вовлекать нас в диалог: сообщает нам о наших промахах и требует от нас ответа. С точки зрения пользователей, все должно быть наоборот: пользователь требует, а компьютер отвечает.

Непосредственное манипулирование позволяет нам указывать на то, что мы хотим. Если мы хотим передвинуть объект из точки A в точку B, мы щелкаем по нему и перетаскиваем мышью. Вообще говоря, лучшие интерфейсы, вызывающие у пользователя состояние потока, – это интерфейсы с большим количеством развитых идиом непосредственного манипулирования.

ПРИНЦИП проектирования

Держите инструменты под рукой.

Большинство приложений сложны настолько, что одного режима непосредственного манипулирования недостаточно для реализации всех функциональных возможностей. Как следствие, приложение обычно предлагает пользователю набор разнообразных инструментов. Эти инструменты в действительности являются различными режимами поведения продукта. Предоставление инструментальных средств потенциально усложняет программу, и требуется затратить еще много сил, чтобы манипулирование инструментом стало действительно простым и не выводило пользователя из состояния потока. В первую очередь мы должны обеспечить полноту и доступность информации о текущем инструменте, а также быстроту и легкость переключения с одного инструмента на другой.

Инструменты должны быть под рукой – предпочтительно на палитре или на панели инструментов для начинающих и середняков, а также в виде клавиатурных сокращений – для специалистов. Тогда пользователь будет их видеть и сможет одним щелчком или нажатием клавиши выбрать любой из них. Если пользователь вынужден переключать внимание с основного занятия на поиски нужного инструмента, он утратит сосредоточенность. Это все равно что встать из-за стола и пойти в другой конец дома за карандашом. Кроме того, никогда не заставляйте пользователя вручную убирать инструменты.

ПРИНЦИП проектирования

Обеспечивайте немодальную обратную связь.

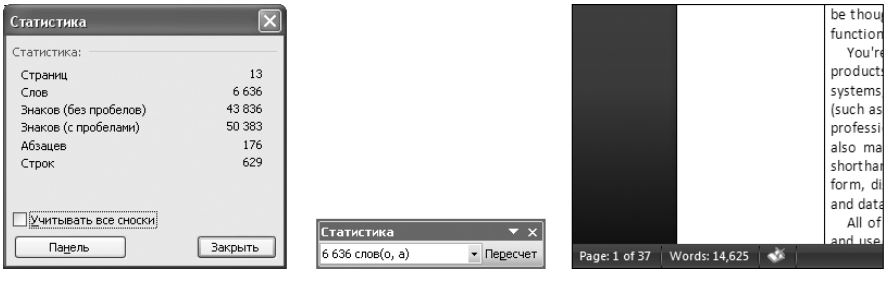
Когда пользователь интерактивного продукта манипулирует инструментами и данными, как правило, желательно, чтобы программа сообщала ему об их состоянии и о результатах произведенных манипуляций. Эта информация должна быть легко читаемой и понятной, не заслоняющей элементы интерфейса и не препятствующей нормальной работе пользователя.

Приложение имеет ряд способов представлять информацию или осуществлять обратную связь с пользователем. К сожалению, самым распространенным подходом является вывод диалогового окна. Эта техника модальна: программа переходит в специальный режим, требующий реакции пользователя, без которой она не может вернуться в нормальное состояние, а пользователь не может продолжать работу. Более удачным подходом является обеспечение **немодальной обратной связи**.

Обратная связь *немодальна*, если информация для пользователя включена в основной интерфейс и не прерывает нормальную работу системы и ее взаимодействие с пользователем. Например, редактор Word сообщает вам номер текущей страницы и раздела, количество страниц в документе, позицию курсора и текущее время. Вся эта информация немодальна: вы просто смотрите на строку состояния внизу окна, и для получения информации вам не требуется выполнять сложные действия.

Однако при желании узнать количество слов в документе нужно вызвать диалоговое окно Статистика из меню Сервис (рис. 10.3,*а*), в котором есть доступ к панели статистики (рис. 10.3,*б*), но даже на ней придется постоянно нажимать кнопку Пересчет для получения точной информации. Авторы журнальных статей, которым важно знать количество слов, предпочли бы получать эту информацию в немодальной форме. И хотя многим людям такая статистика не нужна, внизу экрана есть свободное пространство, способное вместить подобную информацию.

В кабине истребителя есть специальное устройство, которое выводит показания важнейших приборов прямо на лобовое стекло. Пилоту даже не приходится пользоваться периферийным зрением: он видит всю



а) б) в)

Рис. 10.3. *Работая в редакторе Word 2003, вы можете узнать количество слов в документе, выполнив команду Статистика из меню Сервис. Команда открывает диалоговое окно. Чтобы вернуться к работе, вы должны нажать кнопку Закрыть. Такое поведение является полной противоположностью немодальной обратной связи, и оно выводит пользователя из состояния потока. В Word 2007 Microsoft значительным образом улучшила ситуацию: число слов в документе немодально отображается в левом нижнем углу окна рядом со счетчиком страниц (в), поскольку это родственные значения*

необходимую информацию, не отрывая взгляда от самолета противника. Приложение может использовать края экрана для отображения информации о том, что происходит в рабочей области. Многие графические приложения, такие как Adobe Photoshop, имеют на периферии окна линейки, миниатюры и другие элементы немодальной обратной связи. Типы обогащенной немодальной обратной связи подробно обсуждаются в главе 25.

Проектируйте наиболее вероятное, будьте готовы к возможному.

Существует много случаев, когда взаимодействие с пользователем (обычно в форме диалоговых окон) проникает в интерфейс без особой необходимости. Часто это происходит, когда программа стоит перед выбором. Большинство программистов решают проблему выбора с позиций логики, и это отражается на создаваемом продукте. С точки зрения логики, если утверждение справедливо в 999 999 случаях из миллиона и ложно в одном случае, то оно ложно. Так работает булева логика. Однако для большинства людей оно будет безусловно истинным. Утверждение *может* быть ложным, но *вероятность* этого мала настолько, что ею можно пренебречь. Одним из самых продуктивных методов оркестровки пользовательских интерфейсов является разграничение возможного и вероятного.

Программисты обычно считают, что возможность и вероятность – одно и то же. Например, пользователь может завершить работу с программой и сохранить свою работу, а может выбросить документ, над которым работал в течение шести часов. Любой поворот событий возможен. Вероятность того, что пользователь выбросит результаты своего труда, – не больше, чем одна тысячная. Тем не менее типичная программа содержит диалоговое окно с вопросом, не хочет ли пользователь сохранить изменения (рис. 10.4).

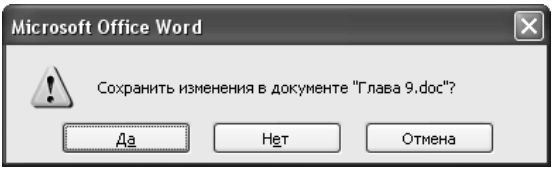


Рис. 10.4. *Это, бесспорно, самое ненужное диалоговое окно в мире графических пользовательских интерфейсов. Конечно же, мы хотим сохранить свою работу! Это нормальный ход событий. Отказ сохранить ее был бы таким экзотическим событием, что обрабатывать его нужно было бы в каком-нибудь редко используемом диалоговом окне. Одно это диалоговое окно заставляет пользователя узнать едва ли не больше бесполезных и сбивающих с толку фактов об оперативной памяти и жестком диске, чем любое другое взаимодействие с компьютером. От этого диалогового окна следует отказаться*

Диалоговое окно, изображенное на рис. 10.4, неуместно и бесполезно. Как часто вы отказываетесь от изменений, которые внесли в документ? Это диалоговое окно подобно сварливой жене, предупреждающей мужа, чтобы он не пролил суп на рубашку, всякий раз, как он обедает. Мы обсудим смысл отказа от этого окна в главе 17.

Умения программистов оценивают по их способности создавать программы, обрабатывающие множество возможных, но маловероятных условий, возникающих в сложных логических системах. Однако это не означает, что они должны переносить свое умение обрабатывать экзотические ситуации на пользовательский интерфейс. Навязчивое присутствие пограничных ситуаций выдает те интерфейсы, которые были спроектированы программистами. Диалоговые окна, элементы управления и настройки, используемые сто раз в день, расположены рядом с окнами, элементами и настройками, которые могут понадобиться раз в год или не нужны вообще.

Возможно, что с вашей машиной столкнется автобус, однако *вероятно*, что вы доберетесь до работы без приключений. Вы же не останетесь дома из страха перед автобусом? Так пусть возможные события не отвлекают вас от вероятных в тот момент, когда вы разрабатываете интерфейс.

Представляйте информацию с учетом контекста.

Способ представления информации приложением – это еще один путь для создания помех в сознании пользователя. Особые злоупотребления наблюдаются при представлении количественной, то есть цифровой информации. Если приложение должно сообщить пользователю об объеме свободного пространства на диске, оно может поступить, как в свое время поступал Менеджер файлов в Windows 3.x, а именно – вывести *точное* количество свободных байтов (рис. 10.5).

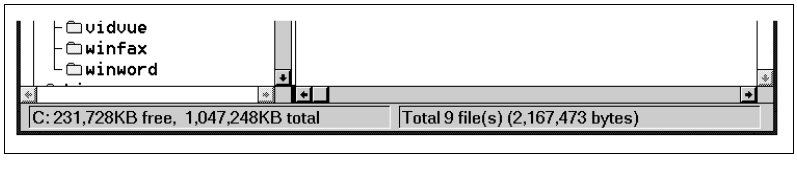


Рис. 10.5. *Менеджер файлов в Windows 3.x гордо сообщал пользователю точное количество байтов, занятых файлами на диске. Помогает ли такая точность понять, что диск пора чистить? Конечно, нет. Более того, разве семизначное число является лучшим способом индикации состояния диска? Разве графическое представление соотношения свободного и занятого пространства (например, в виде круговой диаграммы) не было бы более осмысленным? К счастью, теперь Microsoft Windows использует для индикации свободного места на диске круговые диаграммы*

В левом нижнем углу окна выведено количество свободных байтов на диске и общее количество байтов на диске. Эти числа трудно воспринимать и трудно интерпретировать. Когда диск содержит больше десятка миллиардов байтов, для нас уже неважно, сколько сотен байтов свободно. Тем не менее программа подсчитывает килобайты. Несмотря на такую точность, передача информации затруднена. В действительности пользователь хочет знать, заполнен ли диск до отказа – или можно добавить программу размером 20 Мбайт и еще останется достаточно места. Голые цифры, какими бы точными они ни являлись, мало помогают в этом вопросе.

Специалист в области визуального представления данных Эдвард Тафти (Edward Tufte) говорит, что представление количественной информации должно отвечать на вопрос: «По сравнению с чем?» Знание того, что на диске свободно 231 728 Кбайт, не столь полезно, как знание того, что это 22% от общего размера диска. Другой афоризм Тафти: «Показывайте данные», а не просто сообщайте о них в текстовой или численной форме. Круговая диаграмма, показывающая занятое и свободное место на диске разным цветом, гораздо понятнее сообщает о масштабе и пропорциях использования дискового пространства. Она объясняет нам, *что* в действительности означает число 231 728 Кбайт. Цифры должны оставаться на экране, но их статус должен быть сведен к меткам на диаграмме. Кроме того, их точность должна находиться в пределах разумного и быть одинаковой повсеместно. Смысл информации должен быть представлен визуально, а цифры используются лишь для поддержки.

В Windows XP и Windows Vista корпорация Microsoft одной рукой дает, а другой отбирает. Менеджер файлов (см. рис. 10.5) уже давно канул в небытие, а на смену ему пришел Проводник (рис. 10.6). Эта замена привела к появлению диалогового окна со свойствами жесткого диска. Занятое пространство окрашено в синий цвет, а свободное – в лиловый. Такую круговую диаграмму легко воспринимать. Теперь вы моментально определяете, что – о счастье! – ваш жесткий диск GranFromage почти пуст.

К сожалению, эта круговая диаграмма не встроена в интерфейс Проводника. Пользователю приходится искать в меню пункт, позволяющий ее открыть. Чтобы узнать, насколько заполнен диск, вы должны вызвать модальное диалоговое окно, которое, хотя и сообщает нужную информацию, отрывает вас от основного занятия и уводит далеко от того места, где потребовалась эта информация. Проводник – программа, которая позволяет просматривать, копировать, перемещать и удалять файлы, но в нем вы не можете легко узнать, пора ли вам что-то удалить. Эта секторная диаграмма должна быть встроена в интерфейс Проводника. В Windows 2000 она отображается в левой части окна при выборе диска в Проводнике. В XP Microsoft сделала шаг назад – и диаграмма снова прописалась в диалоговом окне. Следует выводить

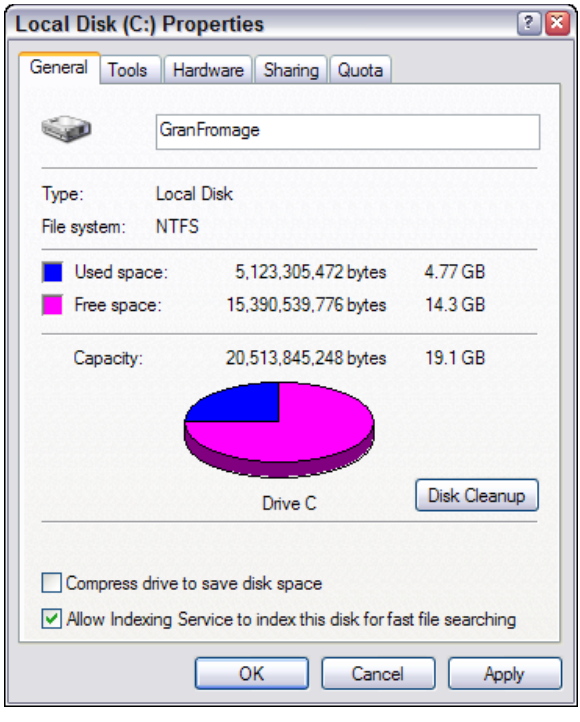


Рис. 10.6. *В Windows XP и Windows Vista компания Microsoft заменила электрический стул смертельной инъекцией. Вместо длинных неразборчивых чисел внизу окна Менеджера файлов можно открыть диалоговое окно свойств в Проводнике. Хорошая новость: наконец нам наглядно дают понять, как чувствуют себя жесткие диски. Плохая новость: чтобы это понять, необходимо прекратить текущую деятельность и открыть диалоговое окно – всего лишь для получения базовой информации, которая должна быть доступна всегда. В Windows 2000 эта диаграмма автоматически отображалась в левой части окна Проводника при выборе диска; решение Windows XP представляет собой шаг назад*

круговую диаграмму наряду с численными данными все то время, пока открыто окно Проводника, если пользователь не предпочтет убрать диаграмму с экрана.

Организуйте непосредственное манипулирование и графический ввод.

Программные продукты редко представляют численную информацию в наглядном виде. Еще реже они позволяют вводить данные в графической форме. Многие программы дают возможность вводить числа, а затем – по команде – преобразуют эти числа в графическое представле-

ние. Редкие продукты позволяют пользователю создать график и по команде преобразовать его в последовательность чисел. Исключение – современные текстовые процессоры: почти каждый процессор позволяет указывать позиции табуляции и отступы путем перетаскивания маркера на линейке. Пользователь как бы говорит: «Я хочу, чтобы абзац начинался здесь», – и позволяет программе вычислить, что отступ равен 1,347 дюйма от левого поля. К счастью, пользователь не обязан вводить число 1,347 вручную.

Этот принцип действует в самых разных ситуациях. Когда нужно упорядочить элементы какого-либо списка, пользователь, возможно, захочет расположить их в алфавитном порядке. Однако он может захотеть расположить их по своему вкусу, а не в соответствии с неким алгоритмом. У пользователя должна быть возможность перетаскивать элементы списка непосредственно, без вмешательства каких бы то ни было алгоритмов.

Отображайте состояния объектов и статус приложения.

Когда человек спит, он выглядит спящим. Когда он проснулся, он выглядит бодрствующим. Если человек занят делом, он и выглядит соответственно: его взгляд сосредоточен на объекте работы, а поза является закрытой и демонстрирует занятость. Если человек, наоборот, ничем не занят, это тоже понятно по его виду: его поза открыта, а взгляд блуждает в поисках контакта. Люди не просто ожидают друг от друга подобной обратной связи; поддержание общественного порядка приводит к определенной зависимости от этой обратной связи.

Наши приложения и устройства должны вести себя аналогично. Когда программа «спит», она должна выглядеть спящей. Приложение в активном состоянии и должно выглядеть активным, а приложение, выполняющее определенные действия, должно выглядеть соответственно. Когда компьютер занят важной работой, например выполняет сложные вычисления или подключается к базе данных, должно быть очевидно, что приложение не будет так шустро реагировать на наши запросы, как обычно. Когда компьютер отправляет факс, мы должны видеть анимацию сканируемого и отправляемого факса (или хотя бы немодальный индикатор хода операции).

Точно так же состояние объектов пользовательского интерфейса должно быть ясным для пользователей. Большинство приложений электронной почты умеют очевидным образом показывать, какие сообщения не были прочитаны, на какие был написан ответ, а какие пользователь уже переслал. Разовьем идею: правда, было бы здорово, если, глядя на события в календаре Microsoft Outlook или IBM Lotus Notes, можно было бы понять, сколь многие люди согласились участвовать и как много людей еще не ответили?

Отображать состояние программы и объектов лучше всего с помощью обогащенной немодальной обратной связи, которая уже обсуждалась ранее в этой главе. Более подробное обсуждение и примеры немодальной обратной связи можно найти в главе 25.

Избегайте ненужных сообщений.

Программистам необходимо точно знать, что происходит в программе. Для них это связано с возможностью детального контроля внутренних процессов. Зато пользователям нет никакого дела до мелких подробностей того, что творится внутри программы. Например, люди, не разбирающиеся в технике, могут быть встревожены сообщением, что база данных была модифицирована. Гораздо лучше, когда программа просто выполняет свою работу, сообщает об успешном завершении операций и не нагружает пользователя подробностями о том, *как* она добилась результата.

Многие приложения усердно держат пользователя в курсе всех подробностей своей работы, даже если он не имеет ни малейшего представления о том, что делать с этой информацией. Программы выводят диалоговые окна, сообщающие нам, что соединение установлено, записи отправлены, пользователи зарегистрированы в системе, транзакции записаны, данные переданы, – и уйму других бесполезных фактов. Для программистов эти сообщения – все равно что ровный гул мотора, журчание ручья или плеск волн о берег моря: они свидетельствуют о том, что все благополучно. По всей видимости, эти сообщения использовались при отладке программы. Однако для обычного человека они равносильны тревожным огням на горизонте, крикам в ночи или предметам, самостоятельно летающим по комнате.

Как было сказано ранее, приложение должно недвусмысленно сообщить пользователю, что выполняет некую работу, но подробная обратная связь должна быть организована более тонко. В частности, вывод сведений, перечисленных выше, в форме диалоговых окон прервет взаимодействие пользователя с программой, не предложив взамен никакой компенсации.

Важно не прерывать работу ради выдачи сообщения о том, что все протекает нормально. Когда происходит ожидаемое событие, не нужно сообщать о нем с помощью диалогового окна. Поберегите диалоговые окна для событий, выходящих за рамки нормального положения дел.

Не используйте диалоговые окна, чтобы сообщить, что все нормально.

По сходным причинам не следует прерывать работу, чтобы побеспокоить пользователя сообщением о несерьезных проблемах. Если программа не смогла создать соединение с сервером, не выводите ради этого диалоговое окно. Поместите в окне программы индикатор состояния, чтобы заинтересованный пользователь знал, что происходит, а пользователь, занятый другими делами, не отвлекался.

Секрет оркестровки взаимодействия с пользователем заключается в подходе, ориентированном на цели. Вы должны спросить себя, способно ли конкретное взаимодействие быстро приблизить пользователя непосредственно к его цели. Современные приложения нередко отказываются делать хоть что-то самостоятельно, без команды пользователя. Однако пользователь предпочел бы, чтобы приложение предприняло разумный первый шаг, который потом можно было бы скорректировать. Так программа приблизила бы к его цели.

ПРИНЦИП проектирования

Избегайте чистого листа.

Гораздо проще не задумываться о том, чего хочет пользователь, а задать ему кучу вопросов и принять решение на основе его ответов. Сколько вы видели приложений, начинающих работу с большого диалогового окна с обширным списком вопросов? Однако *нормальные* люди (не эксперты) чувствуют себя неуютно, когда им приходится объяснять интерактивному продукту, чего они хотят. Они предпочли бы, чтобы программа сделала то, что, *по ее мнению*, следует сделать, а затем скорректировали бы результат. В большинстве случаев программа в состоянии сделать правильное предположение на основании предыдущего опыта. Например, когда вы создаете новый документ в Microsoft Word, то получаете пустой документ с конкретными отступами и прочими атрибутами, а не оказываетесь лицом к лицу с диалоговым окном, требующим указать все эти детали. Программа PowerPoint ведет себя менее адекватно – она просит пользователя выбрать стиль презентации для каждой новой презентации. Обе программы стали бы гораздо приятнее, если бы запоминали часто применяемые и недавно использованные стили или шаблоны и задействовали их по умолчанию для новых документов.

ПРИНЦИП проектирования

Просите прощения, а не разрешения.

Из того, что мы часто применяем к интерактивным продуктам слово *думает*, вовсе не вытекает, что программа обязана быть интеллектуальной (как это слово понимают люди) и пытаться выработать правильную линию поведения посредством рассуждений. Она просто должна опираться на статистику и совершать действия, правильность

которых весьма вероятна, а затем предоставлять пользователю развитые инструменты для корректировки первой попытки. Это гораздо лучше, чем выдавать пользователю чистый лист, вынуждая его заполнять этот лист самостоятельно. В результате программа не просит разрешения действовать, но просит прощения за уже содеянное.

Для большинства людей совершенно пустой лист является неудобной отправной точкой. Им гораздо легче начать, если на нем уже что-то написано. Пользователь с готовностью подкорректирует предложенный программой черновик, лишь бы избавить себя от умственных усилий, необходимых при разработке проекта с нуля. Как будет показано в главе 11, лучший способ добиться этого – наделить программу хорошей памятью.

Отделяйте функции от их настройки.

Еще одна проблема довольно часто возникает, когда пользователи вызывают функции с большим количеством параметров. Причина проблемы кроется в неспособности разработчиков различать функцию и *настройку* этой функции. Если вы просите приложение выполнить саму функцию, приложение должно просто выполнить ее, а не допрашивать вас о подробностях настройки. Если вы захотите уточнить свои требования, то вызовете диалоговое окно для настройки.

Например, в ответ на просьбу пользователя распечатать документ многие программы открывают сложное диалоговое окно, требующее указать количество копий, ориентацию бумаги, лоток принтера, размеры полей, цветной или монохромной должна быть печать, какой нужен масштаб, использовать ли встроенный шрифт или шрифт PostScript, печатать ли весь документ, текущую страницу либо только выделенный фрагмент, следует ли печатать в файл, и если да, то какое имя будет у этого файла. Все эти возможности полезны, но ведь мы хотели всего лишь распечатать документ и просили только об этом.

Более осмысленный интерфейс включал бы в себя команду печати и отдельную команду для настройки печати. Первая должна обходиться без диалоговых окон и всего лишь выполнять печать, придерживаясь либо последних настроек, либо настроек по умолчанию. Функция настройки печати предложит пользователю все те варианты выбора, связанные с бумагой, шрифтами и количеством копий, что были перечислены выше. Будет вполне логично, если окно настройки позволит тут же распечатать документ.

Кнопка печати на панели инструментов в редакторе Word позволяет распечатать документ без всяких диалоговых окон. Для многих людей это идеальный вариант, но тем, кто имеет несколько принтеров или печатает через сеть, такая кнопка предлагает слишком мало информации. Пользователю может быть интересно узнать, какой принтер выбран,

прежде чем он щелкнет по кнопке или вызовет диалоговое окно, позволяющее выбрать другой принтер. Вот хороший повод для простого индикатора немодальной обратной связи на панели инструментов или в строке состояния. (В настоящее время он реализован как всплывающая подсказка у кнопки печати, что само по себе неплохо, однако обратную связь можно улучшить.) Диалоговое окно настройки печати вызывается командой Печать... из меню Файл. Название команды могло бы быть более понятным, хотя многоточие, согласно стандартам графического пользовательского интерфейса, намекает на существование диалогового окна.

Существует огромная разница между настройкой и вызовом функции. Настройка, в принципе, может и не включать выполнение функции, но вызов функции уж точно не должен включать в себя настройку. Вообще говоря, любой пользователь вызывает команду в десять раз чаще, чем настраивает ее. Пусть лучше он запросит диалоговое окно настройки один раз из десяти, чем будет отказываться от интерфейса настройки *девять* раз из десяти.

Решение Microsoft по поводу функции печати разумно. Размещайте кнопки прямого доступа к функции на панели инструментов, а доступ к диалоговым окнам настройки этих функций реализуйте в виде пунктов меню. Диалоговые окна настройки являются прекрасными обучающими средствами, а кнопки обеспечивают немедленное действие.

Не задавайте вопросы – предоставляйте выбор.

Задавать вопросы – совсем не то же самое, что предоставлять выбор. Разница между этими действиями такая же, как между собеседованием с кандидатом на должность и выбором товаров на полках супермаркета. Человек, задающий вопросы, оказывается в положении, более высоком по отношению к тому, кого он опрашивает. Начальство спрашивает, подчиненные отвечают. Программа, задающая вопросы, заставляет пользователя ощущать себя подчиненным и вызывает у него раздражение.

Диалоговые окна (особенно диалоги подтверждения) задают вопросы. Панели инструментов предоставляют выбор. Диалоговые окна подтверждения прерывают работу, требуют ответа и не уходят, пока не получат то, чего хотят. Панели инструментов всегда присутствуют на экране, тихо и вежливо предлагая то, что у них есть, подобно хорошо организованному супермаркету, предоставляя клиенту роскошь любого выбора одним движением пальца.

В противоположность тому, что думают многие разработчики программ, вопросы и возможность выбора далеко не всегда вызывают у пользователя ощущение собственного могущества. Гораздо чаще пользователь чувствует себя затравленным и встревоженным.

– *Вы желаете суп или салат?*

– Салат.

– *С капустой или шпинатом?*

– Со шпинатом.

– *Вы предпочитаете кухню французскую, итальянскую или островов Океании?*

– Французскую.

– *Обычную или с местным колоритом?*

– Стоп! Принесите мне суп!

– *Вы желаете суп из морепродуктов или куриный бульон с лапшой?*

Пользователи не любят, когда им задают вопросы. У них создается впечатление, что программа:

- невежественна;
- забывчива;
- слаба;
- безынициативна;
- не способна позаботиться о себе;
- капризна;
- излишне требовательна.

Обычно мы осуждаем эти качества в людях – так почему мы должны терпеть их в программном продукте? В отличие от друга за ужином в ресторане, программа задает нам вопросы не потому, что ей просто интересно наше мнение, и не из желания поддержать разговор. Она ведет себя как человек невежественный и пытающийся преувеличить свою значительность. Программа не интересуется нашим мнением – она требует информацию, причем информацию не первоочередной важности. (Более подробный разговор о том, как избежать ненужных вопросов, ведется в главе 12.)

Хуже однократных вопросов – вопросы, задаваемые многократно и без необходимости. Многие банкоматы постоянно спрашивают у пользователей о предпочтительном языке: «Испанский, английский или китайский?». Но вряд ли этот ответ изменится, когда пользователь освоится с банкоматом. Программа, задающая меньше вопросов, покажется пользователю более умной, вежливой и внимательной.

Социологи из Стэнфорда Клиффорд Насс (Clifford Nass) и Байрон Ривз (Byron Reeves) в своей работе «The Media Equation» (Nass and Reeves, 1996) делают убедительный вывод, что люди обращаются с компьютерами и другими интерактивными устройствами, *как с людьми*, и реагируют на них, *как на людей*. Следовательно, мы должны позаботиться о качествах «личности», которую видят пользователи в нашей программе. Является ли она скромной, компетентной и услужливой – или жа-

лобно ноет, раздраженно ворчит, изводит пользователя, а затем просит прощения? В главе 12 мы обсудим, как сделать программный продукт более вежливым и деликатным.

Важно предоставлять пользователю выбор, но существует разница между возможностью сделать выбор на основании предложенной информации и необходимостью подвергаться обязательным допросам (касающимся формы, а не существа дела) со стороны программы. Пользователи предпочли бы управлять программой, как они управляют автомобилем. Автомобиль предлагает водителю богатый выбор действий, обходясь при этом без диалоговых окон. Вообразите себе ситуацию, представленную на рис. 10.7.

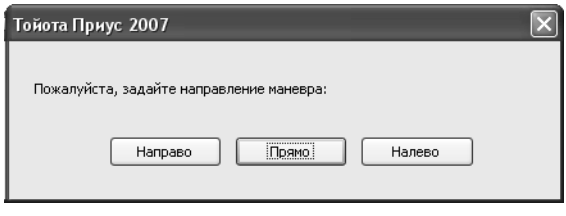


Рис. 10.7. *Представьте себе, что вам приходится управлять автомобилем, щелкая по кнопкам в диалоговом окне! Вы поймете, какие чувства вызывают у обычных людей диалоговые окна в вашей программе. Унизительно, да?*

Прямое манипулирование рулем – не просто подходящая идиома связи с автомобилем. Она ставит вас в положение превосходства над автомобилем – и вы направляете его туда, куда нужно. Никакой пользователь не захочет, чтобы его допрашивали как одного из подозреваемых, – а ведь именно так нередко ведут себя наши программы.

Прячьте рычаги катапультирования.

В кабине каждого истребителя есть ярко окрашенный рычажок. Если за него потянуть, небольшой реактивный двигатель под сиденьем пилота сработает и вытолкнет пилота вместе с сиденьем из самолета, после чего раскроется парашют – и пилот благополучно спустится на землю. Рычажок катапульты можно использовать только один раз, а последствия его использования значительны и необратимы.

Катапульта в истребителе необходима, так же как средства настройки необходимы в сложных настольных приложениях. Превратности бизнеса и требования, предъявляемые к программным продуктам, заставляют адаптировать их к специфическим ситуациям – и мы должны обеспечить возможность такой адаптации. Фирмы, выкладывающие миллионы долларов за заказное программное обеспечение для бизнеса или за массовые лицензии на использование тысяч копий продукта,

будут недовольны, если программа окажется неспособной адаптироваться к их внутренним стандартам. Программа должна приспосабливаться, но эту процедуру следует рассматривать как однократное действие или в крайнем случае как действие, выполняемое редко и только квалифицированными сотрудниками фирмы. Иными словами, рычаг катапульты нужен, но используется он нечасто.

Программы должны иметь «катапульты», чтобы пользователи могли время от времени перемещать внутри интерфейса *стабильные объекты* (см. главу 11) или существенно (иногда необратимо) менять функциональность либо поведение приложения. Единственное, что никогда не должно случаться, – так это непреднамеренное катапультирование (рис. 10.8). При проектировании интерфейса следует убедиться, что пользователь не сможет нечаянно катапультироваться, когда ему требуется лишь слегка подкрутить настройки программы.

Рычаги «катапульты» бывают двух типов. Одни сильно изменяют внешний вид программы (расположение инструментов и рабочих областей), а другие выполняют какие-либо необратимые действия. Обе эти функции должны быть спрятаны от неопытных пользователей. Второй тип наиболее опасен. В первом случае пользователь может быть удивлен или напуган результатом, но он, по крайней мере, сможет вернуться к прежнему состоянию после некоторых усилий. Во втором случае и пользователю, и его коллегам придется смириться с последствиями.

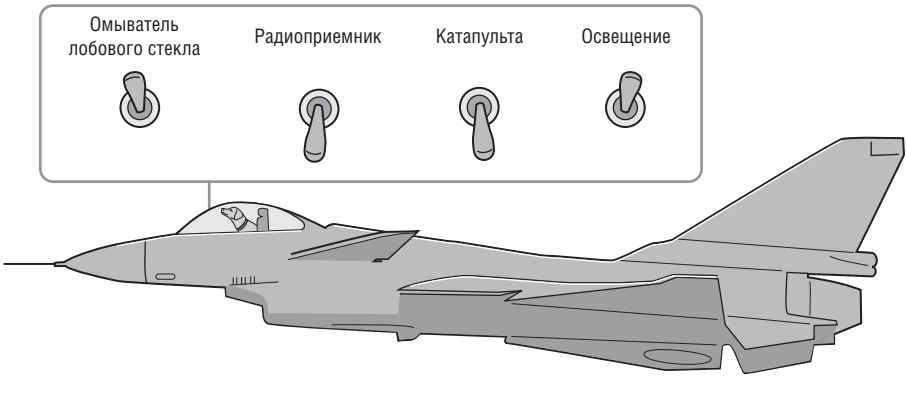


Рис. 10.8. *Рычаг катапульты способен вызвать катастрофические последствия. Еще минуту назад пилот удобно сидел в кресле самолета – и вот он уже кувыркается вверх тормашками в синем небе, а его самолет продолжает полет без него... Катапульта нужна для безопасности пилота, но конструкторы самолета делают все возможное для того, чтобы пилот не включил катапультирование по ошибке. Если разрешить ничего не подозревающему пользователю настраивать программу, меняя стабильные объекты интерфейса, это будет сравнимо со случайным включением катапульты. Прячьте рычаг катапульты!*

Если вы будете соблюдать принципы потока и оркестровки, ваша программа позволит пользователю увлеченно и максимально эффективно работать в течение долгого времени. Продуктивно работающий пользователь – это счастливый пользователь, а счастливые и эффективно работающие клиенты – цель любого производителя цифровых продуктов. В следующей главе мы продолжим обсуждение методов, позволяющих повысить продуктивность работы пользователей путем удаления ненужных барьеров, появляющихся вследствие проектирования по модели реализации.

ПРИНЦИП проектирования

Оптимизируйте скорость реакции; предупреждайте о задержках.

Обрабатывая большой объем информации или общаясь с удаленными серверами, принтерами, сетями, приложение может замедлить свою работу или перестать реагировать на действия пользователя. Нет ничего более опасного для состояния потока пользователя, чем необходимость смотреть на экран, ожидая ответа компьютера. Крайне важно проектировать интерфейсы таким образом, чтобы они достаточно быстро реагировали на команды пользователя – ни один «гламурный» интерфейс на свете не способен впечатлить пользователя, когда он работает со скоростью улитки из-за интенсивной отрисовки экрана.

Здесь важно сотрудничать с разработчиками. Различного рода взаимодействие может, в зависимости от платформы и технологической среды, иметь достаточно высокую стоимость с точки зрения скорости реакции приложения. Вы должны не только выступать за интерфейс, обеспечивающий пользователя насыщенным взаимодействием с минимальными задержками, но и создавать решения, учитывающие выбор технической платформы в начале проекта, который нельзя изменить. Когда задержки неизбежны, важно четко сообщить об этом пользователю и дать ему возможность отменить операцию, вызывающую задержку, а в идеале – еще и возможность делать другую работу во время ожидания.

Если приложение выполняет потенциально «долгоиграющие» задачи, не забывайте время от времени проверять, что делает пользователь. Возможно, он барабанит по клавиатуре или бешено щелкает мышью, завывая при этом: «Да нет же, не хотел я перестраивать *всю* базу данных. Это же займет 4,3 миллиона лет!»

В ряде исследований, проведенных еще в конце 60-х годов, ученые выяснили, что восприятие пользователем времени реакции можно грубо разделить на несколько интервалов (Miller, 1968):

- До **0,1 секунды** пользователи воспринимают отклик системы как моментальный. Они чувствуют, что напрямую манипулируют пользовательским интерфейсом и данными.

- До **1 секунды** пользователи чувствуют, что система **реагирует**. Вероятно, они замечают задержку, однако эта задержка недостаточно велика, чтобы прервать мыслительные процессы.
- До **10 секунд** пользователи замечают, что система работает медленно, и отвлекаются, однако способны сохранять некоторое внимание к приложению. Здесь важно наличие индикатора хода работы.
- **После 10 секунд** внимание пользователя полностью рассеивается. Он уходит за чашкой кофе или переключается на другое приложение. В идеале такие длительные процессы должны проводиться в фоновом режиме или без участия пользователя, позволяя ему заняться другой работой. В любом случае следует четко обозначать состояние и ход процесса, в том числе оставшееся время. И просто обязателен механизм отмены.

В общем, создание успешного продукта требует не только предоставления полезных функций. Следует задуматься также об оркестровке различных функциональных элементов, позволяющей пользователям достигать состояния потока при решении своих задач. Лучшие пользовательские интерфейсы не поражают пользователей своей красотой – скорее они почти незаметны, поскольку с ними крайне легко работать.

11

Оптимизация налогообложения

Программным продуктам часто свойственны взаимодействия, вынуждающие пользователя выполнять дополнительную работу. Программисты, как правило, настолько сосредоточены на технологиях, лежащих в основе продукта, что не особенно вникают в действия, которые требуются от человека, работающего с этими технологиями, чтобы он мог достичь своих целей. В результате получается приложение, взимающее с пользователя **налог** – в виде умственного, а иногда и физического напряжения во время работы.

Когда нам нужно поехать на работу, мы открываем гараж, садимся в машину, запускаем двигатель, выезжаем из гаража задним ходом и закрываем дверь гаража, прежде чем начать движение в сторону работы. Все эти действия нацелены собственно на автомобиль, а не на наше продвижение к месту назначения. Если бы у нас был транспортер, как в фильме «Звездный путь» (Star Trek), мы бы просто ввели нужные координаты и сразу оказались в нужном месте, – никаких вам гаражей, двигателей и светофоров. Нет, мы не жалуемся на особенности езды на автомобиле – мы пытаемся подчеркнуть разницу между двумя видами действий, которые совершает человек для выполнения повседневных задач.

Любая крупная задача, например поездка на работу, распадается на ряд подзадач. Некоторые из них направлены непосредственно на достижение цели, например управление автомобилем на дороге. **Налоговые задачи**, с другой стороны, не приближают нас к цели, но все равно необходимы для ее достижения. Примеры налоговых задач: отпирание и запирание двери гаража, запуск двигателя, остановки у светофоров, долив масла, заправка и прохождение техосмотра.

Налоги – это работа, удовлетворяющая потребности либо наших инструментов, либо внешних агентов, с которыми мы сталкиваемся, пытаясь достичь цели. Различия не всегда очевидны, так как мы привыкли

считать налоги частью наших задач. Многие из нас ездят в автомобиле так часто, что им трудно отделять действия по отпиранию гаража от действий по вождению автомобиля. Однако манипуляции с гаражной дверью нужны автомобилю, а не нам, и они не продвигают нас к цели в той же степени, как манипуляции с педалью газа и рулевым колесом. Остановка на красный свет – выдвигаемое обществом требование, которое также не способствует достижению цели. (Впрочем, оно способствует достижению родственной цели – прибытию к месту назначения *в целости и сохранности*.)

Когда речь заходит о программных продуктах, разделение подзадач на непосредственно ведущие к цели и налоговое бремя достаточно очевидно. Как и в примере с автомобилем, некоторые налоговые задачи тривиальны, и решать их не составляет труда. С другой стороны, бывают задачи такие же неприятные, как замена спустившего колеса. На ум приходят такие примеры налогов, как установка приложения, настройка сетей, создание резервных копий файлов.

Проблема с налоговыми задачами заключается в том, что усилия, направленные на их решение, не способствуют достижению цели. Когда нам удается сократить налоги, мы делаем работу пользователя более эффективной и продуктивной, а также повышаем пригодность нашей программы к использованию, создавая более качественный опыт взаимодействия с ней. В качестве проектировщика взаимодействия вы должны внимательно следить за появлением налогов и предпринимать шаги по их искоренению с энтузиазмом врача, борющегося с инфекцией.

Существование налогов в пользовательских интерфейсах вызывает справедливое раздражение пользователей цифровых продуктов. Любому проектировщику и менеджеру продукта следует отлавливать налоги взаимодействия во всех проявлениях и удалять их из своих продуктов.

Сокращайте налоговое бремя при любой возможности.

Налоги в графическом пользовательском интерфейсе

Одна из главных претензий, предъявляемых к графическим интерфейсам опытными пользователями – особенно теми, кто привык к системам с командной строкой, – заключается в том, что осмысленное манипулирование окнами и меню требует дополнительных усилий. Командная строка позволяет просто набрать команду, которую компьютер немедленно выполнит. В оконных системах пользователям приходится открывать различные папки в поисках файла или программы, прежде чем они смогут открыть файл или запустить программу на исполнение.

После появления окна программы на экране приходится растягивать его до нужного размера и перетаскивать в подходящее место.

Все эти претензии имеют под собой основания. Подобные манипуляции с окнами фактически являются налогом. Они не приближают пользователя к намеченной цели, а являются накладными расходами – требованием, выдвигаемым программой прежде, чем она снизойдет до оказания помощи пользователю. Однако каждый знает, что графические интерфейсы проще в использовании, чем системы с командной строкой. Так где же правда?

Противоречие возникает из-за того, что реальные механизмы взаимодействия остаются скрытыми. В интерфейсе с командной строкой налоговое бремя пользователей еще тяжелее: прежде чем начать работать с системой, требуется выучить команды. Кроме того, пользователь не может изменить экран по своему вкусу. Налоги командной строки снижаются только после того, как пользователь потратит много времени и сил на ее освоение.

Зато для неопытного, случайного пользователя визуальная ясность графического интерфейса оказывается подспорьем в навигации и изучении возможностей. Пошаговый способ общения пользователя с графическим интерфейсом помогает тем, кто еще не знаком с функциональностью системы. Он помогает в работе и тем, кто должен решать несколько задач одновременно и запускать сразу несколько приложений.

Налоги и опытные пользователи

Любой пользователь, желающий разобраться в интерфейсе командной строки, может быть автоматически отнесен к категории экспертов. А любой пользователь, разобравшийся в одном интерфейсе командной строки, быстро разберется в любом другом интерфейсе, включая графический. Эти пользователи без труда постигнут любые тонкости обращения с программой. Они запускают программу, имея четкое представление о том, что и как требуется сделать. Такому пользователю только мешает помощь, которую интерфейс предлагает новичку.

Снижая бремя налогов, мы должны быть очень внимательны. Не следует снижать его лишь в угоду опытным пользователям. С другой стороны, нельзя перекладывать все налоги на плечи опытных пользователей, создавая удобства для новичков.

Трехколесный велосипед

Поддержка новичков и пользователей, работающих с продуктом время от времени, – это та область, где проектировщикам приложений сложно избежать создания излишнего налогового бремени. Легко найти оправдание той функциональности, которая облегчает жизнь новичкам, изучающим программу. К сожалению, эта функциональность быстро превращается в дополнительную нагрузку, когда пользователь

переходит из категории новичков в категорию вечных середняков, что мы обсуждали в главе 3. Функциональность, добавленная к программе ради обучения пользователей (скажем, пошаговые сценарии), должна легко отключаться. Дополнительные колеса для велосипеда помогают учиться начинающим, но мешают тому, кто уже научился кататься.

Не приваривайте дополнительные колеса к велосипеду намертво.

«Наглые» налоги

Существуют действия, в которых не нуждается никто – ни новички, ни специалисты. Это и есть «наглые» налоги. Операции, связанные с настройкой аппаратной части, например указание программе, какой COM-порт она должна использовать, компьютер мог бы выполнить и самостоятельно. Подобные аспекты следует убирать из пользовательского интерфейса и заменять интеллектуальным поведением программы, скрытым от пользователя.

Визуальные налоги

Визуальный налог – это работа по расшифровке видимой информации, которую приходится выполнять пользователю. Поиск конкретного элемента в списке, выяснение того, откуда начинать чтение текста на экране или какие элементы являются активными, а какие служат просто украшениями, – все это визуальные налоги.

Проектировщики иногда сами загоняют себя в угол, слишком сильно полагаясь на визуальные метафоры. Речь идет о таких метафорах, как телефоны, копиры, степлеры и факсы на рабочих столах или картотечные шкафы с папками в ящиках. Эти визуальные метафоры позволяют быстро понять связь между элементами интерфейса программы и ее поведением, однако, когда пользователь поймет эти основы, работа с метафорами окажется «наглым» налогом (подробное обсуждение недостатков визуальных метафор проводится в главе 13). Кроме того, место на экране, отведенное под изображения, расходуется крайне нерационально, особенно в монопольных приложениях (типы приложений мы подробно обсуждали в главе 9). Чем дольше мы видим окно программы изо дня в день, тем больше мы негодуем по поводу огромного количества пикселов, расходуемых лишь на то, чтобы сообщить уже известные нам сведения. Небольшое изображение телефона, которое помогло нам набрать номер в первый день пользования, теперь превратилось в препятствие, затрудняющее быструю связь.

Пользователям временных приложений часто требуются некоторые инструкции, чтобы эффективно использовать продукт. Расходование экранного пространства на эти цели во временном приложении, как правило, не имеет столь губительных последствий, как в монопольном.

Временные приложения используются нечасто, поэтому пользователи нуждаются в дополнительном обучении тому, что делает приложение и как им управлять. Зато в монопольном приложении малейший дополнительный налог со временем становится невыносимым. Другой значимый источник визуальных налогов – расточительная стилизация графики и элементов интерфейса (рис. 11.1) Визуальный стиль в первую очередь должен поддерживать прозрачную передачу информации и демонстрацию состояния интерфейса.

В некоторых приложениях украшения могут быть уместны для создания определенного настроения, атмосферы или придания продукту индивидуальности. Однако излишнее украшательство может снизить эффективность работы пользователей, принуждая их расшифровывать различные визуальные элементы, чтобы понять, что является элементами управления и важной информацией, а что служит просто украшениями. Более подробно о поиске равновесия в вопросах создания удобного визуального дизайна интерфейсов читайте в главе 14.

Рис. 11.1. *Стартовая страница Disney.com – замечательный пример визуальных налогов. Текст подвергся сильной стилизации и не следует композиционной сетке. Пользователям тяжело различать украшения и элементы навигации. Это требует от них дополнительной мыслительной работы при работе с сайтом. Не всегда это плохо: правильно подобранный объем такой работы может стать хорошим источником развлечения – как, скажем, в головоломках*

Как выявить налоги

Некоторые функции могут оказаться нужными случайному пользователю или пользователю с необычными предпочтениями. Такие функции можно считать налогами только в том случае, если пользователь вынужден выполнять их, не имея альтернативы. Пример такого рода функций – управление размерами и положением окон. Единственный способ определить, является ли та или иная функция налогом, – соотнести ее с целями персонажа. Если важному персонажу необходимо видеть на экране сразу два приложения, чтобы сравнивать или переносить информацию, возможность изменить размеры и расположение их главных окон не является налогом. Если же у персонажа нет такой потребности, необходимость настройки окна любой из этих программ есть не что иное, как налог.

Прекращение работы

Существует особая разновидность налогов, столь распространенная, что заслуживает отдельного разговора. В предыдущей главе мы ввели понятие **потока** – высокопроизводительного состояния человека, в котором он исключительно продуктивно и гармонично применяет свои рабочие инструменты. Состояние потока является естественным, и люди достигают его без внешнего толчка. Требуются определенные усилия, чтобы вывести пользователя из этого состояния. Вмешательство в поток – телефонный звонок или сообщение об ошибке – вполне способно на это. Некоторые из подобных раздражителей неизбежны, но другие вовсе не обязательны. Прерывание потока без веской причины является *прерыванием работы из-за ерунды*. Это одна из самых разрушительных форм налогового бремени.

Не прерывайте работу из-за ерунды.

Неудачно спроектированный программный продукт позволяет себе делать предположения, которые никогда не сделал бы уважающий себя человек. Например, такой продукт безапелляционно сообщает, что файл не существует, поскольку оказывается недостаточно интеллектуальным, чтобы поискать файл в подходящем месте. При этом он неявно обвиняет в потере файла *вас*! Бывает, что программа с готовностью выполняет команды пользователя, «подвешивающие» систему так, что требуется перезагрузка компьютера. Пользователи совершенно справедливо расценивают подобное поведение программных продуктов как идиотизм.

Сообщения об ошибках, уведомления и запросы на подтверждение операций

Пожалуй, самой распространенной формой налогов являются сообщения об ошибках и диалоговые окна с требованием подтвердить операцию. Они вездесущи до такой степени, что их искоренение требует большого труда. В главе 25 мы подробно обсудим этот вопрос, а пока достаточно сказать, что эти элементы серьезно обременяют пользователя, и вы должны исключать их из своих приложений, где только возможно.

Типичное диалоговое окно с сообщением об ошибке является абсолютно необязательным. Оно либо сообщает пользователю информацию, до которой ему нет дела, либо требует, чтобы он исправил ситуацию, которую программа обычно может и должна исправить самостоятельно. На рис. 11.2 показано сообщение об ошибке, выводимое приложением Adobe Illustrator 6, когда пользователь пытается сохранить документ. Не совсем понятно, что пытается сказать приложение, но звучит устрашающе.

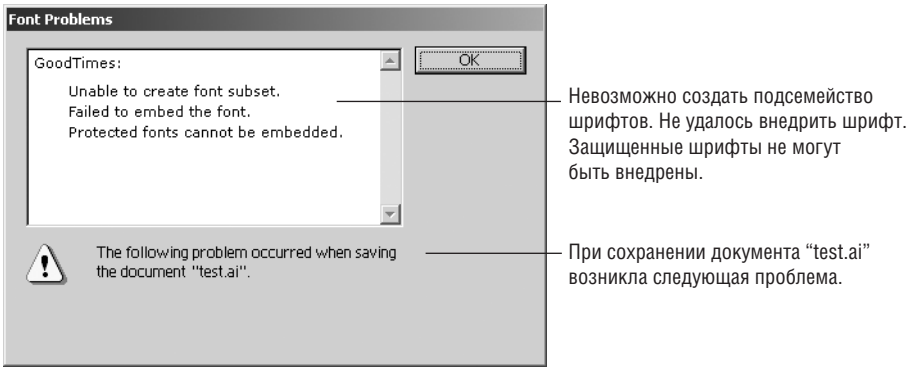


Рис. 11.2. *Отвратительное и бесполезное сообщение об ошибке, прерывающее работу из-за ерунды. Невозможно ни понять, ни проверить то, о чем оно сообщает, и не остается ничего, кроме как признать вину нажатием на кнопку* OK. *Это сообщение всплывает только при сохранении документа приложением, то есть именно тогда, когда вы доверили программе сделать что-то очень простое и бесхитростное. Это приложение без помощи пользователя не может даже сохранить файл, но не сообщает, какая именно помощь требуется*

Это сообщение прерывает и растягивает и без того длительную и неприятную процедуру. Пользователь не может позволить себе выпить чашечку кофе после того, как выдаст программе команду сохранить результат своего труда, потому что, вернувшись к компьютеру, рискует обнаружить, что операция не завершена, а программа бездумно приостановила процесс. В главе 25 мы покажем, как избегать подобных сообщений.

А вот еще один вариант раздражающего поведения – на этот раз от Microsoft Outlook.

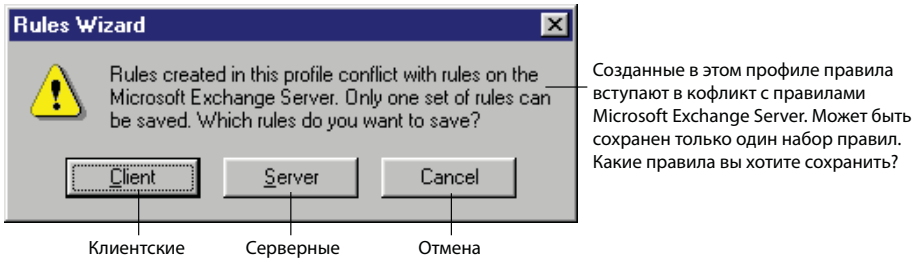


Рис. 11.3. *Еще один ужасный запрос на подтверждение, прекращающий работу из-за ерунды. Если приложение достаточно разумно, чтобы понять разницу, почему оно не может самостоятельно исправить проблему? Предлагаемые варианты действий пугают. Диалоговое окно сообщает, что можно взорвать одну из двух коробок. В одной мусор, а в другой ваша собака – только приложение не сообщает, что где. А если нажать кнопку Cancel (Отмена), что это будет означать? Вдруг приложение все же решит взорвать вашу собаку?*

Это диалоговое окно предлагает нам решиться на необратимое и, возможно, дорогостоящее действие, не располагая при этом никакой информацией! Если оно появляется после того, как вы изменили какие-то правила, разве не разумно предположить, что вы хотите придерживаться новых правил? А если предполагается, что вы не хотите этого, то почему бы не сообщить вам дополнительную информацию о том, какие именно правила конфликтуют и какие были приняты последними? Не понимая, что происходит, пользователь щелкнет по кнопке Cancel... Он отменяет диалоговое окно и оставляет конфликт правил неразрешенным? Он отменяет последние изменения, которые привели к конфликту? Этот плохо продуманный интерфейс вызывает у пользователя страх и ощущение неопределенности, что абсолютно не нужно. В главе 24 мы обсудим, как улучшить ситуацию.

Пользователи, просящие разрешение

Во времена интерфейсов командной строки и текстовых меню интерфейсы часто выполняли команды пользователя не напрямую. Если пользователь хотел, например, изменить свой адрес, ему вначале приходилось просить на это разрешение. Программа выводила экран, где можно было изменить адрес. Такой подход – «наглый» налог на использование программы. К сожалению, сегодня ситуация не улучшилась: чтобы изменить один из своих сохраненных адресов на Amazon.com, нужно щелкнуть по кнопке и отправиться на отдельную страницу. Если нужно изменить отображаемое значение, должна быть возможность изменить его здесь и сейчас. Не следует заставлять пользователя просить разрешение или вести его в другую комнату.

Не заставляйте пользователей просить разрешения.

Подобно ситуации из последнего примера многие приложения отделяют *отображение* значений (таких как имена файлов, числа и выбранные режимы) от *редактирования* этих значений. Это модель реализации, в которой ввод и вывод считаются различными процессами. Но в ментальной модели пользователей это различие отсутствует. Человек думает: «Вот число. Я щелкну по нему и наберу новое значение». Если приложению нечего ответить на такое желание пользователя, это означает, что интерфейс содержит налог. Если настройки могут изменяться пользователем, он должен иметь возможность изменить их там же, где они отображаются приложением.

Разрешайте ввод везде, где имеется вывод.

Действие, противоположное обращению за разрешением, может оказаться актуальным в определенных обстоятельствах. Пользователь не просит у программы разрешения на открытие диалогового окна, а, наоборот, сообщает ей, что некоторое диалоговое окно следует закрыть и больше никогда не открывать. Так он сможет избавить себя от домогательств со стороны бесполезного диалогового окна, которое программа ошибочно считает полезным. Компания Microsoft в последнее время активно пользуется этой идиомой. (Если новичок нечаянно откажется от диалогового окна и будет не в состоянии вернуть его, он сможет прибегнуть к другой идиоме – пункту меню Справка, где сказано что-то вроде «Вернуть назад все окна, от которых пользователь отказался».)

Распространенные налоговые ловушки

Вы должны неусыпно следить за появлением малейших признаков дополнительных налогов в интерфейсе и изничтожать их на корню. Мириады мелких необязательных операций заставляют пользователя выполнять гигантский объем лишней работы. Следующий список поможет вам выявлять налоги:

- Не заставляйте пользователя переходить к другому окну ради выполнения операции, влияющей на текущее окно.
- Не заставляйте пользователя вспоминать, где в файловой иерархии расположены его файлы.
- Не заставляйте пользователя изменять размер окна без необходимости. Когда на экране появляется дочернее окно, программа должна установить его размер в соответствии с его содержимым. Не создавайте ни большое пустое окно, ни маленькое и требующее прокрутки.
- Не заставляйте пользователя передвигать окна. Если на рабочем столе есть свободное место, открывайте окно программы там, а не поверх других окон.

- Не заставляйте пользователя заново вводить предпочтения. Если он установил шрифт, цвет, отступ или громкость, сделайте так, чтобы ему не пришлось повторно выполнять настройку, пока он сам не захочет.
- Не заставляйте пользователя вводить данные во все поля в соответствии с неким произвольным критерием полноты. Если пользователь хочет опустить какие-то детали при вводе данных в форму, не вынуждайте его вводить их. Исходите из предположения, что у пользователя есть веские причины скрывать информацию. Полнота базы данных (в большинстве случаев) не стоит того, чтобы нервировать пользователя.
- Не заставляйте пользователя просить разрешения. Как правило, это признак того, что ввод и вывод отделены друг от друга.
- Не просите пользователя подтверждать свои действия (это правило подразумевает наличие развитой функции отмены).
- Не позволяйте действиям пользователя приводить к ошибке.

Навигация как налог

Самый важный факт, который следует уяснить в отношении навигации, заключается в том, что навигация в основном является налогом. За исключением компьютерных игр, где успешное ориентирование в лабиринте препятствий является *целью*, навигация в программном продукте или на веб-сайте редко отвечает потребностям, целям и желаниям пользователя. (Хотя следует особо отметить ситуацию, когда качественно спроектированная навигация может быть эффективным способом рассказать пользователям о возможностях продукта, что определенно лучше соответствует их целям.)

Необязательная или затрудненная навигация становится серьезным источником раздражения для пользователей. Авторы этих строк считают, что плохо спроектированная навигация представляет собой *проблему номер один* при разработке любого интерактивного продукта – будь то приложение для работы на настольном компьютере, веб-приложение или что-то иное. Навигация – та составляющая программы, где лучше всего видна модель реализации, использованная программистом.

Навигация в программах происходит на нескольких уровнях:

- между различными окнами, представлениями или страницами;
- между панелями или фреймами внутри окна, представления или страницы;
- между инструментами, командами или меню;
- по информации, отображаемой внутри панели или фрейма (прокрутка, панорамирование, масштабирование, переход по ссылкам).

У читателя может возникнуть недоумение по поводу некоторых пунктов этого списка. Мы сознательно придерживаемся расширенного определения навигации как *любого действия, которое переносит пользователя в новую область интерфейса или требует от него поиска объектов, инструментов либо данных*. Причина этого проста: эти действия требуют, чтобы люди понимали, где находятся в рамках интерактивной системы и как найти и активировать нужную функцию. Когда мы задумываемся об этих действиях как о навигации, становится ясно, что они являются налогом и, следовательно, должны быть исключены или минимизированы. Обсудим каждый из этих типов навигации более подробно.

Навигация между экранами, представлениями или страницами

Навигация между несколькими представлениями внутри приложения или веб-страницами является, пожалуй, самым дезориентирующим видом навигации. Навигация между окнами требует переключения внимания, разрушает состояние потока, в котором находится пользователь, и принуждает его перейти в новый контекст. Действие перехода к новому окну нередко приводит к тому, что содержимое исходного окна частично или полностью перекрывается. Пользователю приходится, как минимум, беспокоиться об управлении окнами, а это налог, еще больше разрушающий состояние потока. Если пользователям приходится постоянно переходить от одного окна к другому, продуктивность их работы падает, а степень дезориентации и раздражения повышается. Они отвлекаются от текущих задач. Когда количество окон достигает некоторого предела, пользователь становится настолько дезориентированным, что ему грозит **навигационная травма**: он чувствует себя заблудившимся внутри интерфейса. В монопольных приложениях возникновения этой проблемы можно избежать, поместив основные взаимодействия в одно главное представление, содержащее несколько независимых панелей.

Навигация между панелями

Окно может содержать некоторое количество панелей, соприкасающихся и разграниченных разделителями (см. главы 19 и 20) либо наложенных друг на друга и обозначенных вкладками. Смежные панели могут решить многие навигационные проблемы, поскольку содержат полезные вспомогательные функции, ссылки или данные, имеющие непосредственное отношение к основной рабочей области, что сводит навигацию практически к нулю. Если объекты могут быть перенесены с одной панели на другую, эти панели должны быть расположены по соседству.

Проблемы возникают, когда смежных панелей становится слишком много или когда их расположение на экране не соответствует логиче-

ской последовательности работы пользователя. Большое количество смежных панелей приводит к визуальной засоренности интерфейса и путанице. Пользователь не знает, где искать необходимое. Кроме того, переполнение окна панелями вызывает необходимость в прокрутке – а это еще одна навигационная проблема. Тем самым затрудняется навигация в пределах одного окна. Некоторые веб-порталы, пытаясь угодить всем и каждому, создают подобные проблемы.

В ряде случаях, в зависимости от решаемых пользователем задач, уместны панели, организованные в виде вкладок. Такая организация создает определенную дополнительную нагрузку и способна дезориентировать пользователя, потому что вкладка, на которую он перешел, закрывает предыдущую. Впрочем, эта идиома уместна в рабочей области, где пользователь имеет дело с несколькими документами или разными внешними представлениями одного документа (как, например, в Microsoft Excel, рис. 11.4).

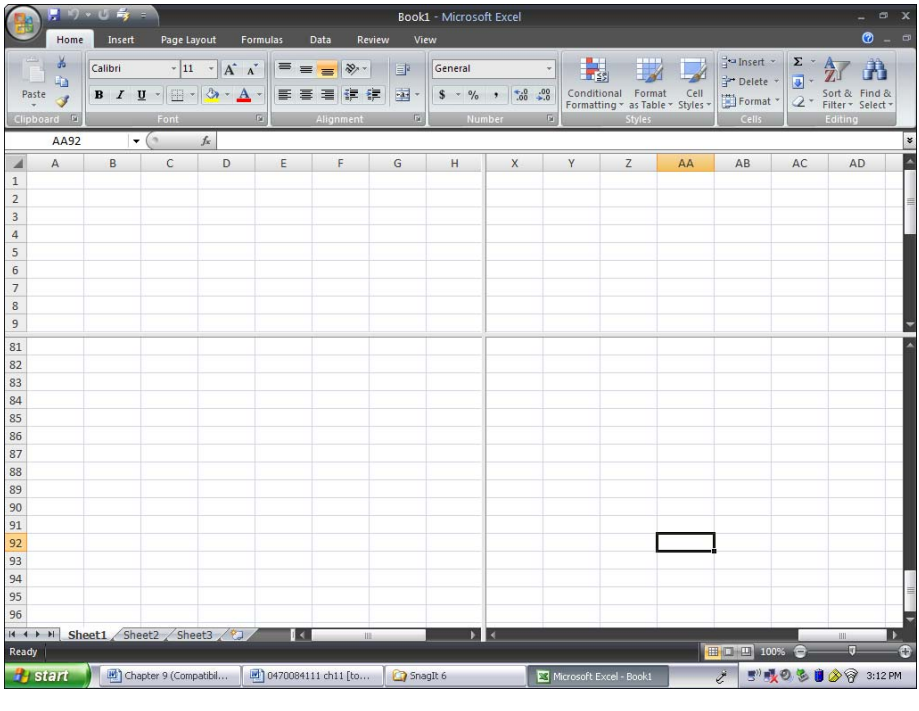


Рис. 11.4. *Microsoft Excel применяет панели внутри вкладок (ярлычки которых видны слева внизу экрана), чтобы дать возможность пользователю переключаться между электронными таблицами. В Excel применяются также разделители, так что пользователь может видеть различные части одного документа без постоянной прокрутки туда-обратно. Обе эти идиомы снижают для пользователей Excel дополнительные навигационные налоги*

Некоторые программисты применяют вкладки для разбиения сложной функциональности на фрагменты. Они полагают, что пользоваться продуктом проще, если он разрезан на части. На самом деле разбиение функциональности и размещение частей на разных панелях увеличивает налоговое бремя и затрудняет ориентацию и понимание продукта пользователем.

Вкладки – это способ экономить место на экране, применяемый, когда требуется втиснуть всю информацию и функции в ограниченное пространство. Классический пример – диалоговое окно настройки. Не думаем, что кому-то захочется увидеть разом все настройки сложного приложения. Однако в большинстве случаев применение вкладок создает заметные навигационные налоги. Редко когда удается подобрать достаточно точное название вкладки для панели, с которой она связана, и пользователям приходится просматривать все вкладки в поисках нужной информации или функции.

Вкладки могут быть уместными при наличии нескольких панелей, не используемых одновременно и поддерживающих происходящее в основной рабочей области. Эти вспомогательные панели можно сложить в стопку, и пользователь будет выбирать нужное одним щелчком. Классический пример – микшер цвета и область цветовых образцов в Adobe Illustrator (рис. 11.5) Эти два инструмента предлагают взаимоисключающие способы выбирать цвет для рисования, и пользователь, как правило, знает, какой способ подходит для текущей задачи.

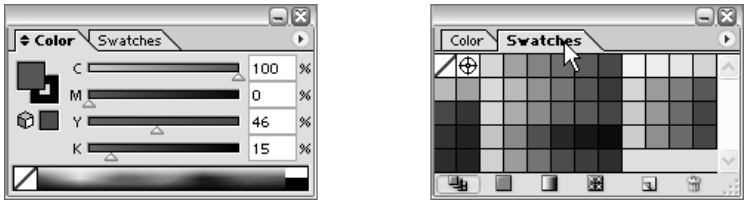


Рис. 11.5. *Вкладки палитр в Adobe Illustrator позволяют пользователям переключаться между микшером и образцами – двумя альтернативными механизмами выбора цвета*

Навигация между инструментами и меню

Еще одна важная и не осознаваемая разработчиками форма навигации возникает из необходимости пользоваться различными инструментами, палитрами и функциями. Оптимальная пространственная организация соответствующих элементов интерфейса в пределах панели или окна очень важна для минимизации лишних движений мыши, которые в лучшем случае вызывают у пользователя раздражение и усталость, а в худшем – профессиональные заболевания. Инструменты, применяемые часто и в сочетании с другими инструментами, следует группировать и делать доступными мгновенно. Работа с меню требует

от пользователя бóльших усилий по навигации, поскольку содержимое меню невидимо, пока меню не открыто. Часто применяемые функции должны быть доступны через панели инструментов, палитры или аналогичные элементы интерфейса. Меню должны использоваться только для выполнения команд, к которым пользователь обращается нечасто (мы еще вернемся к обсуждению организации элементов интерфейса в этой главе, а панели инструментов обсудим более подробно в главе 23).

Adobe Photoshop 6.0 ведет себя нежелательным образом, когда заставляет пользователей выполнять навигацию между элементами на палитрах. Скажем, инструменты Заливка (Paint Bucket) и Градиент (Gradient) занимают одну ячейку на палитре инструментов. Чтобы выбрать один из них, вы должны щелкнуть по видимому элементу управления и дождаться появления меню, которое позволит активизировать один из инструментов (рис. 11.6). И это притом, что оба инструмента служат для заливки цветом и используются достаточно часто. Было бы разумнее поместить их рядом на палитре, чтобы избежать подобной навигации, разрушающей состояние потока пользователя.

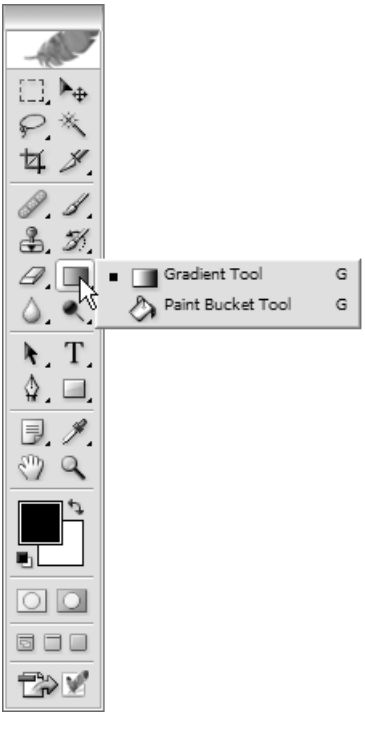


Рис. 11.6. *Adobe Photoshop прячет инструмент Заливка в комбо-кнопке (см. главу 21) на палитре инструментов. Несмотря на то, что пользователям часто требуется как Градиент, так и Заливка, им приходится обращаться к этому меню каждый раз, когда нужно переключиться с одного инструмента на другой*

Навигация по информации

Навигация в информационном содержимом окна или панели осуществляется несколькими методами: прокруткой (панорамированием), переходами по гиперссылкам и масштабированием. Первые два метода широко распространены: прокрутка встречается практически в каждой программе, а переход по ссылкам – практически на каждой веб-странице (впрочем, идиомы гиперссылок все чаще используются и в приложениях, не связанных с Всемирной паутиной). Изменение масштаба применяется преимущественно для визуализации объемных и детализированных плоских изображений.

Прокрутка необходима часто, но потребность в ней следует минимизировать. Как правило, удается найти компромисс между разбивкой на страницы и прокруткой информации. Вы должны понять ментальные модели ваших пользователей и последовательность действий в их работе, чтобы выбрать подходящий вариант.

В двухмерных графических программах вертикальная и горизонтальная прокрутка в порядке вещей. Навигация еще больше упрощается при наличии мини-карты. Эта техника обсуждается ниже в данной главе наряду с другими визуальными указателями.

Гиперссылки – очень важная навигационная парадигма в среде Всемирной паутины. Поскольку она изменяет внешний вид экрана, разработчик должен позаботиться о визуальных и текстовых навигационных ориентирах для пользователя.

Масштабирование и **панорамирование** – это навигационные инструменты для работы с двухмерными и трехмерными изображениями. Они уместны при создании двухмерных и трехмерных сцен и моделей и при просмотре трехмерных моделей реального пространства (например, в виртуальных экскурсиях). Они, как правило, не способны оказать помощь при исследовании случайных или абстрактных данных, представленных более чем в двух измерениях. В некоторых программах визуализации масштабирование означает более детализированный показ объекта, то есть имеет скорее логический, чем пространственный характер. По мере увеличения объекта его атрибуты (нередко текстовые) появляются поверх изображения. Такой вид взаимодействия обычно дает хорошие результаты, если реализован через смежную вспомогательную панель, представляющую свойства выделенных объектов в стандартизированной хорошо читаемой форме. Пользователям трудно разобраться в пространственном масштабировании; логическое масштабирование является загадочным для всех, кроме профессионалов в сфере визуализации и редких программистов.

Панорамирование и масштабирование, а тем более их сочетание создают огромные навигационные проблемы для пользователей. И хотя ситуация улучшается с развитием доступных через Интернет карт, пользователю по-прежнему легко заблудиться в виртуальной реальности.

Люди не привыкли перемещаться в неограниченном трехмерном пространстве, и они с трудом воспринимают трехмерное пространство, спроецированное на двухмерный экран (более подробно о трехмерном манипулировании мы поговорим в главе 19).

Улучшение навигации

Существует много способов улучшения (исключения, сокращения, ускорения) навигации по вашим программам, веб-сайтам и электронным устройствам. Вот наиболее эффективные:

- уменьшение количества пунктов назначения;
- создание «дорожных указателей»;
- организация обзора;
- ассоциирование элементов управления с функциями;
- адаптация интерфейса к нуждам пользователя;
- отказ от иерархических структур.

Обсудим эти методы более подробно.

Уменьшение количества пунктов назначения

Самый эффективный метод улучшения навигации очевиден: сократите число пунктов назначения при работе с продуктом. Под «пунктами назначения» здесь понимаются рабочие режимы, формы, диалоговые окна, страницы, окна и экраны. Когда количество режимов, страниц или экранов минимально, возможности пользователя по ориентированию в программе значительно возрастают. В терминах четырех типов навигации, перечисленных ранее, данная директива означает следующее:

- Сведите количество окон и представлений к минимуму. Одно полноэкранное окно с двумя, максимум тремя представлениями – оптимальный вариант для многих пользователей. Сведите к минимуму количество диалоговых окон, особенно немодальных. Программы и веб-сайты с десятками различных типов страниц, экранов и форм сложны для навигации.
- Сократите количество смежных панелей в окне или на веб-странице до минимального количества, необходимого пользователю для достижения своих целей. Разумный максимум для монопольных приложений – не более трех панелей, однако здесь нет жестких правил: многим приложениям требуется больше панелей. На веб-страницах все, что сложнее двух областей навигации и одной области с данными, вызывает у пользователей излишнее напряжение.
- Сведите количество элементов управления к тому минимуму, который позволяет пользователям достигать своих целей. Понимание пользователей, которое дают персонажи, позволит вам исключить

функции и элементы управления, о которые пользователи будут спотыкаться ввиду их ненужности или нежелательности.

- Минимизируйте прокрутку везде, где это возможно. Иными словами, создавайте вспомогательные панели такого размера, чтобы информация, которую они содержат, не нуждалась в постоянной прокрутке. Двухмерные и трехмерные графики и сцены должны по умолчанию иметь такой размер, чтобы пользователю не приходилось постоянно прибегать к панорамированию. Масштабирование, особенно когда оно выполняется неоднократно, является для большинства пользователей самым трудным типом навигации, так что его применение должно быть вызвано желанием пользователя, а не суровой необходимостью.

Многие интернет-магазины имеют запутанную навигацию, поскольку проектировщики пытаются услужить всем покупателям в рамках одного сайта. Если посетитель покупает книги, но не интересуется компакт-дисками, ссылка на страницу с компакт-дисками не должна бросаться ему в глаза. Предоставьте ему больше площади для покупки книг – и навигация упростится. С другой стороны, если пользователь посещает страницу со своей учетной записью часто, кнопка (или вкладка), предоставляющая доступ к его учетной записи, в его версии сайта должна быть заметной.

Создание «дорожных указателей»

Помимо сокращения числа пунктов назначения можно дополнительно упростить пользователям ориентирование при помощи более качественных подсказок – **дорожных указателей**. Подобно морякам, ориентирующимся по звездам и береговой линии, пользователи ориентируются по **стабильным объектам**, встроенным в интерфейс.

Во вселенной рабочего стола стабильными объектами являются окна программ. Каждая программа, как правило, имеет главное окно (окно верхнего уровня). Самые заметные особенности главного окна тоже могут считаться стабильными объектами – это строки меню, панели инструментов и прочие палитры и визуальные элементы, такие как строки состояния и линейки. Вообще говоря, каждое окно интерфейса имеет свои отличительные черты и быстро становится узнаваемым.

В среде Всемирной паутины действуют аналогичные правила. В качественно спроектированные веб-сайты тщательно внедрены стабильные объекты, которые не изменяются в процессе совершения покупок. Это, в частности, меню навигации вверху страницы. Такие области не только четко передают навигационные возможности, но и своим постоянным присутствием на экране помогают пользователям ориентироваться (рис. 11.7).

В электронных устройствах аналогичные правила действуют в отношении экранов, однако здесь роль дорожных указателей могут выполнять аппаратные элементы управления – особенно с учетом того, что

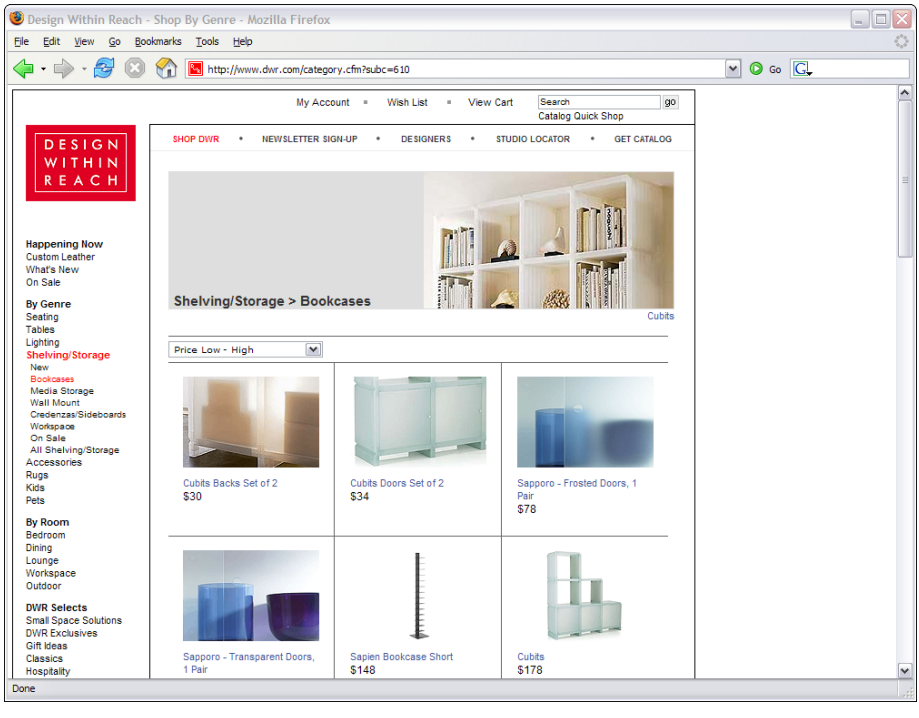


Рис. 11.7. *На сайте Design Within Reach многочисленные стабильные области задействованы на большинстве страниц. К ним относятся гиперссылки и поле поиска у верхнего края страницы, а также инструменты навигации в боковых колонках. Они помогают пользователям понять, куда можно отправиться дальше, а также информируют о текущем положении*

они могут визуально и тактильно сообщать пользователю о своем состоянии. Например, кнопки управления автомагнитолой, подсвечивающиеся при нажатии, и даже положения стрелки на циферблате дают навигационную информацию при надлежащей поддержке со стороны программного обеспечения.

В зависимости от приложения содержимое главного окна также может быть легко узнаваемым (особенно когда речь идет о киосках и приборах с маленьким дисплеем). Некоторые программы предлагают несколько способов представления данных, так что общий вид экрана может меняться в зависимости от выбранного представления. Однако отличительные черты настольного приложения обычно определяются уникальным сочетанием меню, палитр и панелей инструментов. Отсюда следует, что меню и панели инструментов нужно считать вспомогательными средствами навигации. Для успешной навигации не требуется большого количества дорожных указателей. Достаточно, чтобы они были хорошо видны. Нет нужды говорить, что указатели, которые

вдруг пропадают, не способствуют навигации. Это означает, что они должны быть постоянной принадлежностью интерфейса.

Стремление сделать все страницы веб-сайта похожими друг на друга продиктовано маркетинговыми соображениями, однако, если зайти слишком далеко, пользователей это может и дезориентировать. Ясно, что общие элементы страниц должны выглядеть и располагаться единообразно. Тем не менее если разделы сайта будут визуально различаться, это поможет пользователям ориентироваться.

Меню

Самым заметным объектом программы является главное окно с заголовком и строкой меню. Качество меню определяется его надежностью и непротиворечивостью. Неожиданное изменение меню может кардинально подорвать доверие к нему пользователя. Это справедливо в отношении как меню в целом, так и его отдельных пунктов.

Панели инструментов

Если в приложении есть панель инструментов, ее следует рассматривать как узнаваемый дорожный указатель. Поскольку панели инструментов являются идиомами для вечных середняков, а не для новичков, правила о недопустимости изменения внешнего содержания, столь строгие в отношении пунктов меню, в отношении панелей инструментов смягчаются. Удаление самой панели, конечно, дезориентирует, поскольку она является стабильным объектом. Хотя возможность сокрытия панели должна быть предусмотрена, не следует убирать ее неожиданно, при этом пользователь должен быть защищен от случайного удаления панели из окна. В некоторых программах кнопки, скрывающие панель инструментов, расположены на самой панели! Это совершенно недопустимое размещение рычага катапульты.

Другие дорожные указатели в интерфейсе

Палитры инструментов и закрепленные области экрана, в которых выводится или редактируется информация, тоже следует отнести к стабильным объектам, облегчающим навигацию по интерфейсу. Разумное использование свободного пространства и хорошо читаемых шрифтов необходимо для того, чтобы эти дорожные указатели были заметны и понятны.

Организация обзора

Обзоры играют в интерфейсе ту же роль, что и дорожные указатели: они способствуют ориентированию пользователей. Разница между ними заключается в том, что обзоры помогают пользователю ориентироваться внутри информационного содержания, а не в приложении как таковом. Поэтому сама область с обзором должна быть стабильным объектом; содержание же ее зависит от просматриваемых данных.

Рис. 11.8. *В своем продукте Photoshop компания Adobe применяет прекрасную идиому обзора – панель Navigator (слева), которая содержит миниатюру изображения и с помощью рамки выделяет на ней ту часть, которая в данный момент видна внутри рабочей области. Эта панель не просто создает контекст навигации – ее можно применять для панорамирования и масштабирования главного изображения. Аналогичная идиома представлена справа: в приложении для бизнес-графики Google Finance небольшая диаграмма внизу экрана представляет общий вид и контекст для увеличенного изображения, размещенного вверху экрана*

В зависимости от характера информации обзоры могут быть графическими или текстовыми. Блестящим примером графического обзора является панель с уместным названием Navigator в Adobe Photoshop (рис. 11.8).

Во Всемирной паутине самой распространенной формой обзора является текст, а именно – вездесущие «хлебные крошки» (рис. 11.9). Они не только показывают пользователю, в какой точке иерархической структуры данных он находится, но и снабжают его ссылками, позволяющими переходить к другим узлам этой структуры. Эта идиома до некоторой степени утратила популярность, когда веб-сайты стали отходить от иерархической организации в пользу ассоциативной организации, которая не слишком хорошо уживается с «хлебными крошками».

Интересным инструментом обзора является **аннотированная полоса прокрутки**. Такие полосы особенно уместны при прокрутке текста. Они грамотно используют тот факт, что как полосы прокрутки, так и текстовая информация линейны по своей природе. Это позволяет выводить

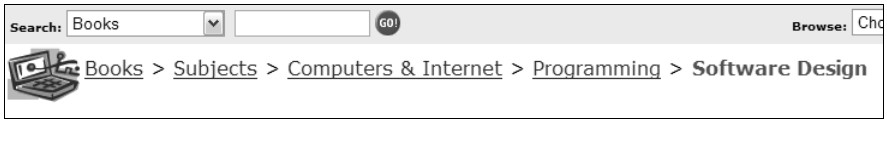


Рис. 11.9. *Типичные «хлебные крошки» на Amazon.com. Пользователь видит, где он уже побывал, и может перейти по любой ссылке на этом пути*

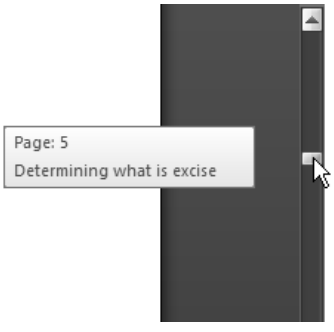


Рис. 11.10. *Аннотированная полоса прокрутки в Microsoft Word 2007 дает пользователю полезный контекст в процессе пролистывания документа*

на экран информацию о местонахождении выделенных областей и, возможно, других атрибутов форматированного или неформатированного текста. Подсказки о том, где находятся эти элементы, появляются в соответствующие моменты на траектории движения ползунка прокрутки. Когда происходит прокрутка, то аннотированный атрибут текста выводится на экран (рис. 11.10). Вариант аннотированной полосы прокрутки присутствует в редакторе Microsoft Word: активирующаяся в процессе прокрутки подсказка содержит номер страницы и ближайший заголовок.

Ассоциирование элементов управления с функциями

Ассоциирование описывает отношение между элементом управления, объектом воздействия и получаемым результатом. Некачественное ассоциирование проявляет себя в том, что элемент управления не соотносится ни визуально, ни символически с объектом, на который он действует. Некачественная ассоциация заставляет пользователя остановиться и задуматься о соотношениях, а это выводит его из состояния потока. Плохое ассоциирование элементов управления с функциями увеличивает когнитивную нагрузку на пользователя и чревато серьезными ошибками.

Хороший пример проблем, связанных с ассоциированием, мы наблюдаем в интерфейсе обычной кухонной плиты, далекой от мира компьютерных технологий. Практически каждый, кому приходилось готовить, испытывал раздражение от неподходящего ассоциирования ручек кухонной плиты с горелками, которые они открывают. У типичной плиты, изображенной на рис. 11.11, горелки расположены по углам квадрата. Однако ручки для этих горелок расположены в ряд на передней панели плиты.

В этом случае мы имеем дело с проблемой **физического ассоциирования**. *Результат* использования элемента управления достаточно очевиден: горелка включится, когда вы повернете ручку. Однако неясен

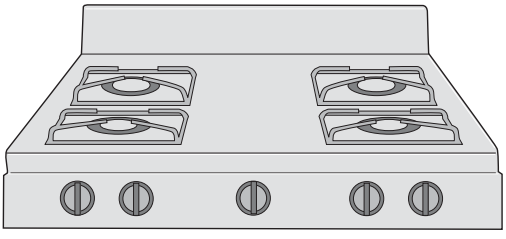


Рис. 11.11. *Кухонная плита с плохим физическим ассоциированием элементов управления. Какой горелкой управляет крайняя левая ручка – левой передней или левой задней? Тому, кто пользуется плитой, приходится каждый раз заново искать ассоциацию*

целевой объект элемента управления: какая именно горелка нагреется? К какой горелке относится крайняя левая ручка – левой передней или левой задней? Пользователи должны выяснять это либо методом тыка, либо разглядывая крохотные пиктограммы рядом с ручками. Неестественность ассоциации вынуждает пользователей каждый раз изучать ее заново. Это действие со временем уходит в подсознание, но оно выполняется каждый раз и может закончиться ошибкой, когда пользователь спешит или отвлекается (что нередко происходит во время приготовления еды). В лучшем случае пользователь почувствует себя глупо, если повернет не ту ручку, а еда останется холодной, пока он не заметит ошибку. В худшем дело может закончиться ожогами и пожаром.

Решение этой проблемы заключается в таком изменении расположения ручек, чтобы было понятно, какой горелкой управляет каждая из них. Совсем не обязательно располагать ручки в точности так же, как горелки, однако их позиции должны ясно показывать целевую горелку каждой ручки. Плита, изображенная на рис. 11.12, – хороший пример эффективного ассоциирования элементов управления.

При такой схеме расположения понятно, что левая верхняя ручка управляет левой верхней горелкой. Расположение каждой ручки ви-

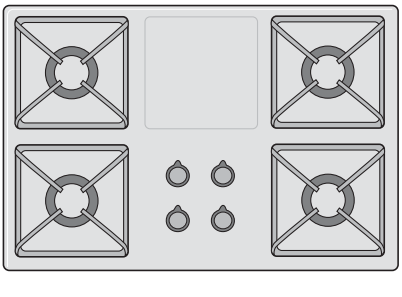


Рис. 11.12. *Недвусмысленное ассоциирование. Здесь понятно, с какой горелкой ассоциируется та или иная ручка, поскольку пространственная организация ручек четко связывает каждую ручку с горелкой*

зуально отражает, какую горелку она включает. Дональд Норман (Norman, 1989) называет такое интуитивно понятное размещение **естественным ассоциированием**.

Другого рода пример неудачной ассоциации приведен на рис. 11.13. В этом случае неясно **логическое ассоциирование** понятий с действиями.

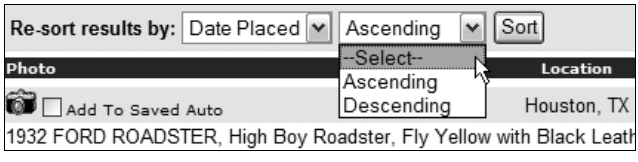


Рис. 11.13. *Пример проблемы логического ассоциирования. Если пользователь захочет посмотреть последние пункты, какую сортировку он должен выбрать – по возрастанию или убыванию? Эти термины плохо ассоциированы с тем, как пользователь воспринимает время*

Этот веб-сайт предлагает вниманию пользователя пару разворачивающихся меню, позволяющих отсортировать результаты поиска по дате. Пункт, выбранный в первом меню, влияет на то, что будет во втором. Если в первом меню выбрано Re-sort Results by: Date Placed (Отсортировать результат по: Дате размещения), то второе меню содержит пункты Ascending (По возрастанию) и Descending (По убыванию).

В отличие от плохого ассоциирования, характерного для кухонных плит, здесь *целевой объект* каждого элемента управления понятен – пункты разворачивающегося меню влияют на таблицу, расположенную под ним. Однако остается неясным *результат*: что получит пользователь, если он выберет сортировку по возрастанию?

Термины, выбранные для обозначения способов сортировки данных, не проясняют, что именно должен выбрать пользователь, который хочет, чтобы последние по времени пункты попали в начало списка. Фразы «по возрастанию» и «по убыванию» плохо связаны с ментальными моделями времени большинства пользователей. Люди не думают, что даты «возрастают» или «убывают»; скорее, они считают, что даты и события бывают «давние» и «недавние». Быстрое решение этой проблемы состоит в замене формулировок на Most Recent First (Вначале самый последний) и Oldest First (Вначале самый давний), как на рис. 11.14.

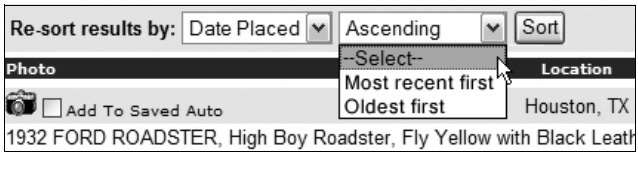


Рис. 11.14. *Внятное логическое ассоциирование. Здесь употребляются термины, которые пользователи легко ассоциируют с сортировкой по времени*

Создаете ли вы бытовые устройства, настольные приложения или же веб-сайты, вы всегда можете столкнуться с проблемами ассоциирования. Это такая область, где внимание к деталям окупается с лихвой. Вы можете существенно улучшить свой продукт за счет поиска и устранения проблем, связанных с ассоциированием, даже если у вас мало времени на внесение изменений. Что в результате? Продукт, в котором легче разобраться и с которым приятнее иметь дело.

Адаптация интерфейса к нуждам пользователя

Адаптация интерфейса подразумевает организацию его таким образом, чтобы минимизировать объем типичной навигации. На практике это означает размещение наиболее востребованных функций и элементов управления в самых удобных для пользователя местах и перенос редко используемых функций в глубь интерфейса, чтобы пользователь не «спотыкался» о них. Не следует удалять из программы редко используемые функциональные возможности, однако убрать их из рабочей области пользователя необходимо.

Адаптируйте интерфейс под типичную навигацию.

Самым важным принципом правильной адаптации интерфейса является принцип **соразмерности усилий**. Хотя он действует в отношении всех пользователей, он особенно уместен, когда речь идет о вечных середняках. Это принцип гласит, что люди готовы прилагать дополнительные усилия, если результат того стоит. Естественно, ценность результата определяется пользователем. Она не имеет никакого отношения к тому, насколько трудно было реализовать функциональную возможность, и полностью зависит от целей пользователя.

Если пользователь действительно чего-то хочет, он будет больше работать, чтобы получить желаемое. Например, если человек хочет стать хорошим теннисистом, он пойдет на корт и будет усиленно тренироваться. Тому же, кто не любит теннис, даже короткая тренировка покажется утомительной. Если пользователь хочет красиво отформатировать документ, разбить его на несколько колонок, задействовать разные шрифты и броские заголовки, чтобы произвести впечатление на начальника, у него будет высокая мотивация к исследованию всех закоулков программы и выяснению того, как добиться желаемого результата. Он приложит к проекту *соразмерные усилия*. Если другой пользователь просто захочет распечатать старый документ в одну колонку одним шрифтом, ничто не заставит его изучать средства форматирования документов.

Пользователи готовы прикладывать усилия соразмерно результату.

Сказанное означает, что если вы добавите к программе функциональные возможности, которыми сложно пользоваться, пользователи будут готовы терпеть трудности только в обмен на достойное вознаграждение. Поэтому пользовательский интерфейс вашей программы не должен быть сложным, когда речь идет о достижении простых результатов, но *может* быть сложным при необходимости достигать *сложных* результатов (и при условии, что такая необходимость возникает не слишком часто).

В интерфейсе допустима ситуация, когда расширенные функциональные возможности требуют от пользователя незначительных дополнительных усилий, будь то поиск в меню, открытие диалогового окна или открытие ящика стола. Принцип соразмерных усилий позволяет **адаптировать** интерфейс так, что простые повседневные функции всегда будут под рукой. Расширенные возможности, которые нужны не столь часто, но предлагают пользователю значительные выгоды, можно убрать подальше и доставать только по мере необходимости. Практически любой автоматический цифровой фотоаппарат может послужить хорошим примером адаптации: наиболее востребованная функция – спуск затвора – представлена крупной кнопкой, которая легко доступна в любой момент; менее востребованные функции, такие как коррекция экспозиции, требуют взаимодействия с экранными меню.

Вообще говоря, элементы управления и окна должны быть организованы в интерфейсе по трем параметрам – частоте использования, степени влияния на внешний вид интерфейса и степени риска.

- **Частота использования** определяет, как часто пользователь обращается к элементам управления, функциям, объектам и окнам при повседневном использовании продукта. Элементы и инструменты, необходимость в которых возникает чаще всего (много раз в течение дня), должны всегда быть под рукой, как говорилось в главе 10. Элементы, используемые реже (один или два раза в день), должны быть не дальше, чем «на расстоянии» одного-двух щелчков. Доступ к прочим элементам может потребовать двух или трех щелчков.
- **Степень влияния на внешний вид интерфейса** означает масштаб резких изменений в интерфейсе или обрабатываемом документе (информации), вызываемых активизацией той или иной функции либо команды. Вообще говоря, команды, оказывающие сильное влияние, следует прятать подальше (см. пояснения в главе 10).
- **Степень риска** относится к функциям, выполняющим необратимые действия или имеющим другие опасные последствия. Инструкции по запуску межконтинентальных баллистических ракет требуют, чтобы два человека одновременно повернули ключи в противоположных концах помещения – иначе запуск невозможен. Как и в случае с функциями, изменяющими внешний вид интерфейса, опасные функции следует убрать подальше, чтобы пользователи на них

не натыкались. Уровень риска можно считать произведением вероятности события на нежелательные последствия этого события.

Конечно, по мере того как пользователи приобретают опыт работы с этими возможностями, у них возникает потребность в коротких путях, и такие пути следует им предоставить. Когда программный продукт придерживается принципа соразмерных усилий, кривая обучения не исчезает, но пользователь перестает о ней задумываться – что столь же хорошо.

Отказ от иерархических структур

Иерархические структуры – один из самых надежных инструментов программиста. Большая часть данных внутри программы, да и сам ее код имеют иерархическую организацию. По этой причине многие программисты применяют иерархии (модель реализации) и в пользовательских интерфейсах. Как мы помним, первые меню были иерархическими. Однако пользователям трудно осуществлять навигацию в абстрактных иерархических структурах, за исключением тех случаев, когда эти иерархии основаны на ментальных моделях пользователей, а категории являются действительно взаимоисключающими. Программисты часто отказываются принять этот факт, поскольку чувствуют себя рядом с иерархическими структурами вполне комфортно.

Большинство людей знакомы с иерархиями по работе или по семейным отношениям, но иерархия не является естественным понятием для человека, когда речь идет о хранении произвольной информации и доступе к ней. Большинство механических систем хранения информации просты и состоят либо из одной последовательности объектов (как книжная полка), либо из нескольких последовательностей, имеющих один уровень глубины (как в картотечном шкафу с папками). Такой способ организации, подразумевающий один слой из нескольких групп, очень распространен, и его можно встретить в каждом доме и офисе. Поскольку он никогда не имеет больше одного уровня вложенности, мы называем подобную парадигму хранения **моноклинальной группировкой**.

Программисты легко работают с системами многоуровневой вложенности, где экземпляр какого-либо объекта хранится в другом экземпляре того же объекта. Большинству непрограммистов очень трудно воспринять эту идею. В сложных механических системах хранения информации на разных уровнях неизбежно используются разные методы хранения. Вы никогда не найдете папки внутри папок в картотечном шкафу или ящики внутри ящиков. Даже неоднородная схема «папка – ящик – шкаф» редко превышает два уровня вложенности. В метафоре рабочего стола, применяемой большинством оконных систем, вы можете вкладывать папки в папки до бесконечности. Неудивительно, что большинство компьютерных неофитов приходят в замешательство, когда сталкиваются с этой парадигмой.

Большинство людей раскладывает бумаги (и прочие предметы) по стопкам на основании какой-то одной характеристики: сюда – бумаги от фирмы Acme, туда – бумаги по проекту М, а личные бумаги – в ящик. Дональд Норман (Norman, 1994) называет такой подход «*шкаф с бумагами*». И только в компьютере документы по проекту М хранятся в папке под названием **Активные клиенты**, которая хранится в папке **Клиенты**, а та в свою очередь хранится в папке **Бизнес**.

Компьютерные технологии предлагают нам иерархические структуры в качестве инструмента решения жизненных проблем манипулирования большими объемами данных. Но когда эта модель реализации проявляется в модели, предъявляемой пользователям (подробнее о моделях см. главу 2), пользователи приходят в замешательство, поскольку она конфликтует с их ментальной моделью систем хранения. Моноклинальная группировка доминирует за пределами мира компьютеров настолько серьезно, что проектировщики взаимодействия могут игнорировать ее лишь на свой страх и риск.

Моноклинальная группировка не подходит для физического манипулирования данными такого объема, какой обычно представлен в компьютерах, однако это не означает, что она не годится в качестве *модели представления*. Решение этой головоломки – визуализировать структуру сообразно представлениям пользователей (моноклинальная группировка), но реализовать также средства поиска и доступа к данным, возможные лишь при иерархической организации информации. Иными словами, вместо того чтобы вынуждать пользователей осуществлять навигацию по многоуровневым древовидным структурам, дайте им инструменты для *удобного доступа к нужной информации*. Некоторые решения, способствующие этому, обсуждаются в главе 15.

12

Проектирование хорошего поведения

Как мы уже упоминали в главе 10, два социолога из Стэнфорда, Клиффорд Насс и Байрон Ривз, предположили, что люди, по-видимому, имеют инстинктивное представление о том, как следует себя вести с другими разумными существами. Как только какой-нибудь объект проявит интерактивность в достаточной степени (что, например, характерно для среднестатистического программного продукта), этот инстинкт пробуждается. Наша реакция на программный продукт как на разумное существо бессознательна и неизбежна.

Из этого исследования можно сделать фундаментальный вывод: если мы хотим, чтобы пользователям понравился наш программный продукт, он должен вести себя так, как ведет себя человек, приятный в общении. Если мы хотим, чтобы работа пользователя с нашей программой была продуктивна, программа должна вести себя как товарищ по работе, готовый поддержать коллегу. Чтобы этого добиться, полезно рассмотреть уместные рабочие отношения между людьми и компьютерами.

ПРИНЦИП проектирования

Компьютер работает, а человек думает.

Идеальное разделение труда в век компьютеров таково: компьютер работает, а человек думает. Писатели-фантасты и специалисты в области компьютерных технологий дразнят нас перспективами развития искусственного интеллекта: компьютеры будут думать за себя сами. Однако пользователям не особенно требуется помощь в мыслительных процессах: наша способность выявлять закономерности и творчески решать сложные задачи не имеет аналогов в мире кремния. Что нам действительно *требуется*, так это помощь в управлении информацией – в такой деятельности, как доступ, анализ, организация и визуализация

информации. А принятие решений на основе полученной информации лучше всего получается у людей – у «нервного обеспечения».[1]

Проектирование тактичных продуктов

Насс и Ривз утверждают, что программный продукт должен быть *вежливым*, но мы предпочитаем слово *тактичный*. Вежливость может свестись к протоколу – говорить «пожалуйста» и «спасибо», но не делать при этом ничего полезного. Зато подлинная *тактичность* означает чуть больше – первоочередное внимание к интересам других. Тактичное программное обеспечение ориентировано в первую очередь на цели и нужды пользователей, а не на собственные базовые функции.

Если программа скупа на информацию, окружает свои процессы ореолом тайны, заставляет прилагать усилия для поиска обычных функций и с легкостью обвиняет пользователя в своих промахах, человеку обеспечен неприятный и непродуктивный опыт. Это происходит независимо от того, насколько программа вежлива, мила, визуально метафорична, информативна и антропоморфна.

Зато продукт, который ведет себя уважительно и благородно и готов прийти на помощь пользователю, делает огромный шаг к созданию положительного опыта у работающих с ним людей.

Программный продукт должен вести себя как тактичный человек.

Интерактивные продукты часто раздражают нас потому, что они бестактны, а не потому, что им не хватает каких-то возможностей. Совершенно не факт, что создание тактичного продукта значительно сложнее, чем создание грубого или бестактного. Нужно всего-навсего представить себе взаимодействие, которое имитирует качества чувствительного и заботливого человека. Ни одно из этих качеств не противоречит более прагматичным целям обработки данных, стоящим перед любым цифровым продуктом. Сверх того, более гуманное поведение может оказаться самой прагматичной из всех целей, и в сочетании с правильной оркестровкой подобный диалог с пользователями может внести большой вклад в функциональность продукта.

У людей есть множество прекрасных качеств, которые делают их тактичными, и некоторые из них можно в той или иной степени воспроизводить в интерактивных продуктах. Как нам кажется, следующий перечень содержит некоторые наиболее важные качества тактичных интерактивных продуктов (и людей):

---

[1] Авторы используют слово wetware, введенное в широкое обращение писателями в жанре киберпанк. – *Примеч. перев.*

- Проявлять интерес
- Вести себя почтительно
- Проявлять услужливость
- Проявлять здравый смысл
- Предупреждать желания людей
- Проявлять инициативность
- Не перекладывать на других свои проблемы
- Держать коллегу в курсе дела
- Проявлять понятливость
- Иметь уверенность в себе
- Не задавать лишних вопросов
- Принимать ответственность
- Знать, когда можно отклониться от правил

Рассмотрим эти качества подробнее.

Тактичные продукты проявляют интерес к людям

Хороший друг хочет знать о вас больше. Он запоминает, что вам нравится, а что не нравится, чтобы сделать вам что-нибудь приятное в будущем. Любому человеку нравится, когда кто-то учитывает его вкусы.

С другой стороны, большинству программ безразлично, кто ими пользуется. На наших *персональных* компьютерах практически нет программных продуктов, которые знали бы о нас хоть что-нибудь *личное*, хотя мы используем их постоянно, многократно и безальтернативно. Положительный пример такого поведения демонстрируют броузеры Firefox и Microsoft Internet Explorer, которые запоминают вводимую пользователем на сайтах информацию, например адреса доставки или регистрационное имя.

Программа должна изо всех сил стараться запомнить привычки пользователя и особенно команды, которые поступали от него. Программисту, который ее создал, эта информация кажется сиюминутной – когда она нужна программе, программа просто требует, чтобы пользователь сообщил необходимые сведения, а затем забывает их, полагая, что запросит их снова в случае необходимости. Несмотря на то, что программа способна запоминать информацию лучше, чем человек, она все забывает, проявляя *бестактность*. Запоминание действий и предпочтений людей – это один из лучших способов пробудить положительные эмоции от применения цифрового продукта. Мы еще вернемся к теме памяти далее в этой главе.

Тактичные продукты ведут себя почтительно

Хороший работник сферы обслуживания проявляет почтительность по отношению к клиенту. Он относится к клиенту, как к начальнику.

Когда метрдотель в ресторане показывает нам, за какой столик сесть, мы воспринимаем его выбор столика как предложение, а не как приказ. Если мы вежливо попросим его предоставить нам другой столик, когда ресторан почти пуст, то ожидаем, что нас разместят за другим столиком. Если метрдотель откажет нам, мы, скорее всего, предпочтем другой ресторан, где *наши* пожелания ставятся выше пожеланий метрдотеля.

Нетактичный продукт ведет себя как начальник и осуждает поступки человека. Программа имеет право высказывать *мнение*, что мы ошиблись, однако судить наши действия – это дерзость с ее стороны. Программа может *предположить*, что лучше не щелкать по кнопке Submit, пока мы не укажем номер своего телефона, и объяснить возможные последствия, но, если мы захотим отправить данные без номера телефона, мы вправе ожидать, что программа поступит так, как велено. Само слово submit[1] (подчиняться, предоставлять на рассмотрение) и обозначаемое им понятие ставят с ног на голову естественную иерархию, где почтение обязан проявлять именно интерактивный продукт, а не человек. Это программа должна подчиняться пользователям, и само наличие кнопки Submit в любом приложении уже является грубостью. Помимо прочего, это слишком туманное слово, которое может запутать пользователя.

Тактичные продукты услужливы

Если мы спросим работника универсама, как найти необходимый нам товар, он не только ответит на наш вопрос, но и по своей инициативе сообщит более важную сопутствующую информацию – например, что более дорогой и качественный аналогичный товар продается сейчас со скидкой почти за такую же цену.

Большинство программ даже и не пытаются предоставить сопутствующую информацию. Они дают ограниченный ответ на конкретный вопрос, нимало не заботясь о предоставлении других сведений, даже если те непосредственно связаны с нашими целями. Когда мы выдаем текстовому редактору команду распечатать документ, он не сообщает нам, что в лотке принтера мало бумаги, что перед нами очередь из сорока документов, или что другой стоящий неподалеку принтер свободен. Дружелюбный коллега сообщил бы нам об этом.

Поиск правильного способа предлагать потенциально полезную информацию может оказаться непростым. Скрепыша, созданного компанией Microsoft, презирают практически все – за ее «умные» комментарии вроде этого: «Похоже, вы набираете письмо. Моя помощь не требуется?» Настрой у Скрепыша похвальный, но хотелось бы, чтобы он вел себя не столь нагло и понимал, когда пользователь явно не желает

[1] Данное слово используется на кнопке отправки заполненной формы в веб-анкетах. – *Примеч. науч. ред.*

его видеть. В конце концов, хороший официант не спрашивает клиента, хочет ли тот еще воды, а просто наполняет опустевший стакан, и имеет достаточно мозгов, чтобы не шнырять вокруг да около, если видит, что клиент увлечен беседой с девушкой.

Тактичные продукты проявляют здравый смысл

Неподходящие функции в неподходящих местах – отличительный признак плохо спроектированных интерактивных продуктов. В большинстве из них элементы управления востребованными функциями расположены рядом с элементами, которые не используются никогда. Вы легко найдете программное меню, где простые безвредные функции соседствуют с функциями, имеющими необратимые последствия («рычаги катапульты») и предназначенными исключительно для экспертов. Это все равно, что посадить посетителя ресторана рядом с открытым грилем.

Ходит множество страшных историй о том, как компьютерные системы многократно отсылали клиентам чеки на 0 долларов или счета на 957 142 039 долларов и 58 центов. Можно было бы ожидать, что система уведомит какого-нибудь клерка в бухгалтерии о подобных событиях, особенно если они происходят неоднократно, однако здравый смысл – редкость в мире информационных систем.

Тактичные продукты предупреждают потребности человека

Секретарша знает, что вам потребуется номер в гостинице, когда вы отправляетесь в командировку в другой город, даже если вы не попросили ее заказать номер. Она знает, какую комнату вы предпочитаете, и закажет номер без напоминаний с вашей стороны. Она предупреждает ваши потребности.

Веб-броузер большую часть времени простаивает, пока вы просматриваете веб-страницы. Он прекрасно мог бы предупредить ваши потребности и подготовиться к ним. Например, он бы мог за это время загрузить страницы, ссылки на которые представлены на экране. Велика вероятность, что мы захотим перейти по каким-нибудь из этих ссылок. Отменить нежелательный запрос легко, а ждать его выполнения долго. Ближе к концу главы мы обсудим способы рационального использования времени простоя в интересах клиента.

Тактичные продукты инициативны

Инициативный человек демонстрирует более широкий взгляд на порученное дело. Например, он не только помоет посуду, но и вытрет стол и выбросит окурки из пепельницы, поскольку эти действия тоже служат глобальной *цели*: навести порядок на кухне. Когда инициативный работник пишет отчет, он не забудет о красивой обложке и сделает

достаточное количество ксерокопий, чтобы хватило всем заинтересованным лицам.

Рассмотрим пример. Вы вручаете своему помощнику папку с документами и велите убрать на место. Он читает надпись на папке, скажем «Контракты с MicroBlitz», и смотрит, куда ее поставить в картотечном шкафу. Под буквой М он, к своему удивлению, находит папку с точно такой же надписью. Заглянув в нее, он выясняет, что она содержит контракт на 17 изделий, поставленных фирме MicroBlitz четыре месяца назад. Зато в новой папке находится контракт на 32 шестеренки, изготовление и поставка которых запланированы на следующий квартал. Ваш инициативный помощник приклеивает к старой папке ярлык «Контракт с MicroBlitz 7/03», а к новой – ярлык «Контракт с MicroBlitz на шестеренки 11/03». Именно такая инициатива позволяет нам считать его инициативным сотрудником.

Предыдущий ваш помощник был совершенным болваном. К тому же он не был инициативным и, окажись он в этой ситуации, он без тени сомнения поставил бы новую папку рядом со старой. Конечно, он выполнил бы ваше поручение, но мог бы сделать это лучше, что в будущем сократило бы время поиска нужных документов. Потому-то этот человек у вас больше не работает.

Если мы создадим в текстовом редакторе документ с контрактом на шестеренки и попытаемся сохранить его в каталоге MicroBlitz, то редактор предложит нам либо уничтожить старый контракт, либо отказаться от сохранения нового. Программа не только не обладает способностями вашего теперешнего помощника – она даже глупее предыдущего, который был законченным болваном. Она настолько тупа, что усматривает в ваших действиях желание выбросить старый документ лишь потому, что новому вы дали то же название.

Программа, как минимум, должна пометить файлы разными датами и сохранить их. Даже если она не способна выполнить это «радикальное» действие в одностороннем порядке, она должна хотя бы показать вам старый файл (и позволить переименовать его), прежде чем сохранять новый. Существует масса действий, которые программа могла бы выполнить, будь она более инициативной.

Тактичные продукты не перекладывают на вас свои личные проблемы

Общаясь с клиентом, агент любой фирмы помалкивает о своих личных проблемах и выказывает разумный интерес к проблемам клиента. Возможно, подобная однобокость не очень справедлива, но такова суть сферы обслуживания. Точно так же интерактивный продукт должен помалкивать о своих проблемах и проявлять интерес к проблемам использующих его людей. Поскольку компьютеры не имеют своего «я» и не являются чувствительными особами, они превосходно подходят

для этой роли. Тем не менее они, как правило, ведут себя прямо противоположным образом.

Программа жалобно хнычет, выдавая сообщения об ошибках, прерывает нашу работу диалоговыми окнами, требующими подтверждения, и хвастается, выводя на экран никому не нужные уведомления. («Документ успешно сохранен!» О, как это мило с Вашей стороны, госпожа Программа! А что, Вы иногда *неуспешно* сохраняете документы?) Нам нет дела до приступов неуверенности в себе, случающихся у программы при необходимости очистить Корзину. Мы не желаем слушать ее нытье о том, куда помещать файл. Информация о скорости передачи данных и о последовательности загрузки интересует нас не больше, чем рассказ о несчастной любви представителя службы поддержки. Программа не только не должна распространяться о своих проблемах – у нее должно быть достаточно интеллекта, уверенности и полномочий, чтобы решить эти проблемы самостоятельно. Эта тема подробно обсуждается в главе 25.

Тактичные продукты держат нас в курсе дел

Хотя мы и не желаем, чтобы программа непрестанно докучала нам сообщениями о своих страхах и маленьких победах, мы хотим быть в курсе событий, важных для *нас*. Мы не желаем выслушивать рассказы бармена о его недавнем разводе, но будем ему благодарны, если он поместит ценники на видном месте и повесит объявление о том, когда будет трансляция ближайшего футбольного матча, кто играет и сколько очков у нашей любимой команды. Эта информация не прерывает нашу деятельность, она просто находится в поле зрения. Подобно этому программа может обеспечить насыщенную обратную связь, сообщая о том, что происходит. Об этом мы тоже поговорим в главе 25.

Тактичные продукты понятливы

Большинство существующих программных продуктов не слишком сообразительны. В большинстве случаев программы демонстрируют очень узкое понимание проблемы. Программа хорошо справляется с трудной работой, но лишь в том случае, когда получает четкую команду в нужный момент времени. Если, например, вы попросите систему автоматизированного управления складом сообщить, каков запас каких-нибудь деталей, она послушно сверится с базой данных и выдаст вам точное количество деталей на момент запроса. А что, если через двадцать минут кто-то в иногороднем филиале вашей фирмы закажет со склада весь запас этих деталей? Вы будете принимать решения, находясь в заблуждении, а ваш компьютер будет стоять без дела, пропуская миллиарды циклов. Это неумно с его стороны. Если вы однажды запросили информацию о деталях, разве это не должно навести на мысль, что вы запросите информацию о них снова? Возможно, вы не захотите получать отчеты о наличии деталей ежедневно до конца

жизни, но не исключено, что они будут нужны вам в течение недели. Понятливая программа следит за действиями пользователя и на их основании предлагает ему соответствующую информацию.

Продуктам следует внимательно относиться и к нашим предпочтениям, запоминать их самостоятельно, без подсказки. Если пользователь всегда разворачивает окно в полный экран, приложение после нескольких сеансов должно запомнить это и всегда запускаться с уже развернутым окном. То же самое относится к расположению палитр, стандартных инструментов, часто используемых шаблонов и прочим полезным настройкам рабочей среды.

Тактичные продукты уверены в себе

Интерактивные продукты должны вести себя уверенно. Если мы сообщаем компьютеру, что нужно удалить файл, он не должен спрашивать: «Вы уверены?» Конечно, мы уверены, иначе бы не попросили об этом. Он не должен перепроверять нас или себя.

С другой стороны, если у компьютера возникает подозрение, что мы ошибаемся (а оно у него присутствует постоянно), он должен предвидеть, что мы передумаем, и быть готовым восстановить файл по нашей просьбе.

Как часто вам доводилось щелкнуть по кнопке Печать и уйти пить кофе, а потом, вернувшись, увидеть в центре экрана ужасное диалоговое окно с вопросом: «Вы уверены, что хотите отправить документ на печать?» Невозможность положиться на программу способна привести в бешенство и является полной противоположностью тому, как мы представляем себе тактичное поведение человека.

Тактичные продукты не задают лишних вопросов

Как говорилось в главе 10, нетактичная программа задает массу раздражающих вопросов. Слишком широкий выбор быстро перестает быть достоинством и превращается в пытку.

Варианты выбора можно предлагать по-разному. Например, их можно выложить, как товары в витрине магазина. Мы смотрим на витрину, когда нам заблагорассудится, разглядываем, выбираем или игнорируем предложенные товары. Никто не задает нам вопросов. И, наоборот, выбор можно навязывать пользователю, как вопросы таможенника на границе: «Вы провозите что-нибудь, что надо включить в декларацию?» Мы не знаем последствий этого вопроса. Обыщут нас или нет? Программа никогда не должна устрашать пользователей подобным образом.

Тактичные продукты аккуратно обрабатывают свои сбои

Когда ваш друг совершает проступок, он после этого старается исправить то, что еще может быть исправлено. Когда программа обнаруживает фатальную проблему, она оказывается перед выбором – либо по-

пытаться подготовиться к провалу и не причинить вреда пользователю, либо «разбиться и сгореть».

Большинство программ имеют дело с большими объемами данных и множеством параметров. При аварийном завершении работы эта информация, как правило, безвозвратно теряется, и пользователь остается на перроне с чемоданом в руках. Например, программа принимает электронное письмо на ваше имя, и ей не хватает памяти во время выполнения какой-то процедуры, спрятанной глубоко в ее недрах. Как типичное настольное приложение она выдает сообщение, в конечном счете означающее: «Вы облиты с ног до головы», – и закрывается, как только вы щелкнете по кнопке OK. Вы перезапускаете программу (а иногда и перезагружаете компьютер) и обнаруживаете, что программа потеряла письмо. А связавшись с почтовым сервером, вы узнаете, что он стер письмо, поскольку оно было передано вашей программе. Качественная программа не должна вести себя подобным образом.

В приведенном примере программа приняла письмо от сервера, который стер копию, но не позаботилась о том, чтобы письмо было надлежащим образом сохранено на вашем компьютере. Если бы программа убедилась, что письмо сохранено на локальном диске, до того, как она сообщила серверу об успешном получении письма, проблема бы не возникла.

Некоторые качественно спроектированные программные продукты, такие как Ableton Live, великолепный инструмент для исполнения музыки, полагаются на историю операций, позволяющую выполнять откат после сбоев. Это замечательный пример того, как продукты могут следить за действиями пользователей и при возникновении проблем легко выходить из положения.

Даже если программа не завершает работу аварийно, нетактичное поведение представляет угрозу, особенно во Всемирной паутине. Пользователям нередко приходится вводить много информации, заполняя формы на веб-страницах. Заполнив, скажем, десять или одиннадцать полей, пользователь щелкает по кнопке Submit, но из-за какой-то ошибки сайт отвергает ввод и велит пользователю исправить данные. Пользователь щелкает по кнопке, возвращающей его на предыдущую страницу, – и видит, что корректная информация, набранная в десятке полей, выброшена вместе с единственной неправильной строчкой. Вспомните, как ваш злобный учитель географии порвал и выбросил ваш реферат о Южной Америке, потому что вы написали его карандашом, а не авторучкой. Разве вы не испытываете ненависть к географии по сей день? Не создавайте продукты, действующие в духе этого учителя.

Тактичные продукты знают, когда можно отклониться от правил

Когда системы ручной обработки информации подвергаются компьютеризации, что-то неизбежно теряется. Хотя системы автоматизированной обработки заказов могут принять на миллион заказов больше, чем клерк, клерк способен *обойти инструкции*, что немыслимо для автоматизированных систем. В автоматизированной системе почти никогда нет способа отойти от заданного образа действий ради некоторого выигрыша.

Если при ручной обработке заказов клерку позвонит его приятель из отдела продаж и объяснит, что быстрая обработка того или иного заказа может обернуться выгодной сделкой, клерк ускорит свою работу над данным заказом. Если в каком-то заказе будет отсутствовать необходимая информация, клерк частично обработает его и не забудет запросить эту информацию впоследствии, чтобы внести ее в заказ. Автоматизированные системы, как правило, не могут похвастать подобной гибкостью.

Большинство автоматизированных систем признают только два состояния: объект либо отсутствует, либо полностью соответствует требованиям. Промежуточные состояния не распознаются и не принимаются. В любой системе ручной обработки возможен важный, хотя и парадоксальный вариант – неоговоренный, недокументированный, но широко принятый: объект может находиться в **подвешенном** состоянии, когда транзакция принята, но все еще не доведена до конца. Клерк создает это состояние в уме, или на рабочем столе, или в заднем кармане.

Пусть, например, цифровой системе требуется информация о клиенте и о заказе, чтобы выписать счет. В то время как клерк может выписать счет до того, как получит информацию о клиенте, компьютерная система отвергнет транзакцию, не желая выписывать счет без полных сведений.

Характеристика ручных систем обработки, позволяющая работникам выполнять действия в нестандартной последовательности или до того, как будут соблюдены все формальности, называется **сговорчивостью**. Сговорчивость становится одной из первой жертв компьютеризации, а отсутствие этого качества является основной причиной «нечеловечности» компьютерных систем. Налицо естественный результат следования модели реализации. Программист не видит необходимости в создании промежуточных состояний, потому что компьютер в них не нуждается. Тем не менее существует сильная человеческая потребность – иметь возможность слегка отступать от правил.

Одним из достоинств сговорчивых систем является уменьшение количества ошибок. Допуская наличие в системе множества мелких временных ошибок и предполагая, что люди исправят их до того, как воз-

никнут проблемы на следующих этапах работы, мы сможем избежать более серьезных ошибок. Парадоксально, но большинство строгих правил, сформулированных в компьютерных системах, имеют целью предотвращение как раз таких ошибок. Эти негибкие правила разделяют людей и программы на два враждующих лагеря, и, поскольку людям не разрешается допускать недоделки ради предотвращения крупных ошибок, они не заботятся о защите программ от действительно колоссальных проблем. Когда негибкие ограничения накладываются на людей, которым свойственна гибкость, проигрывают обе стороны. В конечном итоге, если людям не позволяется делать то, что они хотят, страдает бизнес, а компьютерным системам все равно в конце концов приходится обрабатывать неверные данные.

В реальном мире недостающая информация (равно как излишняя информация, не вписывающаяся в стандартный формат) нередко является важным средством на пути к успеху. Системы обработки информации редко способны управиться с данными из реального мира. Они моделируют лишь жесткие повторяющиеся данные из транзакций, что-то вроде скелета реальных транзакций, которые включают в себя встречи с людьми, путешествия и развлечения, игру в гольф, имена супругов, детей и звезд спорта. Бывает, что транзакция может быть завершена только через две недели после установленного официального срока. Большинство компаний предпочтет принять работу с недоделками, когда подойдет срок, чем наблюдать, как многомиллионная сделка превращается в дым. В реальном мире ограничения нарушаются постоянно. Тактичный продукт должен осознавать и учитывать этот факт.

Тактичные продукты берут на себя ответственность

Слишком многие интерактивные продукты занимают позицию «я за это не отвечаю». Когда такая программа передает поручение аппаратной части устройства, она «умывает руки», полагая, что глупая аппаратура сама закончит дело. Любому пользователю ясно, что такая программа не является ни тактичной, ни ответственной, что она не подставляет плечо под общую ношу и не помогает пользователю работать эффективнее.

Рассмотрим типичный пример – печать. Приложение отправляет отчет на 20 страниц принтеру и одновременно выводит диалоговое окно наблюдения за процессом с кнопкой отмены. Если пользователь быстро сообразит, что забыл внести одно важное изменение, он щелкнет по кнопке, как только первая страница показалась из принтера. Программа тут же отменит операцию печати. Однако без ведома пользователя, пока принтер готовился выводить первую страницу, компьютер успел послать 15 страниц в буфер принтера. Программа отменяет печать последних пяти страниц, но принтер об отмене не знает ровным счетом ничего. Он знает про первые 15 страниц и продолжает печатать

их. Тем временем программа самодовольно сообщает пользователю, что операция отменена. Программа лжет – и пользователь видит это.

Пользователю нет дела до проблем со связью между приложением и принтером. Его не волнует, что эта связь односторонняя. Зато он твердо знает, что передумал печатать документ до того, как первая страница появилась в выходном лотке, знает, что он щелкнул по кнопке Отмена и что после этого глупая программа продолжила печатать и вывела 15 страниц, хотя он заблаговременно приложил все усилия, чтобы прекратить печать. Причем программа ведь даже подтвердила команду отмены! Выбросив 15 испорченных страниц в мусорную корзину, пользователь недовольно рычит на глупую программу.

Представьте, как развивались бы события, если бы программа могла обмениваться сообщениями с драйвером принтера, а тот – с принтером. Если бы программа была достаточно интеллектуальна, задание на печать можно было бы без труда отменить еще до того, как была испорчена вторая страница. У принтера определенно есть функция отмены, просто программа слишком ленива, чтобы воспользоваться ею.

Проектирование интеллектуальных продуктов

Услужливые продукты и люди должны быть не только тактичными, но и **умными**. Благодаря писателям-фантастам и футурологам существует некоторая путаница относительно смысла понятия «интеллектуальный интерактивный продукт». Некоторые наивные наблюдатели полагают, что умные программы действительно способны вести себя как разумные существа.

И хотя это определенно было бы здорово, факты таковы: нашим цифровым инструментам еще очень и очень далеко до осуществления таких мечтаний. Если вы собираетесь создать продукт в текущей пятилетке, требуется другая интерпретация этого термина. Эта другая интерпретация подразумевает, что интеллектуальные продукты способны работать более усердно даже в непростых условиях и даже тогда, когда пользователи не вовлечены в процесс на сто процентов. Наши мечты о думающих компьютерах – это хорошо, но существует более близкая и более насущная возможность – заставить компьютеры работать более усердно. В оставшейся части главы мы расскажем о некоторых способах заставить программы лучше служить людям.

Использование циклов простоя

Каждая инструкция любого приложения должна пройти через центральный процессор, соблюдая порядок общей очереди, и разработчики оптимизируют код для прохождения через это игольное ушко. Программисты тяжело трудятся, чтобы минимизировать число инструкций и гарантировать мгновенный отклик на действия пользователей.

При этом мы часто забываем, что как только процессор закончил поспешное выполнение работы, он начинает простаивать – до тех пор, пока пользователь не даст другую команду. Мы тратим невероятно много сил на сокращение времени отклика компьютера, но мы совсем или почти совсем не прикладываем усилий к тому, чтобы он проактивно выполнял работу, пока не занят реакцией на действия пользователя. Наши программы повелевают процессором, как полководцы армиями, то останавливая его, то бросая в бой. Атаки – это захватывающе, но безделье пора прекращать.

В современных вычислительных системах пользователям приходится помнить слишком много вещей, например имена файлов и их точное расположение в файловой системе. Если пользователь захочет найти квартальный план, он должен будет либо вспомнить его название, либо выполнить поиск по файловой системе. В это время процессор простаивает, впустую расходуя миллиарды рабочих тактов.

Большинство современных программ не обращают внимания на контекст. Когда пользователь работает над сложным отчетом в условиях острого дефицита времени, программа предлагает ему не больше помощи, чем когда он просто балуется с цифрами на досуге. Ситуация, когда программа с чистой совестью простаивает, а пользователь работает, недопустима. Пора нашим компьютерам подставить плечо под груз, который мы несем ежедневно.

В нормальной ситуации типичный пользователь не может выполнить никакое действие быстрее, чем за несколько секунд. Типичному настольному компьютеру этого времени достаточно, чтобы выполнить, как минимум, *миллиард* инструкций. Почти наверняка эти внутренние рабочие циклы будут выполнены вхолостую. Процессор *просто ждет*. Аргумент против использования этого времени простоя всегда был один: «Мы не можем делать предположения; они могут оказаться ошибочными...» Современные компьютеры настолько мощны, что этот аргумент, сам по себе справедливый, часто не имеет отношения к делу. Проще говоря, не страшно, если программа сделает ошибочное предположение. У компьютера достаточно мощности, чтобы программа сделала несколько предположений и отбросила результаты ошибочных, когда пользователь совершит окончательный выбор.

Вытесняющая многозадачность и потоки в Windows и Mac OS X позволяют выполнять работу в фоновом режиме без заметного снижения производительности обслуживания пользователя. Программа может запустить поиск файла, а когда пользователь начнет что-нибудь вводить с клавиатуры, приостановить поиск до следующей паузы. Рано или поздно пользователь задумается – и у программы будет достаточно времени, чтобы просканировать целый диск. Пользователь даже ничего не заметит. Именно такого рода поведение делает механизм поиска Spotlight в Mac OS X на голову выше того, что реализовано в Windows. Результаты поиска выдаются практически мгновенно, поскольку

операционная система использует время простоя для индексирования жесткого диска.

Всякий раз, когда приложение открывает модальное диалоговое окно, оно переходит в режим простоя, не делая ровным счетом ничего, пока пользователь сражается с диалогом. Такое не должно происходить. Диалоговому окну должно быть несложно пораскинуть мозгами и найти способ помочь пользователю. Что пользователь сделал последний раз в такой момент? Приложение может хотя бы предложить предыдущее действие в аналогичной ситуации в качестве основного варианта для этого раза.

Мы должны научиться сами и научить наши творения думать в упреждающем ключе, чтобы отыскивать способы помочь людям в достижении их целей и решении задач.

Интеллектуальные продукты обладают памятью

Если задуматься, станет очевидно: чтобы человек воспринимал интерактивный продукт как тактичный и интеллектуальный, этот продукт должен обладать знанием о человеке и способностью учиться, наблюдая за его поведением. Представленные выше качества тактичных продуктов подтверждают этот факт: чтобы продукт стал действительно полезным и тактичным, он должен *запоминать* важные факты о людях, с которыми взаимодействует.

Зачастую программами тяжело пользоваться потому, что они работают исходя из рациональных, логических предположений – к сожалению, совершенно ошибочных. Программисты и проектировщики часто предполагают, что поведение пользователей случайно и непредсказуемо и что пользователя необходимо постоянно допрашивать, чтобы выявить правильный курс действий. Хотя поведение человека действительно не такое упорядоченное, как поведение цифрового компьютера, оно редко бывает случайным, так что можно сказать заранее: глупые вопросы будут раздражать пользователей.

Если бы ваше приложение, веб-сайт или прибор могли предсказывать, что пользователь сделает в следующую минуту, разве это не улучшило бы взаимодействие с ним? Если бы ваша программа знала, какой выбор сделает пользователь в том или ином диалоговом окне или в форме, разве нельзя было бы отказаться от этого компонента интерфейса? Разве не следует считать заведомое знание о действиях пользователя страшным секретным оружием при проектировании интерфейса?

Так вот: вы *можете* предсказывать поведение пользователей. Вы *можете* встроить в приложение шестое чувство, которое подскажет ему со сверхъестественной точностью, что именно сделает пользователь в следующий момент! Все эти миллиарды впустую потраченных рабочих циклов процессора можно употребить с пользой: достаточно лишь добавить в интерфейс память.

Когда мы используем слово **память** в этом контексте, то не имеем в виду оперативную память компьютера. Память – это способность приложения отслеживать действия пользователя в ходе многих сеансов его работы и реагировать на них. Если ваша программа будет просто запоминать, что (и как) делал пользователь в ходе нескольких последних сеансов, она сможет руководствоваться этой информацией, выбирая линию своего поведения

Если мы наделим наши продукты сознанием в отношении поведения пользователя, то есть памятью, а также гибкостью, позволяющей представлять информацию и функциональность исходя из предшествующих действий пользователя, то сможем задействовать огромный потенциал и сделать работу пользователя более эффективной и приятной. Кому из нас не хотелось бы иметь умного и заинтересованного помощника, проявляющего инициативу, старание, здравый смысл и обладающего цепкой памятью? Продукт, эффективно использующий свою память, походит на такого мотивированного помощника, который вспоминает полезную информацию и предпочтения начальника без просьб и напоминаний. Простые вещи способны создавать разительный контраст – контраст между продуктом, который пользователи терпят, и продуктом, который они *обожают*. В следующий раз, обнаружив, что ваше приложение задает вопрос пользователю, заставьте его задавать это вопрос себе.

Можно подумать, что вся эта суета с памятью необязательна, – ведь проще каждый раз задавать пользователю вопросы. Программисты, не задумываясь, создают диалоговые окна, чтобы затребовать любую информацию, недоступную напрямую. Однако, как мы говорили в главе 10, *людям не нравится, когда их допрашивают*. Непрерывный допрос – не просто излишнее налоговое бремя для пользователя, но и, с точки зрения психологии, неявный способ выражать сомнение в их компетентности.

Программы в большинстве своем забывчивы, от сеанса к сеансу они ничего или почти ничего не запоминают. Когда же наши программы *все-таки* оказываются достаточно умными, чтобы сохранять какую-то информацию между обращениями к ним, то это, как правило, информация, облегчающая работу *программиста*, но не пользователя. Программа с легкостью выбрасывает информацию о том, как ее использовали, какие настройки были изменены, в каком окружении она работала, какие данные обрабатывала, кто ею пользовался и как часто он обращался к тем или иным функциональным возможностям. При этом она записывает в файлы инициализации имена драйверов, номера портов и прочие подробности, облегчающие жизнь программисту. Можно использовать эти же файлы для существенного повышения уровня интеллектуальности программы с точки зрения пользователя.

Связность задач

Предсказания относительно поведения пользователя на основе его предыдущих действий базируются на принципе **связности задач**: наши цели и способы достижения этих целей (с помощью задач), вообще говоря, являются одними и теми же изо дня в день. Это справедливо не только в отношении чистки зубов и поглощения завтрака, но и в том, что касается нашей манеры пользоваться текстовыми редакторами, почтовыми программами, сотовыми телефонами и бизнес-приложениями.

Когда потребитель работает с вашим продуктом, высока вероятность, что и функции, которыми он пользуется, и способ обращения к ним будут почти теми же, что в прошлые разы. Он, возможно, будет работать с теми же документами или хотя бы с документами того же типа, расположенными в тех же каталогах. Конечно, он не будет каждый раз делать одно и то же, но, скорее всего, выполняемые задачи будут укладываться в довольно ограниченный набор рабочих шаблонов. Вы сможете с высокой вероятностью предсказать поведение ваших пользователей, просто запоминая, что они делали в течение нескольких последних сеансов работы с программой. Это позволит значительно сократить количество вопросов, задаваемых пользователю приложением.

Например, Салли использует Excel совершенно не так, как Кацуо, но вполне постоянна в своих привычках. Кацуо предпочитает шрифт Times размером 9 пунктов, а Салли – Helvetica 12 пунктов, причем Салли пользуется им регулярно. Программе не обязательно спрашивать Салли, какой шрифт использовать: каждый раз Helvetica 12 пунктов будет надежной отправной точкой.

Запоминание выбора и значений по умолчанию

Правило, определяющее, какую информацию должна запоминать программа, формулируется просто: запоминать следует все, что выбирает пользователь.

Следует запоминать все, что выбирает пользователь.

Когда перед приложением встает выбор – и особенно если выбор должен сделать пользователь, – приложение должно запоминать сделанный выбор и помнить о нем между запусками. Вместо жестко запрограммированного значения программа должна выбирать в качестве значения по умолчанию предыдущие настройки – и у нее будет гораздо больше шансов предоставить пользователю то, что ему требуется. Можно перестать задавать вопросы пользователю и автоматически выбрать тот вариант, который пользователь предпочел в предыдущий раз, позволяя человеку скорректировать выбор в том случае, если программа ошиблась. Следует запоминать любые установленные пользо-

вателем режимы – и эти режимы должны действовать, пока пользователь не изменит их вручную. Если пользователь проигнорировал или отключил какую-то функциональную возможность, не следует предлагать ее снова. Когда она понадобится, она сам ее найдет.

В программах, не имеющих памяти, больше всего раздражает то, что они скупы на помощь, когда дело касается файлов и дисков. Файлы и диски – это как раз та область, где пользователю особенно требуется помощь. Текстовый редактор Word запоминает последний каталог, в котором пользователь искал файл. К сожалению, если пользователь всегда помещает файлы в каталог Письма, но однажды отредактирует шаблон из каталога Шаблоны, то все его следующие письма будут сохраняться в каталоге Шаблоны, а не в каталоге Письма. Таким образом, программе следует запоминать не просто последний каталог, из которого были прочитаны файлы. Она должна запоминать последний каталог для файлов *каждого типа*.

Размер и положение окон тоже следует запоминать. Если пользователь развернул окно в прошлый раз, оно должно быть развернуто и в следующий. Если пользователь расположил окно рядом с другим окном, то в следующий раз расположение должно быть точно таким же, без дополнительных указаний со стороны пользователя. Приложения Microsoft Office сегодня удовлетворяют этому требованию.

Запоминание шаблонов

Продукт с хорошей памятью может принести человеку пользу несколькими способами. Наличие у продукта памяти сокращает налоговое бремя – непроизводительные усилия, нацеленные на управление инструментом, а не на работу с ним. Значительная доля нагрузки на пользователя, создаваемой интерфейсом, заключается в необходимости объяснять программе вещи, которые ей уже следует знать. Предположим, например, что вам нередко приходится «инвертировать» текст, чтобы белые буквы были видны на черном фоне. Для этого вы выделяете текст и меняете цвет шрифта на белый. Не снимая выделение, вы устанавливаете черный цвет фона. Если бы программа обращала достаточно внимания на ваши действия, она бы заметила, что вы выполняете две операции форматирования, не снимая выделения. С вашей точки зрения это, в сущности, единая операция. Со стороны программы было бы любезно, если бы она заметила, что подобное необычное поведение пользователя повторяется неоднократно, и создала бы специальный стиль, или, еще лучше, специальный элемент управления в интерфейсе и назвала бы его «Негатив».

Многие популярные программы позволяют пользователям настраивать значения по умолчанию, но это не так эффективно, как наличие памяти. Настройка умолчаний обременительна для пользователей, за исключением самых опытных. Многие просто никогда не поймут, как настраивать умолчания по своему вкусу.

Какие действия запоминать?

Следует запоминать *все*, что делает пользователь. На жестких дисках полно свободного места – и память вашего приложения найдет этому месту достойное применение. Мы привыкли считать, что программы требуют неоправданно много места на диске, потому что большое приложение, как правило, занимает порядка 200 Мбайт. Это типично для программы, но нетипично для данных пользователя. Если текстовый редактор будет сохранять один килобайт информации о поведении пользователя после каждого запуска, это не так уж много. Скажем, вы запускаете текстовый процессор 10 раз в течение рабочего дня. В году примерно 200 рабочих дней, так что получается 2000 запусков программы в год. Потребление места на диске составит 2 Мбайта – и это расчет с запасом на целый год! Обои на рабочем столе занимают на диске немногим меньше места.

Расположение файлов

Все элементы интерфейса, выполняющие открытие файлов, должны запоминать, откуда пользователь берет свои файлы. Большинство пользователей хранят файлы для каждой конкретной программы в довольно ограниченном наборе каталогов. Программе следует запомнить эти исходные каталоги и предлагать их список в диалоговом окне открытия файлов. Пользователь не должен проходить по дереву каталогов до нужного места более одного раза.

Выводы, сделанные программой

Программа должна запоминать не только очевидные факты, но и выводы, которые можно сделать на их основании. Например, если программа запоминает количество измененных байтов при каждом открытии файла, она будет в состоянии помочь пользователю, выполняя некоторые проверки на разумность действий. Предположим, для некоторого файла исторические значения счетчика измененных байтов составляют 126, 94, 43, 74, 81, 70, 110 и 92. Если пользователь в следующий раз изменит 100 байтов, то в этом не будет ничего необычного. Если же вдруг счетчик зашкалит за 5000, то программа должна заподозрить что-то неладное. Хотя есть вероятность, что пользователь нечаянно сделал то, о чем впоследствии пожалеет, она все-таки невелика, поэтому беспокоить его диалоговым окном подтверждения изменений не стоит. Тем не менее со стороны программы будет весьма разумно на всякий случай сохранить помеченную копию файла, в которой эти 5000 байтов еще не изменены. Скорее всего, программе не придется хранить эту копию после следующего обращения пользователя к данному файлу, поскольку пользователь с большой вероятностью заметит ошибку сразу и потребует откатить изменения немедленно.

Межсеансная отмена

Большинство приложений очищают стек отменяемых действий, как только пользователь закрывает документ или само приложение. Это недальновидный шаг. Приложение должно записывать стек изменений в специальный файл. Когда пользователь снова откроет свой файл, приложение загрузит стек отмены со всеми действиями, выполненными в течение прошлого сеанса, даже если он был неделю назад!

Ранее введенные данные

Приложение с хорошей памятью может минимизировать количество ошибок, сделанных пользователем, поскольку пользователю придется вводить меньше информации. Значительная ее часть будет введена автоматически из памяти приложения. Например, если программа, выписывающая счета, введет дату, номер отдела и другую стандартную информацию из своей памяти, у клерка будет меньше возможностей заполнить эти поля с ошибками.

Если приложение запоминает, что ввел пользователь, и на основании этой информации выполняет проверку его действий впоследствии, то она может отфильтровать ошибочные данные. Такая возможность существует в современных веб-броузерах, таких как Internet Explorer и Firefox. Именованные поля данных фиксируют ранее введенную информацию, так что пользователь впоследствии имеет возможность выбирать значения из списка. Если пользователя заботят соображения безопасности, функцию можно отключить, однако для большинства людей она оказывается полезной, поскольку экономит время и сокращает количество ошибок набора.

Манипуляции с файлом третьими приложениями

В промежутке между запусками приложение может оставить работать небольшой процесс, который следит за файлами этого приложения. Процесс отслеживает, куда они скопированы, кто их читает и кто пишет в них. Эта информация может оказаться полезной для пользователя, когда он запустит приложение в следующий раз. Когда он попытается открыть какой-либо файл, программа поможет ему найти этот файл, даже если он был перемещен. Программа сообщит пользователю и о других манипуляциях с этим файлом – например, был ли он распечатан или отправлен по факсу. Конечно, эта информация может и не понадобиться, но компьютер она не особенно нагрузит – в конечном итоге, это всего лишь невостребованные и выброшенные биты информации.

Реализация памяти в ваших приложениях

Когда разработчики признают всю мощь принципа связности задач, процесс создания программного продукта претерпевает значительные изменения. Проектировщики обнаруживают, что их мышление при-

обретает новое качество. Традиционный подход с обязательной выдачей диалоговых окон заменяется обдуманным процессом, в котором проектировщик задается более тонкими вопросами. Как *много* информации должно запоминать приложение? Что именно оно должна запоминать? Должно ли оно помнить больше, чем просто последние настройки? Что понимать под изменением в поведении пользователя? Проектировщики рисуют в своем воображении различные ситуации, например: пользователь принимает один и тот же формат даты 50 раз подряд, а затем один раз вручную вводит дату в другом формате. Какой формат программа должна использовать в следующий раз: тот, который использовался 50 раз, или самый последний? Сколько раз должен быть указан формат, чтобы стать форматом по умолчанию? Программа не должна спрашивать об этом у пользователя лишь потому, что здесь имеется неоднозначность. Она должна по своей инициативе принять разумное решение. Если оно окажется ошибочным, пользователь будет вправе отменить его.

В следующих разделах разъясняются некоторые типичные шаблоны принятия решений пользователем. Это поможет нам найти ответы на сложные вопросы, относящиеся к связности задач.

Сокращение множества решений

Люди предпочитают сокращать бесконечное множество решений до небольшого конечного. Даже если вы каждый раз действуете по-разному, вы все равно выбираете свои действия из небольшого повторяющегося набора вариантов. Люди выполняют **сокращение множества решений** неосознанно, а вот программа может это заметить и действовать соответствующим образом.

Например, если вы вчера ходили за покупками в некоторый универсам, это еще не означает, что вы ходите только в него. Однако когда вам в следующий раз потребуются бакалейные товары, вы, вполне вероятно, пойдете туда же. Аналогично, даже если меню в вашем любимом китайском ресторанчике содержит 250 блюд, весьма вероятно, что вы выберете одно блюдо из пяти-шести любимых. Когда люди едут домой с работы, они выбирают один из немногих маршрутов, в зависимости от дорожной обстановки. Компьютеры, конечно, не переутомятся от того, что запомнят четыре или пять вариантов.

Хотя запоминать последнее действие лучше, чем не запоминать ничего, этот подход может привести к ошибке, если множество решений состоит точно из двух элементов. Если, например, вы всегда читаете файлы из одного каталога, а записываете в другой, то каждый раз, предлагая вам последний каталог, приложение будет ошибаться. Решение состоит в том, чтобы помнить больше, чем один лишь последний выбор пользователя.

Сокращение множества решений наводит нас на мысль, что информация о предпочтениях пользователя, которую должна запоминать про-

грамма, неким образом группируется. Не бывает какого-то одного правильного варианта – они все правильные. Приложение должно находить более тонкие «зацепки», чтобы определить, какое решение из небольшого множества подходит в данном случае. Например, если вы применяете программу оформления чеков для оплаты своих счетов, приложение может быстро сообразить, что только два или три счета используются регулярно. Но как ей определить по чеку, к какому счету он относится? Если программа запомнит, какие получатели и суммы соответствуют тому или иному счету, ей будет легко принимать решение. Вы платите за квартиру каждый раз одну и ту же сумму! То же самое относится к выплате кредита за машину. Сумма, которую вы платите за электричество, может меняться, но она, вероятно, будет отличаться от предыдущей не более чем на 10–20%. Эти сведения могут быть использованы программой, чтобы понять, что происходит, и помочь пользователю.

Пороги предпочтений

Решения, принимаемые людьми, можно поделить на две категории: важные и неважные. Любая деятельность потенциально включает в себя сотни решений, но только некоторые из них важны. Все остальные несущественны. Программные интерфейсы могут воспользоваться идеей **порогов предпочтений** для облегчения работы пользователя.

Когда вы приняли решение купить автомобиль, вам все равно, где брать кредит, до тех пор пока условия примерно равные. Когда вы набрали продукты в универсаме, вам все равно, через какую кассу платить. Когда вы решили покататься на американских горках, вам все равно, в какую тележку вас посадят.

Пороги предпочтений являются ориентиром при проектировании пользовательского интерфейса. Они демонстрируют, что вовсе не обязательно расспрашивать пользователя о мелких шагах процедуры. Когда пользователь просит распечатать документ, не нужно задавать ему вопросы относительно количества копий и ориентации страницы. В первый раз мы можем сделать какие-то предположения, а затем использовать их повторно. Если пользователь захочет изменить настройки печати, он всегда сможет вызвать соответствующее диалоговое окно.

Пороги предпочтений позволяют легко отслеживать, какие функциональные возможности пользователь любит настраивать, а какие устанавливает единожды и впоследствии игнорирует. Вооруженное этим знанием, приложение может предложить пользователю выбор из нескольких вариантов, если ожидает, что пользователю это потребуется, не заставляя его при этом принимать решения по вопросам, до которых ему нет дела.

Правильные решения в большинстве случаев

Принцип связности задач позволяет предсказывать действия пользователя с разумной, но не абсолютной точностью. Если наша программа придерживается этого принципа, естественно задаться вопросом о точности ее предсказаний. Если мы можем предсказать поведение пользователя в 80% случаев, это означает, что в 20% случаев мы ошибемся. Может показаться, что в такой ситуации разумно предложить пользователю выбор, но тогда окажется, что мы беспокоим его необязательным диалоговым окном в 80% случаев. Вместо того чтобы предлагать пользователю варианты на выбор, программа должна делать то, что она считает подходящим в данной ситуации, позволяя пользователю отменить или изменить ее действия. Если функцию отмены легко понять и применить, пользователь не будет обеспокоен необходимостью задействовать ее. В конце концов, ему придется обратиться к этой функции два раза из десяти – вместо того чтобы восемь из десяти раз иметь дело с ненужным диалоговым окном. Первый вариант для человека гораздо лучше.

13

Метафоры, идиомы, ожидаемое назначение

Некоторые проектировщики интерфейса говорят о необходимости найти правильную метафору, чтобы на ее основе реализовать поведение интерфейса. Им кажется, что включение в интерфейс изображений узнаваемых объектов из реального мира облегчит пользователю освоение продукта. Так появляются «нарядные» интерфейсы. Один выглядит как офис с рабочими столами, картотеками, телефонами и адресными книгами, другой – как перекидной блокнот, третий изображает улицу и дома. Если вы тоже находитесь в поисках волшебной метафоры, то попадаете в весьма достойную компанию. Многие из лучших и ярчайших специалистов в области проектирования интерфейса считают выбор метафоры своей первоочередной и важнейшей задачей.

Мы полагаем, что такой буквальный подход ограничен и потенциально ошибочен. Строгое следование метафорам без всякой необходимости привязывает интерфейс к реалиям физического мира. Между тем одно из поразительных качеств цифровых продуктов – то, что модель представления не привязана к ограничениям физического мира и присущим трехмерному пространству неудобствам.

Пользовательские интерфейсы, основанные на метафорах, порождают и массу иных проблем. Существует не так уж много удачных метафор, они плохо поддаются расширению, а способность пользователя распознать их часто сомнительна – особенно при пересечении культурных границ. В проектировании большинства цифровых продуктов информационной эры для метафор, особенно физических и пространственных, очень мало места. В данной главе обсуждаются причины такого положения и альтернативы проектированию, основанному на метафорах.

Парадигмы интерфейса

В концептуальном и визуальном проектировании пользовательских интерфейсов существуют три основных парадигмы: **метафорическая**, **идиоматическая** и **парадигма реализации**. Интерфейсы, ориентированные на реализацию, опираются на *понимание* того, как работает продукт, и тем самым ставят перед пользователем непростую задачу. Метафорические интерфейсы основаны на *интуитивных* представлениях о работе продукта и являются рискованным подходом. А вот идиоматические интерфейсы основаны на *обучении* пользователя тому, как достичь результата, что является для людей естественным процессом.

Разработка пользовательских интерфейсов как дисциплина прошла путь от сильной концентрации на технологии (парадигма реализации) до столь же выраженной концентрации на метафоре. В современных интерфейсах ясно просматриваются все три парадигмы, хотя в литературе описана лишь метафорическая. Метафоры являются отличным средством в общении между людьми (в этой книге вы можете найти массу метафор), однако они малопригодны для проектирования программных продуктов и часто препятствуют созданию действительно хороших интерфейсов.

Интерфейсы в парадигме реализации

Пользовательские интерфейсы, ориентированные на модель реализации, широко распространены в компьютерной индустрии. Эти интерфейсы выражены в терминах, отражающих конструкцию продукта. Чтобы успешно взаимодействовать с таким интерфейсом, пользователь должен понимать, как работает программа. Следование парадигме реализации означает, что проектирование пользовательского интерфейса целиком основано на модели реализации.

Подавляющее большинство программ в настоящее время применяют интерфейс в парадигме реализации и без тени стыда демонстрируют нам свое устройство. Они имеют по кнопке на каждую функцию и по диалоговому окну на каждый модуль кода, а их команды и процессы являются точным отражением внутренних структур данных и алгоритмов.

Мы видим, что интерфейс, созданный на основе модели реализации, может функционировать, только обладая знанием о том, как работает программа. К сожалению, справедливо и другое: чтобы пользоваться таким интерфейсом, *мы* должны разобраться в том, как программа работает.

ПРИНЦИП проектирования

Люди предпочитают добиваться успеха, а не всеведения.

Очевидно, что интерфейсы, ориентированные на реализацию, строить проще всего: всякий раз при создании функции программист просто добавляет фрагмент интерфейса, тестирующий эту функцию.

Такой интерфейс легко отлаживать, а когда что-то работает не так, легко выяснять, в чем причина. Инженерам нравится знать, как работает та или иная вещь, и реализационная парадигма их вполне устраивает. Инженеры предпочитают видеть рычаги и шестеренки, потому что это помогает им понять, что происходит внутри машины. Тот факт, что подобные элементы неоправданно усложняют жизнь пользователей, кажется разработчикам несущественным побочным эффектом. Инженеры, возможно, и готовы разбираться в устройстве продукта, но большинство пользователей не имеют для этого ни времени, ни желания. Успех интересует их больше, чем знания, и такое предпочтение непонятно большинству инженеров.

Стоит упомянуть и близкий к интерфейсу в парадигме реализации вариант «интерфейса, ориентированного на структуру организации». Часто возникает ситуация, когда организация продукта (как правило, веб-сайта) соответствует не тому, как пользователи представляют себе информацию, а тому, как структурирована компания, создающая сайт. На таком сайте обычно имеется по вкладке на каждое подразделение, и ощущается острая нехватка связи страниц друг с другом. Подобно продукту, интерфейс которого ориентирован на реализацию, веб-сайт, ориентированный на структуру организации, требует, чтобы пользователи, желающие найти какую-либо информацию, до тонкостей понимали внутреннее устройство организации.

Метафорические интерфейсы

Метафорические интерфейсы полагаются на интуитивные связи, которые пользователь устанавливает между визуальными элементами интерфейса и его функциональностью. От пользователя не требуется знания основ программирования – и это шаг вперед по сравнению с моделью интерфейса в парадигме реализации. Впрочем, эффективность и полезность такого подхода преувеличена до нереалистичных размеров.

Говоря о метафорах в контексте пользовательского интерфейса и проектирования взаимодействия, в действительности мы подразумеваем визуальные метафоры – картинки, которые служат представлением атрибутов или назначения объектов. Пользователи воспринимают передаваемый метафорой образ и предположительно могут расширить его до понимания предназначения объекта. Метафоры могут охватывать широчайший диапазон: от крохотных пиктограмм на панели инструментов до полноэкранных изображений в некоторых программах и от изображения ножниц на кнопке, обозначающей операцию «Вырезать», до огромной чековой книжки в интерфейсе Quicken. Мы воспринимаем метафоры интуитивно, однако что означает слово «интуитивный» на самом деле?

Толковый словарь русского языка С. И. Ожегова (1984) дает следующее определение интуиции:

> **ин·туи́·ция** *ж.* **1.** Чутье, тонкое понимание, проникновение в самую суть чего-н. <...>; **2.** В философии: непосредственное постижение истины без предварительного логического рассуждения.

Это определение подчеркивает волшебные качества интуиции, но не объясняет, *как* мы ею пользуемся. Интуиция действует на основе предположений. Мы видим связь между разными объектами и обращаем внимание на их сходство, не отвлекаясь на различия. Мы схватываем значения метафорических элементов интерфейса, поскольку мысленно связываем их с понятиями, которые усвоили ранее. Это эффективный путь, позволяющий использовать поразительную способность человека к умозаключениям. Однако этот метод также полагается на особенности стиля мышления пользователей, которые могут не знать языка, не обладать знаниями или не иметь мыслительных способностей, необходимых для установления связей между метафорой и обозначаемой сущностью.

Ограничения метафор

Ошибочно думать, что метафоры являются надежным фундаментом для проектирования пользовательского интерфейса. Это сродни попытке объявить дискеты лучшим носителем информации на том основании, что множество хороших программ когда-то распространялось на дискетах. Метафоры имеют множество ограничений, когда речь идет о современных информационных системах.

Во-первых, метафоры плохо масштабируются. Метафора, уместная для обозначения простого процесса в маленькой программе, часто дает сбой, когда процесс усложняется. Крупные пиктограммы файлов были хорошей идеей во времена, когда в компьютерах использовались дискеты и 10-мегабайтные жесткие диски с парой сотен файлов. Сейчас, когда на дисках объемом 250 Гбайт хранятся десятки тысяч файлов, подобные пиктограммы стали слишком неуклюжими для эффективной работы.

Кроме того, метафоры предполагают сходство ассоциативного мышления у разработчика и пользователя. Если пользователь и проектировщик относятся к разным культурам, метафора может и не работать. Непонимание может возникнуть даже между представителями близких культур или в рамках одной культуры. Что означает изображение самолета – «расписание вылетов и прибытий самолетов» или «заказ авиабилетов»?

Наконец, хотя метафора предлагает новичку способ быстрее освоиться с интерфейсом, она превращается в непомерную обузу, когда бывший новичок переходит в середняки. Отражая физический мир механизмов, большинство метафор концептуально пригвождают пользователей

к земле, навсегда ограничивая возможности программного продукта. Этот аспект применения метафор еще будет обсуждаться в этой главе.

Наше определение интуиции показывает, что рациональное мышление не используется в процессе интуитивного постижения. В компьютерной индустрии и особенно среди проектировщиков пользовательских интерфейсов выражение «интуитивно понятный» часто означает «простой в использовании» или «простой для понимания». Простота использования, очевидно, важна, но она не является поводом относить успех проектировщиков на счет неких метафизических явлений – равно как и сбрасывать со счетов точное значение слова «интуиция». Существуют конкретные причины, почему люди понимают одни интерфейсы и не понимают другие.

Интуиция, инстинкт и обучение

Существуют звуки, запахи и образы, которые вызывают у нас определенную реакцию, не выработанную в результате предварительного сознательного обучения. Когда маленький ребенок видит разъяренную собаку, он *инстинктивно* и без всякого обучения знает, что оскаленные клыки представляют собой большую опасность. Механизмы распознавания таких ситуаций находятся глубоко в подсознании. Инстинкт – это реакция, реализованная, так сказать, «на аппаратном уровне». Он не предполагает размышлений. Интуиция находится на одну ступеньку выше, потому что хотя она также не требует размышлений, но опирается на сумму знаний, усвоенных сознательно.

Примеры инстинктивной реакции при взаимодействии человека с компьютером: мы вздрагиваем и нервничаем, если на экране происходит резкая смена изображения, наш взгляд неизбежно притягивается к рекламе, мерцающей на веб-странице. На внезапный шум или запах дыма, доносящийся из системного блока компьютера, мы тоже реагируем инстинктивно.

Интуиция занимает промежуточную территорию между сознательным изучением и инстинктивным знанием. Если мы на опыте узнали, что предметы, раскаленные докрасна, могут нас обжечь, мы будем склонны классифицировать все подобные предметы как потенциально опасные, пока опыт не убедит нас в обратном. Конкретный предмет, сияющий красным, может и не представлять угрозы, но мы начнем его исследовать с определенной осторожностью.

То, что мы обычно называем интуицией, фактически является мысленным сравнением нового опыта с ранее полученными знаниями. Например, мы моментально догадываемся, что означает пиктограмма мусорной корзины, потому что знаем, что такое реальная корзина для мусора. Наш мозг был подготовлен к установлению такой связи много лет назад. Однако пользоваться реальной корзиной мы научились *не интуитивно*, хотя и без всякого труда. Так мы приходим к третьему типу интерфейса, основанному на том факте, что человеческий мозг

проявляет невероятные способности к обучению, постоянно и без усилий приобретая новые знания.

Идиоматические интерфейсы

Идиоматическое проектирование, которое Тед Нельсон (Ted Nelson) назвал «проектированием принципов», основано на том, как человек изучает и применяет идиомы и речевые обороты, такие как «свой в доску» или «тертый калач». Идиоматические пользовательские интерфейсы решают проблемы, с которыми не справляются предыдущие два типа интерфейсов, поскольку сосредоточиваются не на технических знаниях и не на интуитивных представлениях о функциях, а на изучении простых, неметафорических визуальных и поведенческих идиом, необходимых пользователю для достижения целей и решения задач.

Идиоматические выражения, в отличие от метафор, не провоцируют создание ассоциативных связей. Нет никаких досок и калачей. Это лишь способы описывать качества людей. Мы воспринимаем идиому просто потому, что выучили ее и она имеет конкретный смысл, а не потому, что понимаем ее или она вызывает подсознательные ассоциации в нашем мозгу. В то же время мы способны быстро запоминать и правильно применять идиомы, не всегда отдавая себе в этом отчет.

Идиому нельзя понять ни интуицией, ни рассудком. Наш язык полон идиом, кажущихся бессмысленными, если мы про них не узнали заранее. Если я скажу «этот парень слетел с катушек», вы поймете, что я имею в виду, даже если никаких катушек у несчастного и в помине не было. Понять это выражение путем размышлений или экспериментов невозможно. Вы понимаете его из контекста или где-то прочитали, что оно означает, или вас этому кто-то научил. Вы понимаете смысл выражения, связывающего катушки, полеты и безумие, просто потому, что люди хорошо запоминают подобные вещи.

Человеческий мозг обладает поистине удивительной способностью к изучению и запоминанию большого количества идиом, причем делает это легко и просто, обходясь без параллелей со знакомыми ситуациями и без понимания «внутреннего устройства» идиомы. Это необходимость, поскольку большинство идиом лишены метафорического значения, а происхождение остальных теряется в прошлом.

Графические интерфейсы – в основном идиоматические

Оказывается, большинство элементов интуитивно понятного графического интерфейса в действительности представляют собой визуальные идиомы. Окна, заголовки, кнопки закрытия, разделители экрана, гиперссылки и раскрывающиеся списки – все эти объекты мы впервые узнали как идиомы, а не постигли интуитивно, как метафоры. Применение изображения мусорного бака на компьютерах Macintosh для размонтирования внешнего FireWire-диска перед его отключением – в чистом виде идиома (причем, по мнению многих проектировщиков,

неудачная), хотя сама по себе картинка является визуальной метафорой.

Вездесущая мышь как устройство ввода отнюдь не метафорична. Ее мы тоже приняли как идиому. В фильме «Звездный путь IV» (Star Trek IV) есть сцена, в которой Скотти попадает на Землю XX века и пытается говорить в мышь как в микрофон. Во внешности мыши нет ничего, что указывало бы на ее предназначение или способ применения, и она не похожа ни на что из нашего прошлого опыта, так что постижение этого устройства не опирается на интуицию. Тем не менее научиться пользоваться мышью проще простого. Достаточно трех секунд, чтобы показать новичку, что к чему, – и он на всю жизнь запомнит урок. Мы не знаем и не хотим знать, как мышь устроена, но даже малые дети справляются с ней. Это и есть идиоматическое освоение.

По иронии судьбы, многие знакомые нам элементы графического пользовательского интерфейса, часто принимаемые за метафоры, на деле являются идиомами. Окна с изменяемым размером и бесконечно вложенные файловые каталоги не метафоричны, поскольку у них нет аналогов в реальном мире. Вся их сила как раз в простоте идиоматического освоения.

Хорошую идиому достаточно изучить только один раз

Мы склонны думать, что изучать интерфейсы трудно, поскольку основываемся на опыте работы с интерфейсами, созданными по модели реализации. Эти интерфейсы очень сложны в изучении, потому что эффективная работа с ними требует от пользователя понимания того, как работает программа. Большая часть того, что мы знаем, усвоена нами *без понимания*: лица людей, общественные и личные отношения, мелодии, названия брендов, расположение комнат и мебели в нашем доме или офисе. Мы не *понимаем*, почему чье-то лицо выглядит так, а не иначе; мы просто *знаем* это лицо. Мы отличаем его от других, потому что посмотрели на него и автоматически (причем с легкостью) запомнили.

Все идиомы необходимо изучать; хорошую идиому достаточно изучить только один раз.

Главное наблюдение, связанное с идиомами, состоит в том, что, хотя их приходится изучать, делать это весьма просто, а хорошие идиомы достаточно изучить один раз. Мы без труда запоминаем такие идиомы, как «встать не с той ноги» или «попасть впросак». Человеческий мозг способен запоминать подобные идиомы с первого раза. Столь же легко он может осваивать идиомы интерфейса – выключатели, кнопки закрытия окон, раскрывающиеся меню и комбо-списки.

Идиомы и бренды

Профессионалы в сфере маркетинга и рекламы хорошо понимают силу простого действия или символа, наделенного значением. В конечном счете синтез идиом есть суть процесса создания бренда. Компания берет продукт или название и наделяет его желаемым смыслом. Золотые арки McDonalds, три брильянта Mitsubishi, пять переплетенных олимпийских колец, наконец, летящие окна Microsoft – все это неметафорические идиомы, моментально узнаваемые и наделенные общеизвестным смыслом. Рисунок 13.1 демонстрирует силу идиоматического символа.

Рис. 13.1. *Этот идиоматический символ наделен смыслом благодаря практике его применения, а не связи с другими объектами. У каждого, чье детство прошло в 50-е и 60-е годы, этот объективно бессмысленный символ способен вызвать страх, поскольку он обозначает атомную радиацию. Визуальные идиомы (например, американский флаг) могут быть столь же сильны, как и метафоры, или даже сильнее. Источником этой силы служит то, как мы ими пользуемся и какой смысл им придаем, а не естественная связь с реальными объектами*

Еще об ограничениях метафор

Если при создании пользовательского интерфейса мы полагаемся на метафоры, то сталкиваемся не только с уже упоминавшимися мелкими проблемами, но и с двумя более серьезными. Метафоры трудно подбирать, и они ограничивают наше мышление.

Подбор хорошей метафоры

Нетрудно найти визуальные метафоры для физических объектов вроде принтеров и документов. Гораздо труднее или вообще невозможно подобрать метафору для процессов, отношений, служб и преобразований – понятий, чаще всего ассоциируемых с программными продуктами. Весьма проблематично найти визуальную метафору для переключения каналов, приобретения вещи, поиска литературы, задания формата, изменения разрешения изображения и выполнения статистического анализа, однако именно такие операции чаще всего выполняются с использованием программ.

Сила компьютеров и цифровых продуктов – в их умении справляться с невероятно сложными отношениями внутри огромных наборов ин-

формации. Весь смысл их существования как раз в том, что человеческому мозгу сложно работать с подобными многомерными структурами, так что эти процессы – по своему смыслу – не очень сочетаются с простыми физическими аналогами, понимание которых человеку дается «автоматически».

Проблемы глобальных метафор

Самой же значимой проблемой, возникающей в связи с метафорами, является тот факт, что они привязывают интерфейсы к объектам механической эры. Ярким примером является Magic Cap – интерфейс портативного коммуникатора, выпущенный и разрекламированный в середине 90-х годов фирмой General Magic. Практически каждый аспект этого интерфейса опирается на метафоры. Пользователь получает доступ к сообщениям в почтовом ящике или блокноте на столе. Он проходит по коридору вдоль дверей, за которыми скрываются второстепенные функции. Пользователь выходит на улицу, чтобы получить доступ к услугам третьих фирм, которые, как видно из рис. 13.2, представлены зданиями. Пользователь может войти в здание, чтобы настроить службу, и так далее. Жесткая привязка к этой метафоре подразумевает, что пользователь интуитивно догадается о функционировании этого программного продукта. Минус такого подхода – как только пользователь освоился с программой, метафора начинает мешать

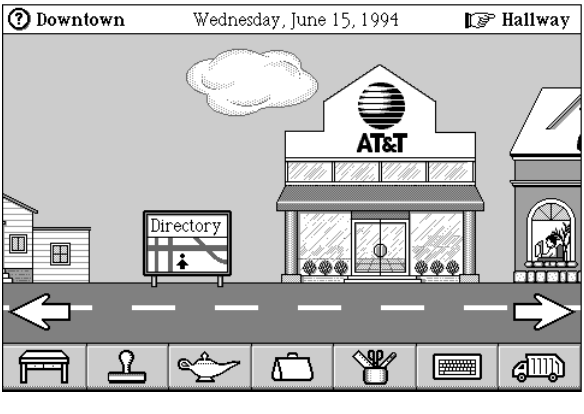


Рис. 13.2. *Интерфейс Magic Cap от General Magic использовался в продуктах Sony и Motorola в середине 90-х. Это была демонстрация силы в области метафорического проектирования. Вся навигация и большая часть взаимодействия были подчинены сверхидее пространственных и физических метафор. Безусловно, проектировщикам было интересно создавать такой интерфейс, но вряд ли он доставлял удовольствие пользователям, достигшим среднего уровня. Все это очень досадно, поскольку некоторые низкоуровневые неметафорические взаимодействия, связанные с вводом данных, были довольно сложны и хорошо реализованы для того времени*

навигации. Пользователь *вынужден* выйти на улицу, чтобы настроить другую службу. Он *вынужден* пройти по коридору до дверей игровой комнаты, чтобы разложить пасьянс. Это нормально в физическом мире, но нет причины вынуждать пользователя к такому поведению в мире виртуальном. Почему бы не отказаться от рабского следования метафоре и не предложить пользователю *простой* доступ к функциям? Оказывается, программисты General Magic впоследствии создали сочетание клавиш для управления вкладками и оформили эту возможность в виде неуклюжего дополнительного модуля, – но, увы, поезд уже ушел.

Интерфейс General Magic был основан на так называемой **глобальной метафоре**. Это одна всеобъемлющая метафора, представляющая собой каркас для всех других метафор системы. Изначальный рабочий стол Macintosh также является глобальной метафорой.

Скрытая проблема глобальных метафор – ошибочное убеждение в том, что метафоры низших уровней, связанные с глобальной, уже благодаря одной этой связи получают преимущества. Искушение растянуть метафору за пределы простого обозначения функции неодолимо – и вот появляется программа дозвона, позволяющая пользователям набирать номера с помощью кнопок – таких же, как на настоящих телефонах. Появляются программы, в которых записные книжки выглядят точно так же, как записные книжки в сумочках и карманах у людей. Не лучше ли выйти за рамки этих технологий промышленного века и воспользоваться реальными возможностями современных компьютеров? Почему бы коммуникационным программам не разрешать конференционные звонки и не осуществлять дозвон по названию фирмы (или филиала)? В конце концов, хотя бы просто скрывать использование телефонных номеров?

Возможно, кому-то кажется разумным представить функцию дозвона изображением телефона на столе, но такой подход фактически лишает проектировщика свободы. Создатели первых телефонов были бы счастливы, если бы им удалось сконструировать аппарат, позволяющий позвонить другу простым указанием на картинку с его портретом. К сожалению, они были ограничены тоскливой реальностью электрических цепей и бакелитового прессования. Зато сегодня мы можем позволить себе роскошь представлять коммуникационные интерфейсы в любом виде. Портреты друзей вполне уместны в таком интерфейсе – так зачем отказываться от них ради представления устаревшей технологии?

С расширением метафор связаны две ловушки – одна для пользователя, другая для проектировщика. Полагаясь на легкоузнаваемость метафоры, пользователь ожидает, что поведение виртуального объекта будет соответствовать поведению реального, на который метафора ссылается. Здесь проектировщика подстерегает ловушка. Он испытывает искушение представить программный продукт в терминах этого физического объекта (представителя механической эпохи, между прочим), чтобы соответствовать ожиданиям пользователя. Однако, как мы виде-

ли в главе 2, перенос механических процессов в компьютер обычно ухудшает их.

Возьмем для примера вездесущую «папку» в современных компьютерных операционных системах. Как механизм для организации документов ее легко понять и освоить, поскольку она напоминает реальную папку из картотеки. К сожалению, как и многие другие метафоры пользовательского интерфейса, она ведет себя не совсем так, как физический аналог, а это уже – повод для когнитивного сопротивления со стороны пользователей. Так, в мире бумажных папок не найти десятикратной вложенности, и потому новичкам тяжело освоиться с навигационными структурами операционных систем.

Реализация этого механизма также дает неприятные ограничения. В мире бумаги один документ не может находиться сразу в двух ящиках или в двух папках в картотечном шкафу – и, как следствие, занесение папки в шкаф следует единой схеме, принятой в организации (по алфавиту на основе имен или по номерам учетных записей). Изначально наши цифровые продукты не имеют таких ограничений, но слепое стремление к метафоре в интерфейсе резко ограничивает нашу способность хранить документ согласно сразу нескольким схемам организации.

Как сказала Бренда Лорел: «Метафоры интерфейса громыхают, словно машины Руба Голдберга[1]: их латают и подвязывают шнурочками при каждой поломке, пока следы ремонта не изменят их настолько, что мы перестанем понимать их назначение и происхождение». Нас удивляет, что проектировщики, у которых есть возможность создать интерфейс для «телефона мечты», предлагают нам все тот же старый телефонный аппарат только потому, что их научили, что сильная глобальная метафора является необходимым атрибутом хорошего пользовательского интерфейса. Из всех заблуждений, порожденных центром Xerox PARC, миф о глобальной метафоре является самым печальным и разрушительным.

Идиоматическое проектирование – это будущее проектирования взаимодействия. Следуя этой парадигме, мы опираемся на естественную человеческую способность к обучению и не вынуждаем пользователя разбираться в том, как работает программа. Бесконечное количество идиом находятся в ожидании своих изобретателей – и лишь ограниченное количество метафор ждут момента, чтобы их кто-нибудь нашел. Метафоры предлагают новичкам грошовый выигрыш при обучении, зато

[1] Руб Голдберг (Rube Goldberg) – американский карикатурист, известный тем, что рисовал механизмы, которые позволяют выполнить простые действия невероятно сложным способом. Ему приписывается примечательная фраза: «Есть два способа достижения цели – простой и трудный. И вот что удивительно: большинство людей всегда выбирают именно тот, что сложнее». – *Примеч. науч. ред.*

впоследствии оборачиваются заметными затратами. Лучше всего проектировать на основе идиом, применяя лишь самые мощные и подходящие метафоры.

Используйте метафоры, если найдете, но не подгоняйте свой интерфейс под произвольный метафорический стандарт.

Никогда не подгоняйте интерфейс под метафору.

Macintosh и метафоры: новый взгляд

Современный графический пользовательский интерфейс был изобретен в середине 70-х в исследовательском центре Xerox PARC (Palo Alto Research Center). В том виде, как его создали разработчики фирмы Xerox, интерфейс состоял из многих элементов – окон, кнопок, мышей, пиктограмм, визуальных метафор и раскрывающихся меню. Все вместе они сформировали незыблемую основу для последующих разработок в области проектирования интерфейсов, поскольку этот ансамбль доказал свое превосходство на практике.

Первой коммерчески удачной реализацией пользовательского интерфейса PARC стал компьютер Apple Macintosh с его метафорой рабочего стола: корзина для мусора, перекрывающиеся листы бумаги (окна) и папки с файлами. Успех пришел к Macintosh не благодаря этим метафорам, а по ряду других причин, в число которых входило внимательное отношение к дизайну и деталям. Вот те удачные решения проектировщиков, которые работали на успех системы:

- Весьма ограниченный, но гибкий лексикон пользовательского взаимодействия с приложениями, основанный на очень простом наборе операций с мышью.
- Возможность сложных непосредственных манипуляций с информационно насыщенными объектами на экране.
- Использование высокого разрешения с квадратными пикселами, которое позволило приблизить экранное изображение к распечатке на бумаге, особенно в связи с появлением нового продукта фирмы Apple – лазерного принтера.

Метафоры позволили структурировать эти важные интерфейсные решения и явились удачным маркетинговым ходом, но никак не главной причиной успеха. В действительности первые годы были для Macintosh довольно трудными, поскольку пользователям требовалось время, чтобы привыкнуть к новому, визуальному способу общения с компьютером. Производители программного обеспечения тоже поначалу побаивались разрабатывать программы для такого радикально иного окружения (фирма Microsoft была исключением).

Впрочем, люди в конце концов сдались: ведь эта система *могла* делать то, что другим системам было не под силу, – представлять информацию на экране методом WYSIWYG (what you see is what you get – «что видишь, то и получишь»). Сочетание наглядных интерфейсов с высококачественной печатью (на принтерах LaserWriter) создало совершенно новый рынок, на котором Apple с Macintosh доминировала много лет. Метафора в успехе Macintosh была несущественным фактором.

Построение идиом

Когда графические пользовательские интерфейсы впервые вышли на сцену, они были настолько лучше всех прочих, что многие наблюдатели приписывали успех самой идее *графического* интерфейса. Это было естественное, однако некорректное предположение. Первые графические интерфейсы, например оригинальный интерфейс Macintosh, были лучше других в основном потому, что их графическая природа требовала ограничения лексикона, посредством которого пользователи взаимодействуют с системой. В частности, способ ввода информации пользователем эволюционировал от набора произвольной команды к строго ограниченному набору операций с мышью. В интерфейсе командной строки пользователь может ввести любую комбинацию символов – практически бесконечное число вариантов. Чтобы ввод был корректным, пользователь должен точно знать, чего ожидает программа. Он должен помнить сочетания букв и символов с абсолютной точностью. Последовательность символов (а иногда и их регистр) играет решающую роль.

В современных графических пользовательских интерфейсах пользователю достаточно указывать курсором мыши на изображения и слова на экране. Имеющийся выбор в большинстве случаев отражает мыслительные процессы пользователя, поэтому исчезает необходимость что-либо заучивать. При помощи кнопок мыши пользователь может выполнять щелчки или двойные щелчки по объектам интерфейса, а также перетаскивать их. Клавиатура применяется для ввода данных и обычно не используется для подачи команд и навигации. Число неделимых элементов в лексиконе пользователя снизилось с десятков (если не сотен) до всего лишь трех, хотя диапазон задач, решаемых при помощи графических программ, не уже, чем у систем с командной строкой.

Чем глубже уровень элементов, составляющих лексикон взаимодействия, тем длительнее и сложнее процесс их изучения. Приобретение словарного запаса для общения на английском языке занимает у человека минимум десять лет, а сложность языка требует постоянной практики для поддержания уровня беглого общения. Зато для тех, кто им овладел, этот язык является исключительно выразительным средством. Ограничение количества элементов лексикона взаимодействия уменьшает его выразительность на низком уровне. Однако более сложное взаимодействие может быть легко построено на базе неделимых

элементов, подобно тому, как буквы образуют слова, а слова формируют предложения.

Правильно составленный лексикон взаимодействия можно изобразить в виде перевернутой пирамиды. Все простые в освоении системы коммуникаций соответствуют модели, изображенной на рис. 13.3. Нижний уровень содержит **примитивы** – базовые строительные блоки языка. В современных графических пользовательских интерфейсах такими примитивами являются наведение указателя мыши на объект, щелчок и перетаскивание.

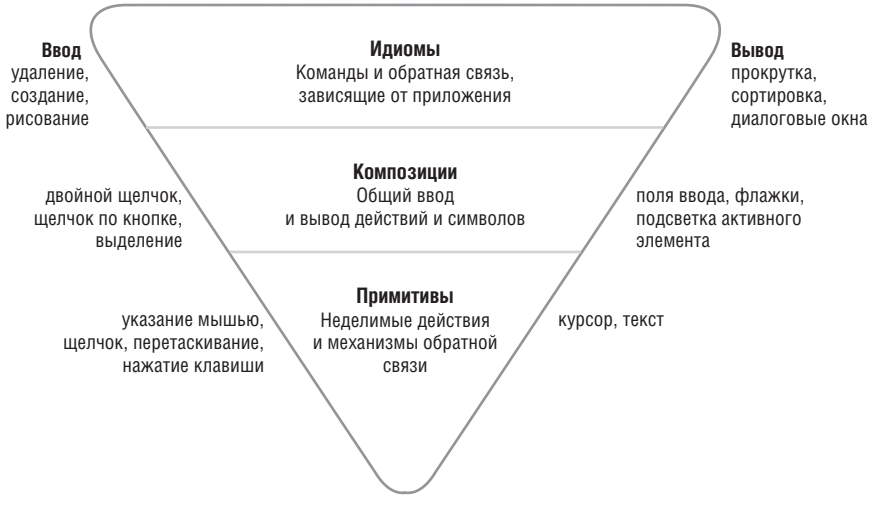


Рис. 13.3. *Одной из основных причин простоты использования графических пользовательских интерфейсов является применение ограниченного лексикона взаимодействия, в котором сложные идиомы опираются на очень небольшой набор примитивов – наведение указателя мыши на объект, щелчок и перетаскивание. Эти примитивы позволяют составить широкий набор простых композиций, которые, в свою очередь, образуют сложные идиомы предметной области, и вся эта конструкция покоится на небольшом наборе действий, которым нетрудно научиться*

Средний уровень содержит **композиции** – более сложные конструкции, состоящие из одного или нескольких примитивов. Сюда входят простые визуальные объекты, на которые воздействует пользователь (например, текст), действия (двойной щелчок и перетаскивание) и объекты манипуляции (например, кнопки, флажки, гиперссылки и маркеры непосредственной манипуляции).

Верхний уровень содержит **идиомы**. Идиомы являются результатом сочетания и структурирования композиций с учетом знаний о **предметной области** рассматриваемой проблемы, то есть информации о сфере деятельности и задачах пользователя, а не о компьютерной реализа-

ции решения. Набор идиом образует лексикон, относящийся к информации о конкретной задаче, решаемой с помощью программы. В графическом пользовательском интерфейсе к идиомам относятся такие элементы, как подписанные кнопки и поля ввода, панели навигации, списки, пиктограммы и даже группы полей и элементов управления или целые панели и диалоговые окна.

Любой язык, не следующий такому принципу построения, изучать довольно трудно. Многие эффективные системы коммуникации за пределами мира компьютеров используют сходные лексиконы. Дорожные знаки в США придерживаются простой схемы цветов и форм. Желтые треугольники – предупреждение, красные восьмиугольники – приказы, а зеленые прямоугольники – информация.

В обмене текстовыми сообщениями посредством телефона нет ничего интуитивного или метафорического. Композиционное взаимодействие – поочередное нажатие на цифровые клавиши для набора алфавитных символов, причем в правильной последовательности – требует обучения, а в сочетании с предиктивным вводом текста является невероятно эффективной идиомой для отсылки коротких текстов с мобильного телефона.

Ожидаемые физические назначения

В своей новаторской работе «The Design of Everyday Things»[1] (Norman, 1989) Дональд Норман ввел термин **ожидаемые назначения** (affordances), который он определяет как «воспринимаемые и фактические качества объекта, преимущественно фундаментальные, которые определяют возможные способы обращения с этим объектом».

Это бесценное для практики проектирования интерфейсов определение. Однако для наших целей не хватает ответа на один ключевой вопрос: *как* мы узнаем, что именно эти назначения предлагают нам? Если мы смотрим на объект и понимаем, как им пользоваться (то есть понимаем его ожидаемые назначения), то мы, должно быть, применяем какой-то метод выявления взаимосвязей.

Поэтому мы предлагаем изменить определение Нормана, а точнее – убрать из него слова «и фактические». После этого термин «ожидаемые назначения» становится чисто когнитивным – он говорит не о реальных возможностях объекта, а о том, что мы *думаем* об этих возможностях. Если на стене рядом с дверью дома расположена кнопка, ее ожидаемым назначением стопроцентно является то, что это кнопка звонка. Если после ее нажатия у нас под ногами откроется люк и мы

[1] Русский перевод выходил под двумя разными названиями: Д. Норман «Дизайн промышленных товаров». – Пер. с англ. – М.: Вильямс, 2008; Д. Норман «Дизайн привычных вещей». – Пер. с англ. – М.: Вильямс, 2006. – *Примеч. ред.*

провалимся под землю, то это не дверной звонок. Тем не менее ожидаемое назначение кнопки останется прежним.

Но почему мы решили, что перед нами кнопка звонка? Просто потому, что так нам подсказывает весь наш богатый опыт взросления и социализации, наши знания об этикете и звонках. Мы знаем, что подобные предметы связаны с электрическими и электронными устройствами, потому что когда-то, много лет назад, стояли на пороге чужого дома вместе с родителями и усвоили, как попасть внутрь.

Но у обсуждаемого вопроса есть и иной аспект. Если мы увидим кнопку в необычном месте, например на капоте автомобиля, мы будем теряться в догадках о ее назначении, но при этом все же догадаемся, что на нее следует нажать пальцем. Как мы пришли к этой догадке? По своей природе человек склонен манипулировать инструментами. Когда мы видим предмет подходящего размера и формы, расположенный в пределах досягаемости, мы автоматически хотим нажать на него пальцем. Длинные предметы, округлые в сечении, вызывают у нас желание ухватиться за них, как за ручку. Все это подразумевал Норман, вводя свой термин. Однако, чтобы внести ясность, мы будем называть инстинктивное понимание того, как обращаться с предметом руками, **ожидаемыми физическими назначениями**. Когда объекты имеют форму, очевидно подходящую для непосредственных манипуляций ногами или руками, мы без письменных инструкций понимаем, что следует с ними делать. Понимание того, как обращаться с инструментом, форма которого соответствует форме человеческой руки, является прекрасным примером интуитивного постижения интерфейса.

В своей работе Норман подробно обсуждает действенность ожидаемых [физических] назначений по сравнению с письменными инструкциями. В качестве типичного примера он приводит дверь, которую следует открывать, толкая металлическую ручку. Ручка имеет такую форму и расположение и столь удобна для захвата, что весь ее вид кричит: «Тянуть на себя!». Неважно, как часто человек проходит через эту чертову дверь, – он всякий раз будет пытаться тянуть за ручку, потому что ожидаемое назначение ручки перевешивает любое количество табличек «От себя».

Существует не так уж много ожидаемых физических назначений. Мы тянем на себя предметы, которые удобно захватить кистью (или зацепить пальцами, если они малы). Мы толкаем плоские предметы рукой (или нажимаем указательным пальцем). Если они расположены на полу, нажимаем ногой. Мы поворачиваем круглые предметы пальцами (как ручку громкости) или обеими руками, если предметы достаточно велики (как рулевое колесо). Такие ожидаемые физические назначения являются базой для проектирования визуального пользовательского интерфейса.

Популярный псевдотрехмерный дизайн таких систем, как Windows, Mac OS и Motif, основывается на затенении и подсветке с целью прида-

ния более объемного вида объектам на экране. Нашему сознанию, привыкшему к манипулированию инструментами, эти объекты предлагают свои виртуальные ожидаемые физические назначения в виде похожих на кнопки изображений, всем видом говорящих: «Нажми меня!».

Семантика ожидаемых физических назначений

Чего не хватает в объектах с чисто виртуальными ожидаемыми назначениями, так это идеи о том, какую функцию они выполняют. Мы видим, что перед нами кнопка, но как узнать, что произойдет, когда мы нажмем на нее? В отличие от механических объектов, мы не можем выяснить функцию виртуального рычага, проследив его связь с другими механизмами, – изучить программу подобным способом не так-то просто. Вместо этого нам приходится полагаться на сопроводительный текст и изображения или – чаще всего – на свой предыдущий опыт и имеющиеся знания. Ясно, что ожидаемые назначения полосы прокрутки связаны с возможностью манипулировать ею, но единственными конкретными указателями являются стрелки на кнопках, намекающие на направленность действия этого элемента. Чтобы узнать, что полоса прокрутки определяет позицию в документе, нам необходимо либо пройти обучение, либо поэкспериментировать.

Элементы управления должны нести на себе текстовые или графические метки, раскрывающие их предназначение. Если функция элемента управления не описана прямо на нем, у нас остаются два пути – эксперимент или обучение. Либо мы обращаемся за помощью к литературе или знакомым, либо действуем методом проб и ошибок. Здесь инстинкт и интуиция бессильны помочь нам. Остается полагаться на эмпирические методы.

Как оправдать ожидания пользователей в отношении физических назначений

В реальном мире объект выполняет свою функцию благодаря определенной физической форме и сочетанию с другими физическими объектами. Пила пилит дерево, потому что она острая и плоская и снабжена ручкой. Дверная рукоятка открывает дверь, потому что соединена с защелкой. Однако в виртуальном мире объект выполняет ту или иную функцию, потому что такой способностью его наделил программист. Мы можем многое узнать о пиле или дверной ручке простым физическим осмотром, и их внешний вид нас не обманет. На экране компьютера мы видим прямоугольник, кажущийся трехмерным, на который явно можно нажать, как на кнопку. Но это не обязательно указывает на то, что на него *нужно* нажать. Ведь результат может быть в буквальном смысле почти любым. Здесь нас можно ввести в заблуждение, поскольку (в отличие от реального мира) отсутствует естественная связь между тем, что мы видим, и тем, что кроется за внешним видом. Иными словами, мы можем не знать, как работать пилой, и мы даже

можем расстраиваться от неумения обращаться с нею, однако она никогда нас не обманет. В ее внешности нет ничего, что не соответствовало бы ее функциональности. А вот создать фальшивку на экране компьютера совсем нетрудно.

Когда разработчик изображает на экране кнопку, он заключает с пользователем договор о том, что внешний вид кнопки изменится от щелчка по ней: она «продавится». Далее – гласит договор – эта кнопка выполнит некое разумное действие в точном соответствии с надписью на ней. Все это звучит как само собой разумеющееся, но поразительно, сколь многие программные продукты поступают с пользовательскими ожиданиями по схеме «заманил и бросил». Это относительно редко происходит с кнопками, но более чем справедливо в отношении других элементов управления, особенно на многочисленных веб-сайтах, где из-за недостатка ожидаемых назначений трудно различить элементы управления, содержимое и украшательства. Убедитесь, что ваша программа удовлетворяет ожидания, которые вызывает у пользователей посредством ожидаемых физических назначений.

14

Визуальный дизайн интерфейсов

Сколько бы сил вы ни вложили в исследование пользователей и создание модели поведения продукта, способствующей достижению их целей, силы эти будут потрачены впустую, если вы не сумеете должным образом донести до пользователей принципы этого поведения. В случае интерактивных продуктов это часто делается визуальными средствами – путем отображения объектов на дисплее (хотя в некоторых случаях поведение продукта приходится доносить посредством физических свойств, таких как форма аппаратной кнопки или тактильные ощущения от нее).

Визуальный дизайн интерфейсов – дисциплина, которую из-за сходства с графическим дизайном и изобразительными искусствами часто воспринимают неправильно. Нередко ее неверно определяют как «наложение шкурки» на интерфейс; нам даже доводилось слышать такую формулировку, как «украшательство продукта».

Наш опыт привел нас к выводу, что визуальный дизайн интерфейсов – очень нужная и уникальная дисциплина, которую следует применять в сочетании с проектированием взаимодействия и промышленным дизайном. Она способна серьезно повлиять на эффективность и привлекательность продукта, но для полной реализации этого потенциала нужно не откладывать визуальный дизайн на потом (иначе получатся попытки «накрасить свинью»), а сделать его одним из основных инструментов удовлетворения потребностей пользователей и бизнеса.

Разработка визуального дизайна интерфейса требует наличия ряда сопряженных навыков. Конкретный набор навыков определяется создаваемым продуктом. Чтобы создавать привлекательные и удобные пользовательские интерфейсы, дизайнер интерфейса должен владеть базовыми визуальными навыками – пониманием цвета, типографики, формы и композиции – и знать, как их можно эффективно применять для передачи поведения и представления информации, для создания

настроения и стимулирования физиологических реакций. Дизайнеру интерфейса также требуется глубокое понимание принципов взаимодействия и идиом интерфейса, определяющих поведение продукта.

В этой главе мы обсудим эффективные стратегии визуального проектирования интерфейса. В третьей части книги содержится дополнительная информация о конкретных интерактивных, а также интерфейсных идиомах и принципах.

Изобразительное искусство, визуальный дизайн интерфейсов и прочие дисциплины дизайна

Художники и визуальные дизайнеры работают с одними и теми же изобразительными средствами. И те, и другие должны быть искусны и опытны во всем, что касается этих средств, но их деятельность служит различным целям. Цель художника – создать объект, взгляд на который вызывает эстетический отклик. Таким образом, изобразительное искусство – способ самовыражения художника на тему, которая у него (а иногда и у общества в целом) вызывает эмоциональный или интеллектуальный интерес. Художник не связан почти никакими ограничениями. Чем необычнее и своеобразнее продукт его усилий, тем выше он ценится.

Дизайнеры, напротив, создают объекты для *других людей*. В то время как современные художники озабочены в основном *самовыражением*, дизайнеры, как отмечают Кевин Маллет и Даррел Сано в своей великолепной книге «Designing Visual Interfaces» (Mullet and Sano, 1995), «заняты поисками наиболее подходящего *представления* для передачи некоторой специфической информации», то есть *коммуникацией*. Если говорить о дизайнерах визуальных интерфейсов, то они ищут наилучшее представление, доносящее информацию о *поведении* программы, в проектировании которой они принимают участие. Придерживаясь целеориентированного подхода, они должны стремиться представлять поведение и информацию в понятном и полезном виде, который поддерживает маркетинговые цели организации и эмоциональные цели персонажей.

Скажем прямо, что визуальный дизайн пользовательских интерфейсов не исключает эстетических соображений, но такие соображения не должны выходить за рамки функционального каркаса. И хотя в визуальных коммуникациях всегда присутствует субъективизм, мы стремимся минимизировать влияние *вкуса*. Мы пришли к выводу, что четкое выражение эмоциональных целей пользователя и бизнес-целей неоценимы, даже когда речь идет о дизайне тех аспектов визуального интерфейса, которые работают на благо бренда, отвечают за опыт пользователей и обеспечивают физиологические реакции. Более подробно о физиологическом уровне когнитивной обработки мы писали в главе 5.

Графический дизайн и пользовательские интерфейсы

Графический дизайн – это дисциплина, над которой долгие годы (примерно до второй половины 80-х годов прошлого века) довлела полиграфия, поскольку дизайн в основном сводился к созданию упаковок, рекламе, форматированию документов и обустройству среды существования. Графические дизайнеры старой школы неуютно чувствуют себя, сталкиваясь с цифровыми носителями информации, поскольку не привыкли создавать графику на уровне пикселов. Молодое же поколение графических дизайнеров обучено обращению с «новым» форматом и вполне успешно применяет концепции традиционного графического дизайна к цифровой графике.

Графические дизайнеры обычно очень хорошо разбираются в визуальных аспектах и хуже представляют себе понятия, лежащие в основе поведения программного продукта и взаимодействия с ним. Талантливые графические дизайнеры, подкованные и в цифровых аспектах, преуспевают в создании информационно насыщенных, эстетически приятных, впечатляющих интерфейсов, которые мы видим в Windows XP и Mac OS X, а также визуально насыщенных интерфейсов для компьютерных игр и приложений, ориентированных на рядового потребителя. Они способны создавать красивую и адекватную *внешность* интерфейсов, а кроме того – привносить фирменный стиль во внешний вид и поведение программного продукта. Для таких специалистов дизайн или проектирование интерфейса есть в первую очередь тон, стиль, композиция, которые являются атрибутами бренда, во вторую очередь – прозрачность и понятность информации и лишь затем (если до этого вообще доходит дело) – передача информации о поведении посредством ожидаемого назначения (см. главу 13).

Дизайнерам визуальной части интерфейса необходимы некоторые навыки, которые присущи графическим дизайнерам, работающим в цифровом формате, но они должны помимо этого обладать также глубоким пониманием и правильным восприятием роли поведения. Их усилия в значительной степени сосредоточены на организационных аспектах проектирования и на том, как донести до пользователя особенности поведения продукта, используя визуальные якоря и ожидаемые назначения. В центре их внимания находится соответствие между визуальной структурой интерфейса с одной стороны и логической структурой пользовательской ментальной модели и поведения программы – с другой. Кроме того, их заботит вопрос о том, как сообщать пользователю о состояниях программы (скажем, как сделать состояние «доступно для изменения» отличимым от состояния «только для чтения») и что делать с когнитивными аспектами пользовательского восприятия функций (композиция элементов, визуальная иерархия, соотношение фигуры и фона и т. п.).

Визуальный информационный дизайн

Информационные дизайнеры работают не над интерактивными функциями, а над визуализацией данных, содержимого и средств навигации. В визуальном дизайне интерфейсов их навыки чрезвычайно важны, в особенности когда речь идет о приложениях, интенсивно работающих с данными, и некоторых веб-сайтах, где содержание перевешивает функциональность. Усилия информационного дизайнера направлены на то, чтобы представить данные в форме, способствующей их верному истолкованию. Результат здесь достигается через управление визуальной иерархией при помощи таких средств, как цвет, форма, расположение и масштаб.

Распространенными объектами информационного дизайна являются всевозможные графики, диаграммы и прочие способы отображения количественной информации. Эдвард Тафти написал несколько новаторских книг, подробно раскрывающих эту тему, включая «The Visual Display of Quantitative Information» (Tufte, 1983).

Промышленный дизайн

Любое обсуждение вопросов промдизайна выходит за рамки этой книги, однако широкое распространение интерактивных приборов и портативных устройств привело к тому, что роль промышленного дизайна в создании интерактивных продуктов растет прямо на глазах. Как и в случае с разницей в профессиональных навыках у графических дизайнеров, дизайнеров интерфейсов и информационных дизайнеров, существует разделение промдизайнеров на две категории: одни специализируются на создании красивых и полезных форм, в то время как таланты других относятся к сфере логического и эргономического представления физических элементов управления способом, который соответствует поведению пользователя и отражает поведение устройства. По мере того как все большее число устройств обзаводятся полнофункциональными дисплеями, возрастает важность сотрудничества проектировщиков взаимодействия, дизайнеров интерфейсов и промдизайнеров в вопросах создания полноценных и эффективных решений.

Строительные блоки визуального дизайна интерфейсов

По существу дизайн интерфейсов сводится к вопросу о том, как оформить и расположить визуальные элементы таким образом, чтобы внятно отразить поведение и представить информацию. Каждый элемент визуальной композиции имеет ряд свойств, таких как форма и цвет, и сочетание этих свойств придает элементу смысл. Каждое отдельное свойство само по себе редко обладает естественным смыслом. Скорее можно сказать, что пользователь получает возможность разобраться

в интерфейсе благодаря различным способам приложения этих свойств к каждому из элементов интерфейса. В тех случаях, когда два объекта обладают общими свойствами, пользователь предположит, что эти объекты связаны или похожи. Когда пользователи видят, что свойства отличаются, они предполагают, что объекты не связаны. Наиболее контрастные объекты сильнее привлекают наше внимание.

Задолго до того, как ребенок начинает понимать речь и говорить, он проявляет способность различать объекты, контрастирующие визуально. Детская передача «Улица Сезам» полагается на эту человеческую способность, предлагая детям выбирать объект, не похожий на другие или не входящий в группу. Визуальный дизайн интерфейсов создает смыслы схожим образом, что на практике дает гораздо лучший результат, чем просто слова.

Создавая пользовательский интерфейс, проанализируйте перечисленные ниже визуальные свойства каждого элемента или группы элементов. Чтобы создать полезный и привлекательный пользовательский интерфейс, следует тщательно поработать с каждым из этих свойств.

Форма

Какую форму имеет объект? Он круглый, квадратный или похож на амебу? Форма – главный признак сущности объекта для человека. Мы узнаем объекты по контурам; силуэт ананаса, текстурированного синим мехом, все равно позволяет нам понять, что это ананас. При этом различение форм требует большей концентрация внимания, чем анализ цвета или размера. Поэтому форма – не лучшее свойство для создания контраста, если требуется привлечь внимание пользователя. Слабость формы как фактора в распознавании объектов становится очевидна, если взглянуть на Dock[1] операционной системы Mac OS X – здесь можно по ошибке вызвать iTunes вместо iDVD или iWeb вместо iPhoto. Пиктограммы имеют различную форму, но обладают сходными размерами, цветами и текстурой.

Размер

Насколько велик или мал объект относительно других объектов на экране? Более крупные элементы привлекают больше внимания, особенно если они значительно превосходят размерами окружающие элементы. Размер является переменной *упорядоченной* и *поддающейся количественному определению*, то есть люди автоматически упорядочивают объекты по размеру и склонны оценивать их по размеру; если у нас есть текст в четырех размерах, предполагается, что относительная

[1] Специальный интерфейсный элемент операционной системы Mac OS X, который позволяет запускать программы и переключаться между ними. – *Примеч. науч. ред.*

важность текста растет вместе с размером и что полужирный текст более важен, чем текст с нормальным начертанием.

Таким образом, размер – полезное свойство для обозначения информационных иерархий. Достаточное расхождение в размерах обычно быстро привлекает внимание человека. Жак Бертен (Jacques Bertin) в своей классической работе «The Semiology of Graphics» (Bertin, 1983) описывает размер как *диссоциативное* свойство. Это означает, что если объект очень мал или очень велик, становится сложно интерпретировать другие переменные, например форму.

Яркость

Насколько темным или светлым является объект? Разумеется, понятия темного и светлого обретают смысл преимущественно в контексте яркости фона. На темном фоне темный текст почти не виден, тогда как на светлом он будет резко выделяться. Как и в случае с размером, значение яркости может быть диссоциативным; скажем, если фотография слишком темная или слишком светлая, становится невозможно разобрать, что на ней. Контрастность люди воспринимают легко и быстро, так что значение яркости может стать хорошим инструментом привлечения внимания к тем элементам, которые требуется подчеркнуть. Значение яркости – также *упорядоченная* переменная, например, более темные (с более низкой яркостью) цвета на карте легко интерпретируются: они обозначают бóльшие глубины или бóльшую плотность населения.

Цвет

Желтый, красный или оранжевый? Цветовые различия быстро привлекают внимание. В некоторых профессиональных областях цвета имеют конкретные значения, и этим можно пользоваться. Так, для бухгалтера красный цвет – отрицательные результаты, а черный – положительные; для трейдера, работающего с ценными бумагами, синий – сигнал покупать, а красный – сигнал продавать (по меньшей мере, в США это так). Цвета приобретают смыслы и благодаря социальным контекстам, в которых проходит наше взросление. Для человека с Запада, выросшего среди светофоров, красный означает «стоп», а иногда даже «опасность», тогда как в Китае красный – это цвет удачи. Белый цвет на Западе ассоциируется с чистотой и миром, а в Азии – с похоронами и смертью. При этом в отличие от размера или яркости цвет изначально не обладает свойством упорядоченности и не выражается количественно, поэтому далеко не идеален для передачи информации такого рода. Кроме того, не следует делать цвет единственным способом передачи информации, поскольку цветовая слепота встречается довольно часто.

Применяйте цвет с умом. Чтобы создать эффективную визуальную систему, позволяющую пользователю выявлять сходства и различия

объектов, используйте ограниченный набор цветов – эффект радуги перегружает восприятие пользователя и ограничивает возможности по передаче ему информации. Кроме того, в вопросах, касающихся цвета, могут возникнуть конфликты между нуждами маркетинга и задачей отражения интерфейсных идей. Чтобы отыскать компромисс в такой ситуации, вам может потребоваться талантливый визуальный дизайнер (и по совместительству дипломат).

Направление

Куда указывает объект – вверх, вниз, или вбок? Направление полезно, когда требуется передавать информацию об ориентации (вверх или вниз, вперед или назад). Помните, что восприятие направления может быть затруднено в случае некоторых форм и при малых размерах объектов, поэтому ее лучше использовать в качестве вторичного признака. Так, если требуется показать, что рынок акций пошел вниз, можно использовать направленную вниз стрелку красного цвета.

Текстура

Грубая или гладкая, однообразная или неровная? Разумеется, изображенные на экране элементы не обладают настоящей текстурой, но способны создавать ее видимость. Текстура редко бывает полезна для передачи различий или привлечения внимания, поскольку требует значительной концентрации на деталях. Кроме того, для передачи текстуры требуются значительные затраты пикселов. И тем не менее текстура может быть важной подсказкой: когда мы видим область, текстурированную резиной, то предполагаем, что следует ухватить устройство за эту область. Засечки и выпуклости на элементах пользовательского интерфейса обычно указывают, что элемент можно перетаскивать, а фаски или тени у кнопки усиливают ощущение, что ее можно нажать.

Расположение

Как располагается элемент относительно других элементов? Подобно размеру расположение – это переменная *упорядоченная* и *выражаемая количественно*, а значит, полезная для передачи иерархии. Расположив наиболее важные или наиболее востребованные элементы слева вверху, мы воспользуемся порядком восприятия элементов на экране на благо продукта. Расположение также может служить средством создания пространственных отношений между объектами на экране и объектами реального мира.

Принципы визуального дизайна интерфейса

Человеческий мозг – великолепное устройство распознавания образов. Оно извлекает смысл из плотных потоков зрительной информа-

ции, обрушивающихся на нас буквально отовсюду. Наш мозг справляется с этим шквалом входных данных, выявляя визуальные закономерности и создавая для наблюдаемых нами объектов систему приоритетов. В конечном итоге это позволяет нам осознанно воспринимать видимый мир. Именно способность зрительной системы человеческого мозга к сборке частей визуального поля в образы на основании визуальных якорей (подсказок) позволяет нам обрабатывать зрительную информацию столь быстро и эффективно. Представьте, что вам вдруг пришлось бы вручную вычислять траекторию полета бейсбольного мяча, чтобы предсказать, где он упадет. Наши глаза и мозг вместе делают это за доли секунды, не требуя от нас сознательных усилий. Процесс создания визуального дизайна интерфейса должен опираться на наши природные способности к обработке визуальной информации, чтобы обеспечить передачу пользователям информации и отражение возможностей и функций программы.

Одной главы совершенно недостаточно для полного раскрытия темы визуального дизайна интерфейсов. Однако существует ряд важных принципов, которые помогут вам создавать интерфейсы, максимально простые в восприятии и приятные для глаза. Как уже говорилось, Кевин Маллет и Даррел Сано дают превосходный и подробный анализ этих базовых принципов; мы лишь кратко обсудим некоторые из наиболее важных принципов визуального дизайна интерфейса.

При создании графических интерфейсов следует:

- использовать визуальные свойства для группировки элементов и создания четкой иерархии;
- создавать визуальную структуру и прокладывать логический маршрут на каждом уровне организации;
- использовать целостные, непротиворечивые и соответствующие контексту образы;
- интегрировать визуальный стиль с функциональностью осмысленно и последовательно;
- избегать визуального «шума» и беспорядка.

Эти принципы и некоторые другие общие закономерности, касающиеся работы с текстом и цветом в графических пользовательских интерфейсах, подробно обсуждаются в следующих разделах.

Используйте визуальные свойства для группировки элементов и создания четкой иерархии

Как правило, имеет смысл группировать логические наборы функциональных или информационных элементов посредством визуальных свойств, например цвета или пространственных характеристик. Последовательно применяя эти визуальные свойства в интерфейсе, вы можете создавать шаблонные образы, которые ваши пользователи быстро научатся распознавать. Например, в Windows XP все кнопки вы-

пуклые, со скругленными углами, а текстовые поля прямоугольные, слегка вдавленные, с белым фоном и синей окантовкой. Благодаря систематическому применению этого образа невозможно перепутать кнопку и поле ввода, несмотря на некоторые сходства.

ПРИНЦИП проектирования

Основой визуального интерфейса являются визуальные шаблоны.

Глядя на любой набор визуальных элементов, пользователь бессознательно задается вопросом: «Что здесь представляет интерес?» – и почти сразу же: «Какая связь между этими объектами?» Мы должны стремиться к тому, чтобы интерфейс содержал в себе ответ на оба вопроса.

Создание иерархии

Исходя из сценариев определите, какие функциональные и информационные элементы должны восприниматься пользователями сходу, какие являются вторичными, а какие нужны лишь в исключительных ситуациях. Такое ранжирование и служит основой для визуальной иерархии.

Используйте цвет, насыщенность, контрастность, размер и положение, чтобы создать видимые различия между уровнями иерархии. Самые важные элементы должны быть более крупными, более ярких цветов, более насыщенными и более контрастными. Их следует располагать над прочими элементами или делать выступающими. Выделяемые элементы лучше всего красить в насыщенные цвета. Менее важные элементы должны быть менее насыщенными, менее контрастными, более мелкими и плоскими. Нейтральные светлые цвета уводят их на второй план.

Разумеется, настройку этих свойств следует выполнять осторожно. Не следует делать самый важный элемент огромным, красным и выпуклым. Часто бывает достаточно изменить лишь одно из свойств. Если обнаружится, что два элемента различной важности состязаются за внимание пользователя, «прикрутить фитиль» менее важного будет лучшим решением, чем пытаться «разжечь» более важный. Так у вас останется больше пространства для создания акцента на самых важных элементах. (Вот хорошая аналогия: если все слова на странице набраны жирным красным шрифтом, выделяется ли хоть одно из слов?)

Создание четкой визуальной иерархии – одна из сложнейших задач в визуальном дизайне интерфейсов, ее решение требует навыков и таланта. Качественную визуальную иерархию пользователи практически не замечают – а вот ее отсутствие и проистекающая из этого путаница сразу бросаются в глаза.

Визуализация связей

Чтобы передать связь элементов, вновь обратитесь к сценариям. Необходимо определить не только элементы со сходными функциями, но и элементы, наиболее часто используемые совместно. Совместно используемые элементы обычно следует сгруппировать в **пространстве**, чтобы минимизировать перемещения мыши, тогда как элементы, которые могут не использоваться вместе, но обладают сходными функциями, можно группировать **визуально**, даже если они не группируются в **пространстве**.

Пространственная группировка объясняет пользователям, каким образом одни задачи, данные и инструменты связаны с другими, и может намекать на правильную последовательность действий. Хорошая группировка посредством *расположения* принимает во внимание порядок задач и подзадач и движение взгляда по экрану: слева направо для западных языков и, как правило, сверху вниз. (Более подробно этот момент мы обсудим чуть позже.)

Элементы, расположенные рядом, как правило, связаны друг с другом. Во многих интерфейсах такая группировка реализована слишком тяжеловесно: куда не взглянешь – рамки, причем иногда рамка заключает в себе всего один или два элемента. Часто того же эффекта более грамотно можно достичь посредством *расстояний*. К примеру, на панели инструментов кнопки могут отделяться друг от друга четырьмя пикселами. Чтобы вычленить файловые команды («открыть», «новый файл», «сохранить») в отдельную группу, достаточно увеличить расстояние между кнопками файловых команд и соседней группой кнопок до восьми пикселов.

Элементы, разделенные большими расстояниями, можно группировать посредством общих визуальных свойств, создавая шаблон, который в конечном итоге приобретет самостоятельный смысл для пользователей. Так, использование объема для создания ощущения физического ожидаемого назначения – вероятно, самый эффективный способ отделять элементы управления от данных и фоновых элементов (более подробно об ожидаемых назначениях читайте в главе 13). Эта стратегия часто применяется в рисовании пиктограмм. В операционной системе Mac OS X применяются яркие цвета для пиктограмм приложений и тусклые – для редко используемых вспомогательных программ. Зеленая кнопка запуска устройства может перекликаться с похожей анимированной зеленой пиктограммой, указывающей, что устройство функционирует нормально.

Определившись с группами и визуальными особенностями этих групп, начинайте подстраивать контраст между группами – подчеркивая или, наоборот, затеняя группы сообразно их важности в текущем контексте. Подчеркивайте различия между группами, но минимизируйте различия между элементами одной группы.

Тест с прищуриванием

Есть хороший способ убедиться, что визуальный дизайн эффективно задействует иерархию и отношения, – дизайнеры называют этот прием **тестом с прищуриванием** (squint test). Закройте один глаз и посмотрите на экран прищуренным вторым глазом. Обратите внимание на то, какие элементы слишком выпирают, какие стали нечеткими, а какие объединились в группы. Эта процедура часто вскрывает не замеченные ранее проблемы в композиции интерфейса.

Создавайте визуальную структуру и прокладывайте логический маршрут на каждом уровне организации

Интерфейсы удобно представлять себе состоящими из визуальных и интерактивных элементов, объединяемых в группы с помощью панелей, которые, в свою очередь, можно группировать в экраны, представления или страницы. Такая группировка, как было сказано выше, может проводиться посредством распределения в пространстве или при помощи общих визуальных свойств. В монопольном приложении может быть несколько уровней таких структур, так что крайне важно сохранять прозрачную визуальную структуру, чтобы пользователь мог легко переходить от одной части интерфейса к другой в соответствии со своим рабочим процессом.

В оставшейся части этого раздела мы опишем ряд важных свойств, помогающих задать четкую визуальную структуру.

Выравнивание и сетка

Выравнивание визуальных элементов – один из главных приемов, позволяющих дизайнеру представить продукт пользователям в систематизированном и упорядоченном виде. Сгруппированные элементы следует выравнивать как по горизонтали, так и по вертикали (рис. 14.1). В общем случае каждый элемент на экране следует выровнять по максимально возможному числу других элементов. Отказ от выравнивания двух элементов или двух групп элементов должен быть осознанным: это допустимо только для достижения конкретного разделяющего эффекта. В числе прочего дизайнерам следует обращать внимание на:

- **Выравнивание подписей**. Подписи для элементов управления, расположенные друг над другом, должны быть выровнены по общей границе. Если все подписи имеют примерно одинаковую длину, выравнивайте их по левой стороне – так пользователям будет легче их читать, нежели при выравнивании вправо.
- **Выравнивание внутри группы функциональных элементов**. Группа связанных флажков, вариантов выбора или текстовых полей должна подчиняться выравниванию стандартной сетки.

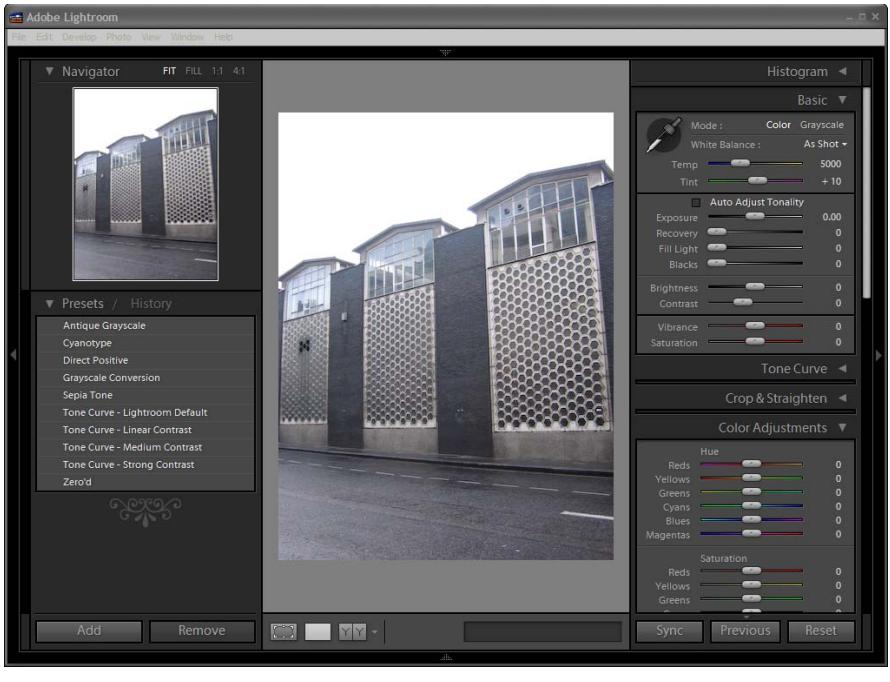


Рис. 14.1. *Adobe Lightroom весьма эффективно использует выравнивание по композиционной сетке. Текст, функциональные элементы и группы элементов управления очень четко выравниваются по сетке с фиксированным шагом. Следует отметить, что отбивка элементов управления и подписей элементов группы вправо может мешать быстрому их прочтению*

- **Выравнивание элементов, разнесенных по группам и панелям.** Группы элементов управления и прочие объекты на экране везде, где это возможно, должны быть привязаны всё к той же сетке.

Сетка – один из самых мощных инструментов визуального дизайнера, стремительно набравший популярность в годы после Второй мировой войны благодаря швейцарским печатникам. Сетка обеспечивает однородность и последовательность структуры композиции, что особенно важно при проектировании интерфейса с несколькими уровнями визуальной или функциональной сложности. После того как проектировщики взаимодействия определили общую инфраструктуру приложения и элементов его пользовательского интерфейса (см. главу 7), дизайнеры интерфейса должны организовать композицию в структуру в виде сетки, которая будет должным образом подчеркивать важные элементы и структуры и оставлять жизненное пространство для менее важных элементов и элементов более низкого уровня.

Как правило, сетка делит экран на несколько крупных горизонтальных и вертикальных областей (рис. 14.2). Качественно спроектиро-

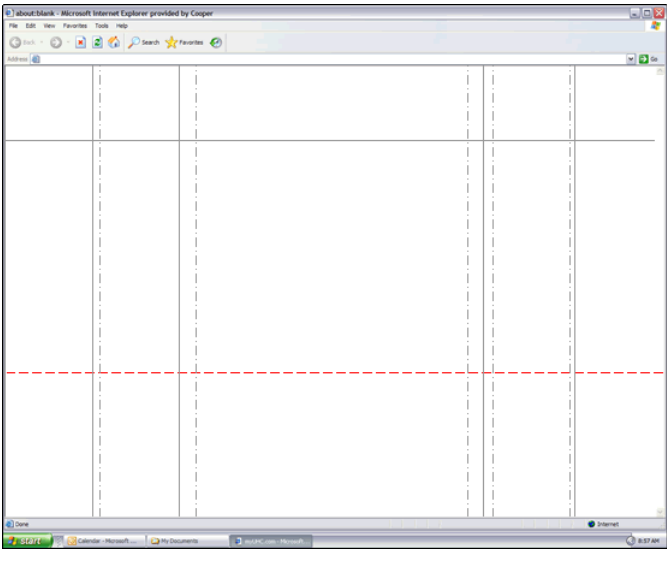


Рис. 14.2. *В этом примере композиционная сетка регламентирует размер и положение различных областей экрана веб-сайта. Она обеспечивает согласованность различных экранов, сокращая как работу дизайнера по созданию композиции, так и усилия, которые должен приложить пользователь, чтобы прочитать и понять содержимое экрана*

ванная сетка задействует понятие *шага*, то есть минимального расстояния между элементами. К примеру, если шаг сетки составляет четыре пиксела, все расстояния между элементами и группами должны быть кратны четырем.

В идеальном случае сетка должна задавать и пропорции различных областей экрана. Такие отношения обычно выражаются дробями. Среди распространенных дробей – прославленное «золотое сечение» (обозначаемое греческой буквой «фи» и равное примерно 1,62), которое часто встречается в природе и считается особенно приятным для человеческого глаза; величина, обратная квадратному корню из двух (примерно 1:1,41), которая является основой международного стандарта размера бумаги (например, листа А4); и отношение 4:3 – пропорция сторон большинства компьютерных дисплеев.

Разумеется, следует стремиться к балансу между идеализированными геометрическими отношениями и конкретными пространственными потребностями функций и информации, представленных на экране. Идеальная реализация золотого сечения никак не улучшит читаемость экрана, на котором объекты свалены в кучу.

Хорошая композиционная сетка *модульна*, то есть является достаточно гибкой, чтобы учесть все необходимые вариации, при этом сохранив согласованность структуры везде, где это возможно. Как и во мно-

гих других вопросах дизайна, здесь важны простота и последовательность. Если две области на экране требуют примерно одинакового пространства, назначьте им в точности одинаковые размеры. Если две области требуют различного пространства, сделайте их значительно различающимися по размерам. Если шаг сетки слишком мал, сетка станет трудно воспринимаемой из-за своей сложности. Мелкие отличия могут вызвать у пользователя ощущение шаткости (хотя он вряд ли осознает причину этого ощущения) и в конечном итоге разрушить огромный потенциал применения сетки.

Решения по поводу композиции должны быть в определенном смысле бескомпромиссными: «почти квадрат» ни на что не годится; «почти один к двум» – тоже. Если композиция на экране близка к простой дробной его части – половине, трети или одной пятой, – скорректируйте композицию таким образом, чтобы она занимала ровно половину, треть или одну пятую часть экрана. Используйте точные, четкие, ярко выраженные пропорции.

Использование сетки в визуальном дизайне интерфейсов дает ряд преимуществ:

- **Удобство применения.** Поскольку сетка делает расположение элементов единообразным, пользователи быстро приобретают навыки поиска нужных элементов в интерфейсе. Если заголовок экрана всегда находится на одном месте, пользователю не приходится задумываться или разглядывать экран в поисках этого заголовка. Последовательность в расположении элементов и выборе расстояний между ними облегчает работу механизмов визуальной обработки в мозгу человека. Качественно спроектированная сетка в существенной степени упрощает восприятие экрана.
- **Эстетическая привлекательность.** Аккуратно применяя сетку и выбирая подходящие соотношения между различными областями экрана, дизайнер может создать ощущение порядка, который удобен пользователям и стимулирует их вступать во взаимодействие с продуктом.
- **Эффективность.** Стандартизация композиции снижает трудозатраты, связанные с созданием высококачественных визуальных интерфейсов. Мы считаем, что создание сетки и включение ее в процесс на ранних этапах детализации проектных решений сокращает число итераций и действий по «доводке» интерфейса. Качественная и явно обозначенная сетка закладывает основу для легко модифицируемого и расширяемого дизайна, позволяя разработчикам находить хорошие композиционные решения, когда необходимо внести изменения.

Создание логического маршрута

Композиция должна не только в точности следовать сетке, но и структурировать эффективный **логический маршрут** через интерфейс для

пользователей, принимая во внимание тот факт, что (в случае западных языков) взгляд движется сверху вниз и слева направо (рис. 14.3).

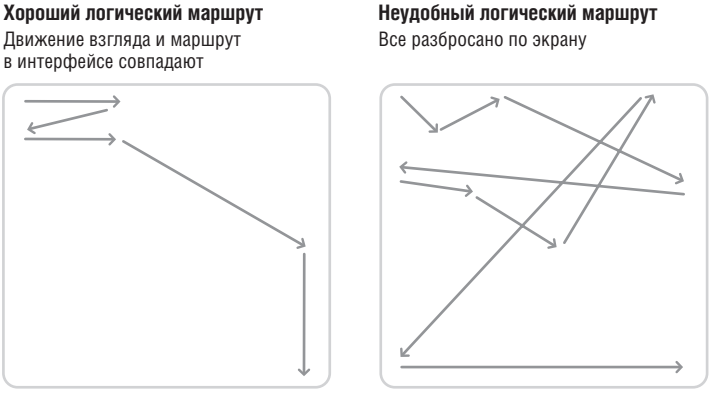


Рис. 14.3. *Движение взгляда по интерфейсу должно прокладывать логический маршрут, позволяющий пользователям эффективно и успешно решать задачи и достигать целей*

Симметрия и баланс

Симметрия – полезное средство организации интерфейса с точки зрения достижения визуального равновесия. Несимметричные интерфейсы обычно выглядят так, словно вот-вот завалятся на один бок. Опытные дизайнеры способны достигать асимметричного равновесия, управляя визуальным весом отдельных элементов, подобно тому, как можно достичь равновесия качелей-доски, если посадить на противоположные концы детей различного веса. В контексте пользовательского интерфейса трудно получить асимметричный дизайн из-за высокой стоимости экранного пространства. Тест с прищуриванием позволяет проверить сбалансированность интерфейса.

В интерфейсах чаще всего применяют два типа симметрии: вертикальная осевая симметрия (симметрия относительно вертикальной линии, проведенной через центр группы элементов) и диагональная осевая симметрия (симметрия относительно диагонали). В большинстве диалоговых окон присутствует симметрия одного из этих типов, чаще всего диагональная (рис. 14.4).

Монопольные приложения, как правило, не обладают симметрией на верхнем уровне (они достигают равновесия посредством качественной сетки), но элементы хорошо спроектированного интерфейса у таких приложений почти наверняка в той или иной степени симметричны (рис. 14.5).

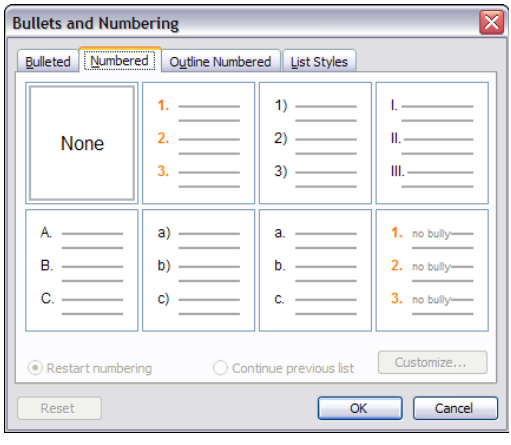


Рис. 14.4. *Диагональная симметрия внутри диалогового окна Список (Bullets and Numbering) приложения Microsoft Word. Ось симметрии проходит из нижнего левого угла в верхний правый*

Рис. 14.5. *Вертикальная симметрия в палитре инструментов Macromedia Fireworks*

Используйте целостные, непротиворечивые и соответствующие контексту образы

Применение пиктограмм и других наглядных элементов способно помочь пользователю разобраться в интерфейсе или, наоборот, вызвать раздражение, запутать или даже оскорбить, если пиктограммы выбраны неудачно. Очень важно, чтобы дизайнеры понимали, какое послание должна передать пользователю программа и какое послание он рассчитывает получить. Хорошее понимание персонажей и их ментальных моделей, как правило, дает прочный фундамент для текстового и визуального языков интерфейса. Играют роль и культурные аспекты. Дизайнеры должны отдавать себе отчет в том, что цвет в различных культурах имеет различные значения (в Китае красный цвет не считается предупреждающим), равно как и жесты (большой палец, поднятый вверх, считается крайне оскорбительным жестом в Турции), и символы (восьмиугольник предписывает остановку в США, но мало в каких еще странах). Кроме того, следует знать цветовые коды, принятые в разных областях человеческой деятельности. Например, в больницах желтый цвет обозначает радиацию, а красный, как правило, используется в ситуациях, угрожающих жизни. На терминале фондового рынка красный – сигнал продавать. Прежде чем вы возьметесь за дело, убедитесь, что понимаете визуальный язык, принятый в сфере деятельности и культурном окружении ваших пользователей.

Визуальные элементы должны быть частью глобального визуального языка, связующего компоненты интерфейса. Это означает, что логически сходные элементы должны иметь общие визуальные свойства, такие как положение, размер, толщина линий и общий стиль, различаясь лишь в том, что подчеркивает их значение. Идея состоит в создании связной системы элементов. Интерфейс, в котором эта цель достигнута, кажется безупречным: ни один элемент не выглядит так, словно его добавили в последнюю минуту.

Помимо функциональной ценности пиктограммы способны играть значительную роль в передаче атрибутов бренда. Яркие «мультяшные» пиктограммы могут попасть в точку, если вы проектируете веб-сайт для детей, а выверенные сдержанные пиктограммы больше подойдут для бизнес-приложения. Независимо от того, какой стиль выбран, соблюдайте преемственность – если на одних пиктограммах линии – жирные и черные со скругленными углами, а на других – тонкие ломаные, визуальный стиль «развалится».

Дизайн и рисование пиктограмм – совершенно самостоятельная область. Создание понятных изображений при низком разрешении требует значительного времени и практики – лучше делегировать эту задачу опытным дизайнерам. Пиктограммы – сложная тема в плане восприятия, так что мы здесь подчеркнем лишь несколько важных ключевых моментов. Тех читателей, кто хочет больше узнать о пригодных

к использованию пиктограммах, мы настойчиво отсылаем к книге Уильяма Хортона (William Horton) «The Icon Book». Примеры в этой книге, возможно, покажутся вам устаревшими, но основные принципы по-прежнему актуальны.

Функциональные пиктограммы

Разработка пиктограмм, представляющих функции или операции, выполняемые над объектами, – занятие нелегкое, но увлекательное. Основная трудность заключается в представлении абстрактных понятий на пиктографическом, визуальном языке. В таких случаях вместо образов, в которых никто не разберется, лучше опираться на идиомы и снабжать их всплывающими подсказками (см. главу 23) или подписями.

В случае очевидных и конкретных функций придерживайтесь следующих правил:

- Помещайте на пиктограмму как **функцию**, так и **объект**, чтобы облегчить пользователю понимание пиктограммы. Глагол в сочетании с существительным понять легче, чем глагол сам по себе. Например, если команду Вырезать представить в виде перечеркнутого крестом документа, это будет понятнее, чем метафорическое изображение ножниц.
- Остерегайтесь метафор и представлений, которые могут не обладать требуемыми значениями в глазах целевой аудитории. К примеру, поднятый вверх большой палец в западной культуре означает «о'кей» и может показаться подходящей пиктограммой для обозначения положительной реакции, однако на Среднем Востоке и в других культурах этот жест является оскорбительным – в любом приложении, предназначенном для глобального рынка, его следует избегать.
- Визуально группируйте взаимосвязанные функции – либо пространственно, либо, если это невозможно, с помощью цвета и иных изобразительных средств.
- Создавайте простые пиктограммы; избегайте лишних деталей.
- Используйте элементы повторно везде, где это возможно, чтобы пользователь изучил их однажды и навсегда.

Символы как отражение объектов

Создание уникальных символов для различных типов объектов в интерфейсе также помогает пользователю лучше понимать интерфейс. Такие символы не всегда могут быть образными или метафорическими – а значит, они часто идиоматичны (сила идиом подробно обсуждается в главе 13). Такие визуальные маркеры позволяют пользователю ориентироваться среди объектов быстрее, чем просто подписи. Чтобы установить связь между символом и объектом, используйте символ всякий раз, когда объект появляется на экране.

ПРИНЦИП проектирования

Элементы, различающиеся поведением, должны различаться и визуально.

Дизайнер интерфейсов должен позаботиться также о том, чтобы символы, представляющие объекты различных типов, зрительно различались. Вычленить конкретную пиктограмму на экране, переполненном почти одинаковыми пиктограммами, столь же трудно, как и найти конкретное слово на экране, заполненном текстом. Очень важно зрительно дифференцировать (противопоставлять) объекты с разным поведением, в том числе различные варианты элементов управления, таких как кнопки, ползунки и флажки.

Визуализация пиктограмм и символов

По мере того как графические возможности цветных дисплеев расширяются, растет и искушение представлять пиктограммы и символы все более детально, достигая почти фотореализма. Однако эта тенденция, в конечном счете, не отвечает целям пользователя, особенно в бизнес-приложениях. Пиктограммы должны оставаться простыми и схематичными, с минимумом цветов и теней, и сохранять свои скромные размеры. В Windows Vista и Mac OS X были предприняты шаги в направлении более подробного представления пиктограмм. Хотя такие пиктограммы выглядят впечатляюще, они незаслуженно привлекают к себе внимание, а при малом размере отображаются плохо – то есть либо занимают избыточно много дорогостоящего места на экране, либо неразборчивы. Кроме того, они вынуждают разработчика пренебрегать визуальной связностью элементов в интерфейсе, поскольку очень немногие функции (в основном те, что относятся к аппаратной части) могут быть адекватно представлены конкретными фотореалистичными изображениями. Фотографические пиктограммы подобны тексту, набранному только заглавными буквами: различия между ними нечеткие, и в них легко запутаться. Пользователям легче всего различать пиктограммы по форме, однако в случае Mac OS X контуры всех пиктограмм одинаково размытые. Интерфейс Mac OS X переполнен фотореализмом, отвлекающим пользователей и не работающим на их благо (рис. 14.6).

Визуализация поведения

Вместо того чтобы описывать результаты работы функций в интерфейсе исключительно словами (или, что еще хуже, не давать никаких описаний), *показывайте* пользователю, какими будут эти результаты. Для этой цели применяйте визуальные элементы. Не следует путать этот прием с использованием пиктограмм на элементах управления, представляющих ожидаемые назначения. В дополнение к тексту, описывающему параметр или состояние, поместите картинку или диаграмму, описывающую *поведение*. Хотя визуализация, как правило,

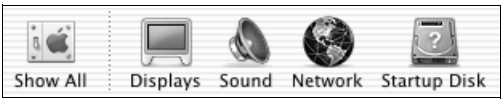


Рис. 14.6. *Фотореалистичные пиктограммы в Mac OS X. Подобный уровень детализации лишь отвлекает внимание от функциональных и информационных элементов. Кроме того, если в некоторых случаях детализация знакомых пользователю объектов оправданна, какой смысл в детализированных изображениях незнакомых предметов или абстрактных понятий (таких как компьютерная сеть)? Сколько пользователей видели «голый» жесткий диск (крайняя справа пиктограмма)? В конечном счете пользователю приходится ориентироваться по подписям, чтобы разобраться в пиктограммах, которые нужны ему не очень часто*

требует дополнительного места на экране, ее способность ясно передавать смысл стоит потраченных на это пикселов. Компания Microsoft совершила это открытие несколько лет назад, и с тех пор диалоговые окна (в частности, в Microsoft Word) изобилуют визуальными элементами, дополняющими текстовые описания элементов управления. Photoshop и другие графические приложения уже давно показывают пользователю результаты операций в виде миниатюр.

Доносите до пользователя функцию и поведение визуально.

Диалоговое окно Предварительный просмотр в Microsoft Word (рис. 14.7) показывает, как будет выглядеть напечатанный документ с учетом установленного размера бумаги и полей. Многие пользователи плохо представляют, как выглядит левое поле размером 1,2 дюйма, а Предварительный просмотр наглядно показывает им это. Компания Microsoft могла пойти еще дальше и организовать непосредственный ввод в этом диалоговом окне. Тогда пользователи могли бы перетаскивать границу левого поля и наблюдать, как меняется числовое значение в соответствующем счетчике. Связанное с таким элементом управления числовое поле ввода не утрачивает своей важности, его нельзя полностью заменить визуальным элементом. Поле ввода показывает точные значения параметров, а визуальный элемент демонстрирует окончательный внешний вид страницы.

Интегрируйте визуальный стиль с функциональностью осмысленно и последовательно

Если дизайнеры интерфейсов решили придерживаться в интерфейсе определенного стиля, это следует делать глобально. Каждый аспект интерфейса должен быть проанализирован с точки зрения соответствия стилю; нельзя ограничиваться лишь некоторыми визуальными

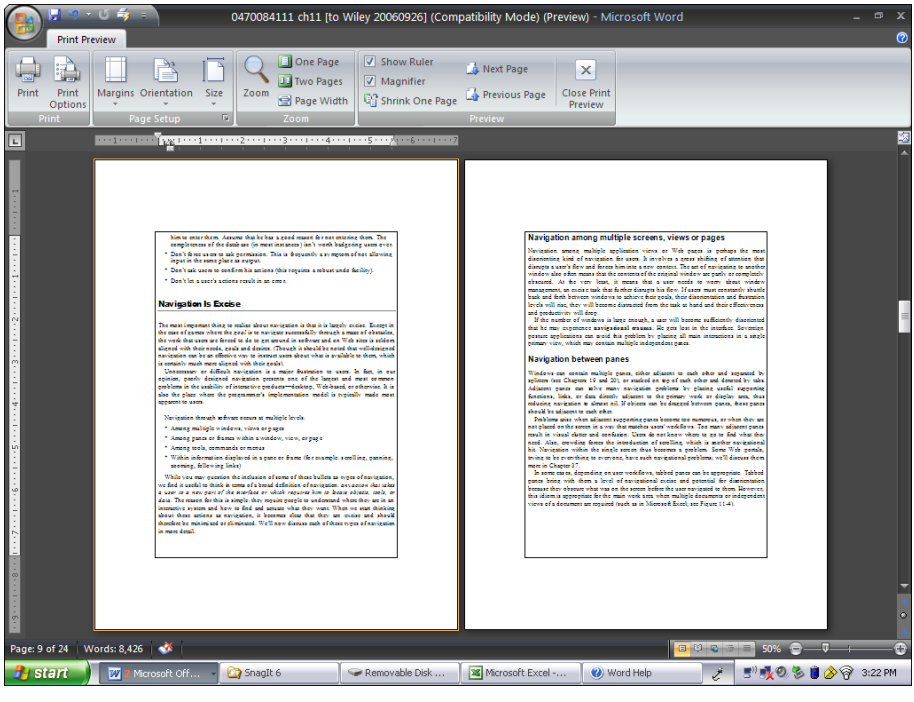


Рис. 14.7. *Функция предварительного просмотра в Microsoft Word наглядно демонстрирует визуальное выражение функций приложения. Эта функция не требует, чтобы пользователь пытался представить, как будет выглядеть поле в 1,2 дюйма, но при этом позволяет легко понять, к каким результатам приводит то или иное значение*

элементами. Вы же не хотите, чтобы интерфейс выглядел так, словно кто-то в спешке замазал его слоем краски? Необходимо убедиться, что функциональные аспекты графического интерфейса программы находятся в полной гармонии с общим визуальным стилем бренда вашей продукции. Поведение программы – часть бренда, и опыт взаимодействия пользователя с продуктом должен отражать баланс формы, содержания и поведения.

Форма и функция

Для многих заинтересованных лиц (владельцев проекта) визуальный стиль – очень притягательная область, однако стилизованные элементы интерфейса следует держать под неусыпным контролем – особенно при проектировании монопольных приложений. Движущими силами в разработке визуального стиля должны быть базовые формы, поведение и ожидаемое назначение (см. главу 13), тогда как соображения эстетического плана не должны мешать передаче смысла в интерфейсе и взаимодействию пользователя с продуктом.

При этом образовательные и развлекательные приложения (особенно адресованные детям) оставляют больше возможностей для экспериментов со стилем. В таких случаях и визуальное впечатление, создаваемое интерфейсом, и содержимое приложения оказывают влияние на то удовольствие, которое пользователь получает от взаимодействия с приложением; это служит весомым аргументом в пользу более тесной связи оформления управляющих элементов с содержимым. Но и здесь следует помнить об ожидаемом назначении, позволяющем пользователям быстро получать доступ к информации.

Бренд, опыт потребителей и пользовательский интерфейс

Большинство успешных компаний вкладывает значительные средства в создание и поддержание своих брендов. Компания, культивирующая собственный бренд, может диктовать цены на свои товары, одновременно морально поощряя лояльность потребителей. Бренд является индикатором высокого качества продукта и предполагает, что пользователь предпочтет его, опираясь на свой вкус.

В самом простом смысле слова бренд есть сумма всех взаимодействий людей с конкретной компанией. Поскольку взаимодействие все чаще происходит по каналам, основанным на современных технологиях, неудивительно, что усилия фирм, направленные на создание «брендовых» интерфейсов, велики как никогда. Если цель состоит в постоянном и позитивном взаимодействии с потребителем, то вербальные, визуальные и бихевиоральные (поведенческие) послания бренда должны быть согласованными и непротиворечивыми. Например, если потребитель пытается узнать цены на DSL-услуги в своем районе и не обнаруживает на веб-сайте телефонной компании полезной информации (даже после значительных усилий по ее поиску), он уходит в полной уверенности, что и сама компания – заведение грубое и неприятное. От этого не спасет никакой в мире дизайн. То же верно и для других каналов: если человек не получает нужных ему ответов, то не имеет значения, что компьютерный голос звучит дружелюбно, система принимает голосовой ввод, а представитель службы поддержки завершает беседу наилучшими пожеланиями.

Хотя компании уже довольно давно овладели способами работы с брендом применительно к традиционным маркетинговым и коммуникационным каналам, многие только-только начинают думать о брендах в терминах пользовательского опыта. Чтобы понять работу с брендами в контексте пользовательского интерфейса, полезно рассмотреть ее с двух точек зрения: первое впечатление и долгосрочные отношения.

Как и в отношениях между людьми, первое впечатление от пользовательского интерфейса исключительно важно. Первые пять минут общения закладывают фундамент для долгосрочных отношений. Чтобы это первое пятиминутное общение оказалось успешным, пользова-

тельский интерфейс должен четко и быстро представить бренд. Как правило, визуальный дизайн играет важнейшую роль в создании первого впечатления – в основном с помощью изображений и цвета. Выбрав подходящую цветовую палитру и стиль изображений для интерфейса, нацеленного на поддержку бренда, вы сильно приблизитесь к использованию бренда для создания положительного первого впечатления.

Первые впечатления могут значительным образом влиять и на субъективную пригодность продукта к использованию. В полезнейшей книге «Universal Principles of Design» (Lidwell, Holden, and Butler, 2003) Уильям Лидвел (William Lidwell), Критина Холден (Kritina Holden) и Джил Батлер (Jill Butler) называют это «эффектом эстетического юзабилити». Согласно сформулированному ими принципу (основанному на результатах исследований), дизайн, более приятный с эстетической точки зрения, люди воспринимают как более простой в применении по сравнению с эстетически менее привлекательным дизайном независимо от действительной функциональности.

После того как у человека сложилось первое впечатление, он начинает оценивать, насколько поведение интерфейса соответствует его облику. Создание бренда и построение долгосрочных отношений с клиентами является выполнением тех обещаний, которые были даны при первом знакомстве. Часто проектирование взаимодействия и управление поведением – лучшие способы выполнить обещания, данные пользователю во время знакомства с брендом.

Избегайте визуального «шума» и беспорядка

Визуальный шум в интерфейсе возникает из-за излишних графических элементов, отвлекающих внимание от элементов, непосредственно связанных с функциональностью и поведением программы. Представьте себе, что вы пытаетесь вести разговор в переполненном и шумном ресторане. Беседовать в такой обстановке бывает просто невозможно. То же самое справедливо в отношении пользовательских интерфейсов. Визуальный шум возникает благодаря необязательным декоративным и излишне «объемным» элементам, злоупотреблению линиями и иными разделителями между элементами управления, неуместному или излишне интенсивному использованием цвета, текстур и контраста.

Беспорядочные интерфейсы пытаются втиснуть лишние функции в ограниченное пространство, в результате чего элементы управления начинают мешать друг другу. Причудливые, запутанные или перегруженные интерфейсы повышают информационную нагрузку на пользователя и уменьшают скорость и точность его попыток разобраться в содержимом экрана. Ричард Сол Верман (Richard Saul Wurman) назвал такую реакцию пользователей «информационной тревогой».

Не усложняйте

Вообще говоря, в интерфейсах следует применять простые геометрические формы, минималистичные контуры и ограниченные наборы не очень ярких или нейтральных цветов, уравновешенные небольшим числом высококонтрастных и ярких цветов, позволяющих подчеркивать важную информацию.

Типографика не должна быть слишком разнообразной: одного или двух шрифтов в нескольких размерах вполне достаточно. Когда несколько элементов дизайна (элементы управления, панели, окна) служат родственным или взаимосвязанным логическим целям, эти элементы должны визуально оформляться таким образом, чтобы работал принцип *наследования*. Наследование дает возможность переносить понимание одного элемента на другие сходные элементы. Если элемент требуется выделить, создайте контраст с прочими элементами через настройку одного или нескольких визуальных свойств, таких как размер, цвет и положение.

Бессмысленные вариации визуальных свойств мешают создавать целостные и удобные интерфейсы. Если расстояние между элементами почти одинаковое, сделайте его идентичным. Если два шрифта имеют примерно равные размеры, сделайте размер одинаковым. За любым визуальным элементом, любым отличием цвета, размера или другого визуального свойства должны стоять определенные причины. Если вы не можете сформулировать причину для отличий, избавьтесь от них.

Хороший графический интерфейс, как любой хороший дизайн, визуально *эффективен*. Он максимально полно использует минимальный набор визуальных и функциональных элементов. Распространенный прием, который практикуют как графические дизайнеры, так и промдизайнеры, – эксперимент с удалением отдельных элементов с целью проверки их вклада в ясность задуманного послания пользователю.

Удаляйте элементы, пока продукт не сломается, а затем верните последний удаленный элемент на место.

Есть известное изречение пилота и поэта Антуана де Сент-Экзюпери: «Совершенство достигается не тогда, когда уже нечего прибавить, но когда уже ничего нельзя отнять». Создавая интерфейсы, постоянно стремитесь к визуальной простоте. Чем больше полезной работы способен выполнить элемент интерфейса (без потери ясности), тем лучше. Как сказал Альберт Эйнштейн, вещи должны быть настолько просты, насколько это возможно, но не проще.

Еще одно понятие, связанное с обсуждаемой темой, – это **усиление**, то есть использование элемента интерфейса для нескольких родственных целей. Так, в Microsoft Windows XP рядом с заголовком окна присутствует пиктограмма (рис. 14.8). Эта пиктограмма визуально передает

информацию о содержании окна (то есть мы можем отличить окно Проводника от окна документа Word) и предоставляет доступ к командам управления окном: Свернуть, Развернуть, Закрыть.

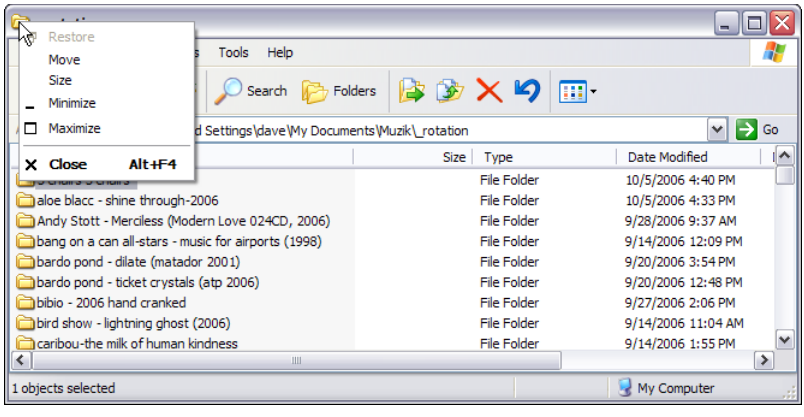


Рис. 14.8. *Пиктограмма в заголовке окна в Windows XP – хороший пример усиления. Она сообщает о содержимом окна и предоставляет доступ к командам управления окном*

Вообще говоря, проектировщики взаимодействия лучше дизайнеров справляются с задачей назначения нескольких функций одному визуальному элементу. Такое сопоставление элементов и функций требует ясного представления о поведении пользователя в конкретном контексте, понимания поведения программы и владения вопросами, связанными с программированием. Но не перестарайтесь: многие дизайнеры интерфейсов слишком увлекаются идеей усиления и в конечном итоге только запутывают пользователей.

Текст в графических интерфейсах

Текст – неотъемлемая составляющая практических всех пользовательских интерфейсов. Письменность позволяет сжато передавать богатую нюансами информацию, однако следует крайне внимательно относиться к применению текста, поскольку он обладает способностью запутывать и усложнять простые вещи.

Человек распознает буквы исходя из их форм. Чем более узнаваема форма, тем проще распознать букву, поэтому СЛОВА, СОСТОЯЩИЕ СПЛОШЬ ИЗ ЗАГЛАВНЫХ БУКВ, ЧИТАТЬ ТРУДНЕЕ, чем написанные обычным образом: в заглавных буквах отсутствуют привычные глазу начертания, и поэтому чтение требует большего внимания. Избегайте в своих интерфейсах слов, состоящих из заглавных букв.

Распознавание отдельных слов – не то же самое, что *чтение*, при котором мы осознанно сканируем отдельные слова и интерпретируем их

значения в контексте. Следует стараться минимизировать объем текста, который пользователю необходимо прочитать, чтобы сориентироваться в интерфейсе. Вот когда пользователь найдет то, что его интересует, у него должна быть возможность прочитать об этом подробнее. Применение коротких простых слов облегчает навигацию при минимальной необходимости в чтении.

Наш мозг способен быстро различать объекты в интерфейсе, если визуальные символы и идиомы указывают на *тип* объекта. После визуальной идентификации типа мы можем прочитать текст, чтобы понять, *какой* конкретно объект мы видим. При такой схеме нам не приходится ничего читать о типах объектов, которые нас не интересуют, и это ускоряет навигацию и избавляет от интерфейсных налогов. Сопроводительный текст выходит на сцену только тогда, когда он важен.

Графически представляйте тип объекта; текстом описывайте сам объект.

Что касается читаемого текста в интерфейсе, рекомендуется придерживаться некоторых принципов:

- **Используйте контрастный текст.** Убедитесь, что текст хорошо контрастирует с фоном и что не используются дополнительные цвета, способные повлиять на удобочитаемость текста. В общем случае мы стремимся к 80% контраста.
- **Используйте подходящий шрифт и кегль (размер).** Как правило, шрифт без засечек и с резкими контурами, такой как Verdana или Tahoma, читается лучше всего. Шрифты с засечками, вроде Times и Georgia, на экране могут выглядеть несколько «неопрятно», но с этим часто можно справиться путем увеличения размера шрифта и сглаживания контуров. Текст мельче 10 пикселов в большинстве ситуаций трудно читать, так что если требуется мелкий текст, обычно лучший выбор – это шрифт без засечек и без сглаживания.
- **Четко формулируйте мысли**. Пользуйтесь минимальным количеством слов, необходимым для ясной передачи смысла. Избегайте сокращений. Если они все-таки необходимы, используйте общепринятые.

Цвет в графических интерфейсах

Цвет – важный аспект практически любого графического интерфейса. Сегодня, в эпоху вездесущих цветных ЖК-дисплеев, люди ожидают увидеть цветной интерфейс даже в портативных компьютерах и телефонах. Но цвет – это нечто большее, чем просто маркетинговый ход; это мощное выразительное средство визуального представления данных и дизайна графического пользовательского интерфейса. Его можно применять с большим эффектом, но им легко и злоупотребить.

Цвет является составной частью визуального языка интерфейса, и пользователи будут пытаться усмотреть смысл в том, как он применяется. В большинстве приложений цвет должен использоваться сдержанно и хорошо сочетаться с прочими элементами визуального языка – символами, пиктограммами, текстом – и пространственными отношениями между ними. Как уже говорилось выше, цвет, используемый надлежащим образом, позволяет привлекать внимание к важным элементам, обозначать связи, а также передавать информацию о состоянии и прочие сведения.

Если не проявлять осторожность, цвет очень легко применить неправильно. Вот самые распространенные ошибки:

- **Слишком много цветов.** Добавление одного цвета, выделяющего важные элементы в наборе, значительно сокращает время поиска. Добавление новых цветов приводит к дополнительному ускорению работы пользователя, но при семи и более цветах скорость поиска значительно падает. Разумно предположить, что сходные результаты будут получены при любом типе навигации по интерфейсу, поскольку число семь отражает количество элементов информации, единовременно сохраняемой в кратковременной памяти человека. Если цветов слишком много, мы тратим время на то, чтобы вспомнить, что означает каждый цвет в отдельности, и потому работа замедляется.
- **Использование насыщенных дополнительных цветов.** Дополнительными являются цвета, противоположные друг другу в числовом представлении. Если такие цвета обладают достаточно высокой насыщенностью и расположены рядом, то порождают зрительные эффекты, препятствующие легкому восприятию и мешающие сосредоточить внимание. Аналогичная ситуация возникает при **хромостереопсисе**, когда цвета с противоположных концов спектра «вибрируют», будучи расположенными по соседству. Попробуйте поразглядывать насыщенный красный текст на насыщенном синем фоне и отметьте, как быстро начинает болеть голова!
- **Чрезмерная насыщенность.** Сильно насыщенные цвета выглядят кричаще и привлекают слишком много внимания. Умеренное насыщение цвета допустимо для небольших областей, привлекающих внимание пользователей, но такие области всегда следует создавать с осторожностью. Когда несколько насыщенных цветов используются вместе, может возникнуть хромостереопсис, а также иные артефакты восприятия.
- **Недостаточный контраст.** Когда цвет объекта отличается от цветов фона лишь оттенком, но не насыщенностью или яркостью, объект становится трудно воспринимать. Фигура и фон должны различаться по яркости и насыщенности, а не только по оттенку. Кроме того, необходимо избегать использования цветного текста на цветном фоне везде, где только возможно.

- **Недостаточная забота о людях с нарушениями цветового восприятия.** Примерно десять процентов мужского населения страдает цветовой слепотой той или иной степени. Это означает, что при использовании (в частности) оттенков красного и зеленого для передачи важной информации следует проявлять внимательность. Любые цвета, применяемые в интерфейсе, должны заметно различаться по насыщенности или яркости. Если интерфейс остается читаемым после преобразования в черно-белый вариант, люди с нарушениями цветового восприятия смогут работать и с цветным вариантом интерфейса. Существуют приложения и фильтры, в частности, созданный Fujitsu ColorDoctor, позволяющие понять, каким увидят ваш продукт люди с различными нарушениями цветового восприятия.

Визуальный дизайн интерфейсов для портативных и прочих устройств

Многие из подходов, обсуждавшихся в этой главе, выросли из проектирования для настольных компьютеров и, соответственно, предполагают большой размер экрана и стационарный контекст использования. Очевидно, проектирование для портативного компьютера, мобильного телефона или медицинского прибора связано с иным набором проблем: здесь и маленький экран, и мобильный контекст использования, и другие методы ввода. Приводимый далее список, конечно же, неполон, но следует принимать во внимание хотя бы эти тонкости:

- **Элементы управления на экране должны бросаться в глаза.** Портативные устройства используются людьми, которые стоят, ходят, перемещаются на тряских поездах, а также находятся в разнообразном оживленном производственном или медицинском окружении, так что элементы управления на экране должны быть гораздо более очевидными, чем их аналоги в настольных приложениях. Иное аппаратное обеспечение и иные контексты требуют другой тактики, но в целом неплохо работают высококонтрастные схемы. Если устройство имеет ограниченные возможности в плане контраста, возможно, для достижения нужного эффекта придется подстраивать размер, цвет или толщину линий.
- **Создавайте визуальные якоря.** Для решения задачи пользователю портативного устройства часто приходится пройти через несколько экранов. Следовательно, важно применять визуальные якоря, помогающие пользователя ориентироваться.
- **Элементы управления на сенсорном экране должны быть крупнее.** Если ваше устройство содержит сенсорный экран, элементы управления следует делать достаточно крупными, чтобы их можно было активировать пальцами. Перья часто теряются, и потому (а еще из-за максимализма компьютерных фанатов) более молодые пользователи не воспринимают работу с пером.

- **Используйте крупные шрифты без засечек.** Мелкие шрифты с засечками трудно читать; поскольку экраны портативных устройств имеют низкое разрешение, следует использовать шрифты без засечек.
- **Четко указывайте наличие дополнительной информации за пределами экрана.** Многие люди не привыкли к идее маленького экрана, требующего прокрутки информации. Если данных больше, чем помещается на экране, не забудьте подчеркнуть это обстоятельство. В идеале дайте пользователю понять, как получить доступ к дополнительным данным.

Принципы визуального информационного дизайна

Подобно разработке визуального интерфейса разработка визуального представления информации имеет свои принципы, и дизайнер может применять их для достижения наилучшего результата. Специалист в области представления информации Эдвард Тафти утверждает, что хороший визуальный дизайн – это «визуализированная ясная мысль» и что такой дизайн достигается через понимание «когнитивной задачи» (цели) зрителя путем использования ряда принципов проектирования.

Тафти утверждает, что в сфере визуального представления информации существуют две важных проблемы:

1. Трудно визуализировать многомерную информацию (информацию с более чем двумя переменными) на двухмерной поверхности.
2. Разрешение конечного носителя не всегда достаточно велико для вывода плотной информации. Вывод информации на дисплей компьютера представляет особую трудность: хотя компьютер позволяет добавить динамику и интерактивность, разрешение дисплея ниже разрешения, достижимого на бумажном носителе.

И хотя оба утверждения справедливы, дизайнер графического интерфейса имеет одно преимуществом перед информационным дизайнером, работающим с бумагой: интерактивность. Бумажный носитель должен передавать сразу весь объем информации, тогда как электронный дисплей может раскрывать информацию постепенно – сообразно потребностям пользователя. Это позволяет компенсировать (по крайней мере частично) ограничения, накладываемые разрешением.

Несмотря на различия между печатной и цифровой средами, существуют универсальные принципы информационного дизайна, не зависящие от языка, культуры и времени, которые помогают повысить эффективность любого представления информации.

В прекрасно оформленной работе «The Visual Display of Quantitative Information» (Tufte, 1983) Тафти выдвигает семь Великих Принципов,

которые кратко обсуждаются в следующих разделах, поскольку они имеют отношение к цифровой информации и интерфейсам.

По Тафти, информация, представленная визуально, должна:

- способствовать визуальному сравнению;
- показывать причинно-следственную связь;
- отображать сразу несколько величин;
- объединять текст, графику и данные в одном изображении;
- гарантировать качество, релевантность и целостность данных;
- группировать объекты в пространстве, а не во времени;
- представлять числовые данные в числовом виде.

Кратко обсудим эти принципы в той степени, в какой они применимы к визуальному представлению информации в цифровых средах.

Визуальное сравнение

Предоставьте пользователям возможность сравнивать взаимосвязанные переменные и тенденции либо сопоставлять варианты «до» и «после». Сравнение создает контекст, делающий информацию более ценной и понятной пользователям (пример см. на рис. 14.9). Adobe Photoshop, как и многие другие инструменты для работы с графикой, активно использует предварительный просмотр, что позволяет пользователям

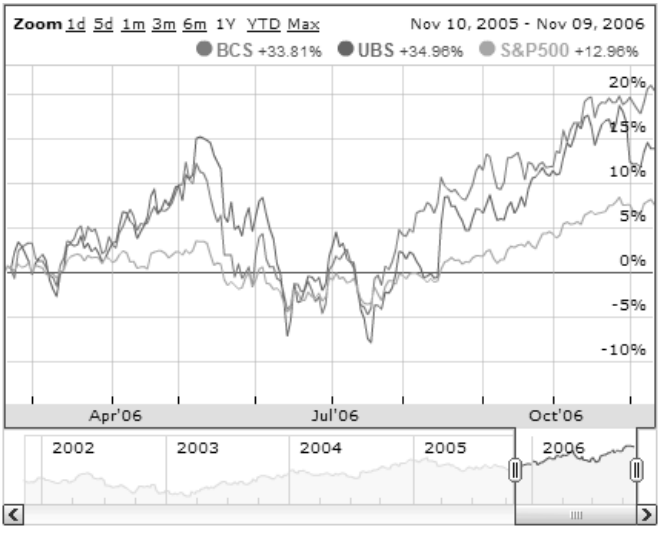


Рис. 14.9. *Этот график из приложения Google Finance сравнивает показатели двух компаний с индексом S&P 500 за определенный период времени. Визуально наблюдаемые закономерности позволяют зрителю понять, что между акциями Barclays Bank (BCS) и UBS существует корреляция, но акции обеих компаний достаточно слабо связаны с индексом S&P 500*

легко сравнивать в интерактивном режиме картину до и после выполнения операции (см. рис. 2.2).

Причинно-следственная связь

Представляя информацию в графическом виде, четко обозначайте причину и следствие. В своей книге Тафти приводит классический пример – катастрофу многоразового космического корабля «Челленджер», которую можно было предотвратить, если бы данные, графически представленные специалистами NASA, более ясно отражали связь между температурой воздуха при старте и серьезностью последствий, вызванных дефектом уплотнительного кольца. В интерактивных интерфейсах необходимо обеспечить немодальную визуальную обратную связь (см. главу 25), чтобы информировать пользователей о последствиях их действий или предоставить им подсказку о том, как следует действовать.

Множественные величины

Окна, содержащие информацию о нескольких взаимосвязанных переменных, должны в случае необходимости отображать все эти переменные одновременно без ущерба для ясности. При этом у пользователя должна быть возможность избирательно включать и отключать вывод величин этих переменных в интерактивном режиме, чтобы облегчить их сравнение и поиски корреляции (причинно-следственной связи). Инвесторы обычно интересуются корреляциями различных ценных бумаг, индексов и индикаторов. Построение графиков изменения нескольких переменных по времени позволяет выявлять такие зависимости (рис. 14.9).

Объединение текста, графики и данных

Диаграммы, требующие дополнительных подписей, легенд или расшифровок, нагружают пользователя добавочной когнитивной обработкой. Чтение и расшифровка дополнительных подписей – еще одна форма навигационного интерфейсного налога. Пользователю приходится переключать внимание с диаграммы на подпись и обратно, а затем соотносить их в уме. Рисунок 14.10 демонстрирует пример интерактивного окна, где интегрированы текст, графика и данные, а также ввод и вывод – весьма эффективное сочетание с точки зрения пользователя.

Обеспечение качества, релевантности, целостности данных

Не следует выводить информацию на экран только потому, что это технически возможно. Убедитесь, что любая отображаемая информация помогает пользователю достигать конкретных целей и уместна в дан-

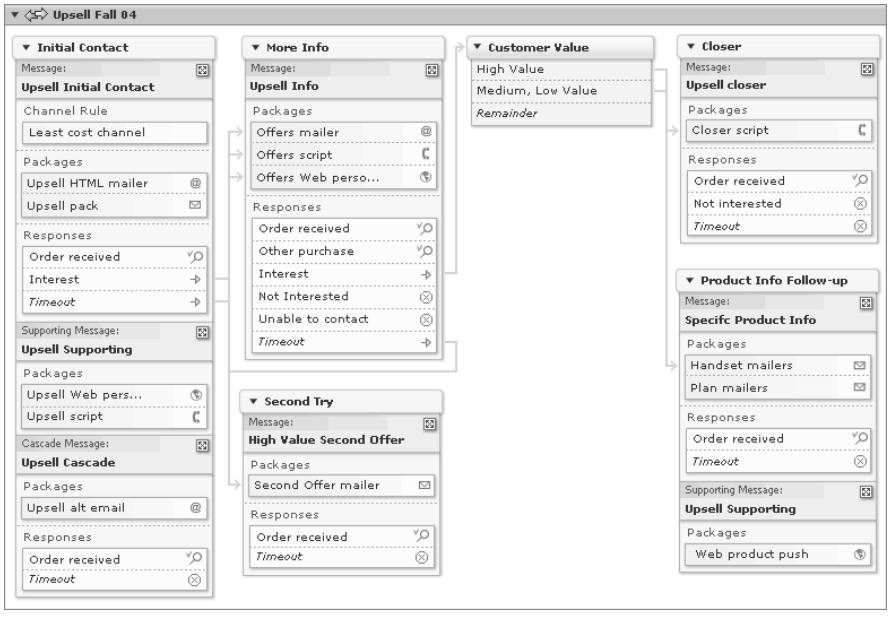


Рис. 14.10. *Данный «План коммуникаций» – элемент интерфейса из приложения для управления исходящими маркетинговыми кампаниями, спроектированного компанией Cooper. Элемент обеспечивает визуальную структуру текстовой информации, которая в свою очередь дополняется пиктографическими обозначениями объектов различных типов. Этот инструмент не только организует вывод структуры плана коммуникаций, но и позволяет пользователю напрямую изменять эту структуру посредством перетаскивания элементов*

ном контексте. Ненадежная (и в любом другом смысле некачественная) информация подорвет доверие пользователей, которое вы пытаетесь заслужить с помощью организации поведения и содержания программного продукта, а также с помощью его визуального брендинга.

Пространственная группировка объектов

Когда вы отображаете данные, меняющиеся во времени, пользователю легче воспринимать их динамику, если значения выводятся рядом, а не перекрываются. Комиксы – хороший пример пространственной организации событий, развивающихся во времени.

Естественно, этот совет относится к статическим информационным окнам; в программном продукте, если позволяют технические возможности (память компьютера или скорость интернет-соединения), для эффективного отображения меняющейся информации можно пользоваться **анимацией.**

Числовое представление числовых данных

Даже если вы предпочитаете пользоваться графиками и диаграммами, чтобы облегчать восприятие тенденций и иной количественной информации, не следует полностью отказываться от вывода собственно числовых данных. Например, круговая диаграмма в диалоговом окне Свойства диска (Windows) позволяет пользователю составить примерное представление о том, сколько свободного места осталось на диске, но здесь же выводится точное количество занятых и свободных килобайтов.

Единство и стандарты

Многие юзабилити-команды внутри компаний считают себя кроме прочего стражами единства оформления в дизайне цифровых продуктов. **Единство** предполагает сходство внешнего вида и поведения различных модулей программного продукта, и в некоторых случаях такой подход распространяется на всю продукцию данного производителя. Особенно актуальны вопросы единства оформления и бренда для крупных фирм, таких как Microsoft и Adobe, которые регулярно приобретают права на программные продукты у мелких фирм. Очевидно, что они заинтересованы в придании каждому приобретенному приложению внешнего вида, свидетельствующего о его принадлежности к первоклассным разработкам компании. Кроме того, Apple и Microsoft заинтересованы в том, чтобы как их собственная продукция, так и приложения, созданные сторонними разработчиками, выглядели и вели себя соответственно операционной системе, в которой они работают. Тогда пользователь будет считать платформу удобной и свободной от внутренних несообразностей.

Выгоды от стандартов на интерфейсы

Стандарты, относящиеся к пользовательским интерфейсам, способствуют достижению этих целей, хотя и ценой определенных затрат. Должное соблюдение стандартов идет во благо пользователю. Якоб Нильсен считает, что следование определенному стандарту сокращает время, затрачиваемое пользователем на изучение интерфейса, и повышает производительность труда пользователя за счет роста скорости работы и сокращения количества ошибок. Этот выигрыш возникает благодаря способности пользователей предсказывать поведение программы на основе предыдущего опыта работы с другими частями интерфейса или с другими программами, удовлетворяющими тем же стандартам.

В то же время соблюдение стандартов приносит выгоду и производителям программных продуктов. Затраты на обучение клиентов и техническую поддержку сокращаются, поскольку единство оформления, обеспечиваемое стандартами, упрощает обучение и облегчает работу с программой. Время и средства, потраченные на разработку, также

сокращаются, поскольку формальные стандарты на интерфейсы предлагают готовые решения, избавляя разработчиков от долгих дебатов на производственных совещаниях. Наконец, качественные стандарты сокращают затраты на сопровождение и способствуют повторному использованию кода и решений проектирования.

Опасности стандартов на интерфейсы

Главная опасность, исходящая от любого стандарта, состоит в том, что соответствующий стандарту продукт хорош лишь настолько, насколько хорош сам стандарт. Разработчики стандарта должны быть исключительно внимательны и первым делом убедиться, что, как говорит Нильсен, стандарт описывает действительно удобный интерфейс и подойдет *разработчикам*, которым придется создавать интерфейс в соответствии со спецификациями.

Кроме того, рискованно видеть в стандартах универсальное средство для создания хорошего интерфейса. Предполагать, что стандарт является панацеей от всех проблем проектирования интерфейсов, – все равно что утверждать, будто для написания хорошего романа достаточно подробного справочника по русскому языку. Большинство стандартов для интерфейсов ориентированы на *синтаксис* интерфейса, то есть на его внешний вид, но мало говорят о глубинном поведении и о высокоуровневой логической и организационной структуре. И тому есть причины: общий стандарт на интерфейс не включает в себя информацию о контекстах конкретных реализаций. В нем не учитывается специфическое поведение пользователей и приемы их работы в рамках контекста; вместо этого в центр внимания помещаются общие вопросы восприятия и познания и иногда визуальный брендинг. Эти вопросы важны, но они относятся к представительской стороне дела. Правила построения взаимодействия с пользователем, на которые и опирается инфраструктура интерфейса, в стандартах отсутствуют.

Стандарты, рекомендации и общие правила

Стандарты, безусловно, полезны, но они должны эволюционировать в соответствии с развитием технологии и расширением наших знаний о пользователях и их целях. Некоторые проектировщики и программисты относятся к стандартам интерфейса компаний Apple и Microsoft так, словно те были записаны на скрижалях на горе Синай. Обе компании широко публикуют свои стандарты пользовательских интерфейсов, но сами при этом свободно и часто нарушают их, обновляя рекомендации уже по факту. Когда компания Microsoft предлагает стандарт интерфейса, она чувствует себя вправе заменить его чем-то более подходящим в следующей версии. И это совершенно естественно: проектирование интерфейсов все еще переживает период младенчества, и было бы неразумно усматривать пользу в стандартах, которые сдерживают подлинное новаторство. Резкий в визуальном плане

переход компании Apple от OS 9 к OS X в некоторой степени способствовал рассеиванию бытовавшего среди поклонников Mac убеждения, что стандарты высечены на граните. Первая версия Macintosh была выдающимся достижением именно потому, что выходила за рамки всех предыдущих платформ и стандартов компании Apple. С другой стороны, успех пришел к Mac благодаря тому, что производители программных продуктов следовали образцу, заданному Apple, и создавали интерфейсы, которые выглядели, работали и вели себя как продукция этой компании. Аналогично, многие успешные программы для Windows без тени смущения копируют Word, Excel и Outlook.

Таким образом, к стандартам лучше всего относиться как к неким *рекомендациям* или *общим правилам*. Излишне строгое соблюдение этих рекомендаций или следование им без учета нужд пользователей в конкретном контексте может привести к втискиванию интерфейса в рамки неподходящей модели взаимодействия.

Когда можно нарушать рекомендации

Так что же нам делать с рекомендациями по разработке интерфейса? Вместо того чтобы спрашивать, следует ли *придерживаться* стандартов, зададим другой вопрос: «Когда следует *нарушать* стандарты?» Тогда и только тогда, когда на то есть веская причина.

Придерживайтесь стандарта, если нет действительно стоящей альтернативы.

Но что такое «веская причина»? Более удачная идиома? Оценка удачности идиомы, как правило, весьма затруднительна, поскольку не может быть выполнена исключительно количественными методами. Лучший ответ на этот вопрос будет таким: если большинство целевых пользователей (персонажей), проверивших некоторую идиому на практике, согласны с тем, что она значительно лучше, имеет смысл использовать именно ее. Именно так получили право на существование панели инструментов, представление разметки страницы, вкладки и другие идиомы. Возможно, исследователи и изучали их в лабораториях, но успех к этим идиомам пришел благодаря их полезному присутствию в реальных программных продуктах.

Может оказаться, что причины, по которым разработчик решил отклониться от стандарта, недостаточно оправданны, – и конечный продукт пострадает. Однако вы и другие проектировщики сможете извлечь урок из своей ошибки. Кристофер Александер называл это «спонтанно-естественным процессом», имея в виду врожденный и малоизученный процесс медленного продвижения в попытке улучшить принятое решение. Новые идиомы (а также новые способы использования старых идиом) – фактор риска. Вот почему так важны тщатель-

ное целеориентированное проектирование и соответствующее тестирование на реальных пользователях в реальной рабочей обстановке.

Единство и стандарты за рамками одного приложения

Применение стандартов и рекомендаций становится непростой задачей, когда компания, продающая разнообразные программные продукты, решает, что все они должны иметь единообразный пользовательский интерфейс.

Как говорилось выше, это имеет смысл с точки зрения создания и поддержки визуального бренда, хотя и влечет за собой определенные сложности. Если анализ персонажей и рынков показывает, что пользователи двух разных продуктов образуют непересекающиеся группы, а их цели и потребности расходятся, то возникает вопрос: а не разумнее ли разработать два разных визуальных бренда, адресованных конкретным категориям пользователей, чем продвигать один, не имеющий четкого адресата? Когда речь заходит о поведении программного продукта, вопрос становится еще более животрепещущим. Единый стандарт *может* играть важную роль, если клиенты пользуются несколькими продуктами как единым комплексом. Но даже в этом случае разумно спросить: должно ли графическое приложение для презентаций, такое как PowerPoint, иметь интерфейс, сходный с интерфейсом текстового редактора, например Word? Компания Microsoft имела благие намерения, но зашла слишком далеко в реализации глобального стилевого стандарта. Приложение PowerPoint мало выиграло, получив структуру меню, аналогичную меню Excel и Word, зато много потеряло в простоте использования, поскольку придерживается чуждого интерфейса, не соответствующего ментальным моделям пользователей. С другой стороны, проектировщики где-то все же провели границу – и в PowerPoint появилось окно сортировки слайдов – элемент интерфейса, не встречающийся больше нигде.

ПРИНЦИП проектирования

Единство оформления не подразумевает косности в решениях.

Таким образом, проектировщики должны помнить, что поддержание единообразия не заменяет гибкости в принятии решений, особенно когда эта гибкость жизненно необходима. Рекомендации, касающиеся интерфейса и взаимодействия, должны развиваться и эволюционировать вместе с программной продукцией, к которой они относятся. Иногда приходится отклоняться от правил, чтобы лучше обслужить пользователей и поработать на их цели (а иногда и на цели вашей компании). Когда возникает такая ситуация, старайтесь вносить изменения и новые рекомендации таким образом, чтобы не противоречить стандартам. Руководствуйтесь духом, а не буквой закона.

III

Детальное проектирование взаимодействия

Глава 15. Совершенствуем поиск и извлечение данных
Глава 16. Отмена
Глава 17. Новый взгляд на файлы и операцию сохранения
Глава 18. Улучшаем ввод данных
Глава 19. Указание, выделение, непосредственное манипулирование
Глава 20. Поведение окон
Глава 21. Элементы управления
Глава 22. Меню
Глава 23. Панели инструментов
Глава 24. Диалоговые окна
Глава 25. Ошибки, уведомления, подтверждения
Глава 26. Проектирование для различных потребностей

15

Совершенствуем поиск и извлечение данных

Новый цифровой мир, в котором мы сегодня живем, поражает огромными объемами информации, доступной в приложениях, на портативных компьютерах и устройствах, в сетях и во Всемирной паутине. К сожалению, безграничным возможностям доступа к информации сопутствует и серьезная проблема проектирования: как сделать так, чтобы люди легко находили то, что ищут, и – еще более важно – чтобы они находили именно то, что им нужно?

Хорошие новости: огромных успехов в этой области добиваются компании, подобные Google с ее разнообразными поисковыми машинами и Apple, создавшей для системы Mac OS X высокоэффективный механизм поиска Spotlight (мы о нем еще поговорим). Однако, хотя эти решения и указывают направление разработки эффективного взаимодействия, они являются лишь отправной точкой. Google – очень удобный способ поиска текста и видео во Всемирной паутине, но это не означает, что такие же шаблоны взаимодействия подойдут для сценария поиска с иным прицелом.

Как в случае большинства задач в области проектирования взаимодействия, мы обнаружили, что создание пригодного решения следует начинать с глубокого понимания ментальных моделей пользователей и контекстов использования. Эта информация позволяет структурировать системы хранения и извлечения информации исходя из их конкретного назначения. В этой главе мы обсудим методы поиска данных с точки зрения взаимодействия и представим некоторые ориентированные на человека подходы к решению проблемы отыскания полезной информации.

Системы хранения и извлечения информации

Система хранения есть способ безопасного содержания предметов на складе. Она состоит из хранилища и приспособлений для помещения объектов в хранилище и изъятия их из хранилища. **Система извлечения** обеспечивает поиск предметов в хранилище исходя из связанных с предметами свойств, таких как название или порядковый номер.

В реальном мире операции сохранения и извлечения неразрывно связаны. Если мы кладем вещь на полку (помещаем на хранение), то получаем также способ позже найти эту вещь (взять ее с полки). В цифровом мире единственное, что жестко связывает операции сохранения и извлечения, – наше недостаточно совершенное мышление. Компьютеры способны на невероятно изощренные методы извлечения информации – при условии, что мы сможем выйти за пределы традиционного мышления.

Цифровые механизмы хранения и извлечения традиционно основывались на понятии «папки» («директории» или «каталога» в системах UNIX). Несомненно, метафора папки дала нам пригодный способ обращаться с компьютерными системами хранения и извлечения (способ, во многом похожий на способ хранения физических объектов), но, как говорилось в главе 13, метафорическая природа этого шаблона взаимодействия является ограничителем. В конечном счете, использование папок и директорий в качестве основного механизма извлечения информации требует от пользователей знания о том, где сохранена эта информация. Это печально, поскольку цифровые системы способны обеспечить значительно более удобные способы извлечения информации, чем механические системы. Однако прежде чем обсуждать способы совершенствования поиска и получения информации, рассмотрим состояние дел в этой области.

Хранение и извлечение в физическом мире

Мы можем владеть книгой или молотком, не давая им специальных названий и не выделяя постоянного места хранения в доме. Книгу можно идентифицировать не только по названию, но, например, по цвету обложки или по формату. Однако когда предметов становится достаточно много, поиск нужного упрощается, если мы храним их организованно.

Всё на своих местах: хранение и извлечение по местоположению

Для книг и молотков нужно подходящее место, где мы их и найдем, когда потребуется. Мы не можем просто свистнуть и ждать, пока они прибегут. Мы должны знать, где они лежат, чтобы пойти туда и взять их. В физическом мире местоположение предмета является средством найти его. Помнить, куда мы положили вещь (то есть ее адрес), крайне

важно как для поиска вещи, так и для того, чтобы вернуть ее на место. Когда, например, нам нужна ложка, мы идем к буфету, в котором храним ложки. При поиске ложки мы не используем какие-то ее характеристики. Аналогичным образом, когда нам нужна книга, мы идем туда, где ее оставили, или предполагаем, что она хранится вместе с другими книгами. Мы не ищем книгу по ее характеристикам, например по содержанию.

В этой модели система хранения объединена с системой извлечения. Обе основаны на запоминании местоположения. Это спаренная система хранения и извлечения.

Извлечение по указателю

Система, при которой *все лежит на своем месте*, представляется довольно удачной, но имеет один недостаток: она ограничена возможностями человеческой памяти. Она прекрасно работает в отношении книг, молотков и ложек в вашем доме, но не подходит для книг, хранящихся, например, в Библиотеке Конгресса США.

В отношении книг на библиотечных полках мы пользуемся другим инструментом поиска – десятичной классификацией Дьюи (или ее международным вариантом). Каждой книге назначается уникальный «позывной» на основании ее темы. Книги располагаются на полках в соответствии с их номерами (а затем сортируются по фамилиям авторов). Получается библиотека, где книги сгруппированы по тематике.

Остается лишь каким-то образом выяснить номер нужной книги. Естественно, никто не в состоянии запомнить все номера. Решением задачи является **указатель** – сборник записей, позволяющих определить *расположение* вещи по ее **атрибуту**, скажем по названию.

Традиционные библиотечные каталоги на карточках предоставляют возможности поиска по трем атрибутам: автор, тема и название. Когда книга принимается на хранение в библиотеку и получает номер, на нее заводятся три карточки, и в каждую заносятся сведения о книге, включая десятичный номер по классификации Дьюи. В заголовке каждой карточки находится имя автора книги, название темы или название книги. Затем карточки помещаются в соответствующие разделы каталога в алфавитном порядке. Когда вам нужна книга, вы ищете ее в одном из указателей и выясняете ее номер. Затем вы находите стеллаж, на котором указан диапазон номеров, в который попадает предмет вашего поиска. Вы просматриваете полки, постепенно сужая диапазон номеров, пока не найдете нужную книгу.

Вы *физически* получаете книгу с помощью этой системы хранения, но вы *логически* ищете книгу с помощью системы поиска. Полки с номерами – система хранения. Карточки в каталоге – система извлечения. Вы идентифицируете положение книги в одной системе, но получаете ее в другой. В типичной университетской или публичной библиотеке

читатели не допускаются к полкам. Как читатель, вы пользуетесь только системой извлечения. Библиотекарь приносит вам книгу как пользователь системы хранения. Уникальный порядковый номер является мостиком между этими двумя взаимозависимыми системами. В физическом мире системы хранения и извлечения могут быть весьма трудоемкими в применении. В частности, в старых, некомпьютеризированных библиотеках обе системы были недостаточно гибкими. Добавление четвертого указателя, например по дате поступления, стало бы для такой библиотеки неразрешимой задачей.

Хранение и извлечение в цифровом мире

На компьютере создать новый указатель гораздо проще, чем в реальном мире, где приходится иметь дело с книгами, стопками бумаги и карточками. По иронии судьбы в системе, где наконец-то стало возможным создание динамических ассоциативных механизмов поиска, мы часто не реализуем *никакой* системы извлечения информации.

Если вы хотите найти файл на диске, вы должны знать его имя и путь к нему. Это все равно что сжечь карточный каталог в библиотеке и сказать читателям, что они легко найдут то, что им требуется, если будут запоминать номера, написанные на обложках книг. Мы полностью переложили ношу поиска файлов на память пользователей, а центральный процессор работает вхолостую, выполняя миллиарды команд `NOP`.[1]

Хотя наши настольные компьютеры могут вести сотни различных указателей, мы игнорируем эту возможность и часто не создаем никаких указателей на файлы, хранящиеся на дисках. Вместо этого, когда нам нужно найти файлы, мы вынуждены вспоминать, куда мы их положили и как назвали. Это упущение является одним из самых деструктивных. Это шаг назад в проектировании современного программного обеспечения. Такое положение дел можно объяснить тесной взаимозависимостью между файлами и организационными системами, в которых они существуют. В реальном мире такая взаимозависимость отсутствует.

С файловыми системами хранения, которые мы создали для себя, все в полном порядке. Сложность в том, что мы забыли создать адекватные системы *извлечения* файлов. Вместо этого мы вручаем пользователю систему хранения и называем ее системой извлечения. С таким же успехом можно вручить ему пакет с продуктами и назвать его вкусным ужином. Нет никаких причин менять системы хранения файлов. Модель UNIX замечательно работает. Наши приложения могут с легкостью запоминать имя и расположение файлов, с которыми работали, так что им система извлечения не нужна. Она нужна нам – людям.

---

[1] `NOP` (от `No Operation`) – инструкция процессора на языке ассемблера, которая предписывает ничего не делать. – *Примеч. науч. ред.*

Цифровые методы извлечения

Существуют три основных способа найти документ в компьютере. Вы можете вспомнить, где он находится в рамках файловой структуры, – это **позиционное извлечение**. Вы можете вспомнить его идентифицирующее имя – это **извлечение по названию**. Следует заметить, что позиционное извлечение и извлечение по названию используются совместно. Третий метод, **ассоциативное извлечение**, или **извлечение по атрибутам**, основан на возможности искать документ по какой-то характеристике, присущей самому документу. Например, если вы захотите найти книгу в красной обложке, или книгу, где обсуждаются системы перевозки грузов по узкоколейным железным дорогам, или книгу с фотографиями паровозов, или книгу, где упоминается Теодор Джуда (Theodore Judah)[1], то вам придется прибегнуть к ассоциативному извлечению.

Основой большинства цифровых систем хранения является сочетание позиционного извлечения и извлечения по названию. Однако ассоциативный метод хранения в основной массе цифровых систем не реализован. Игнорируя ассоциативный метод, мы теряем возможность искать по атрибутам и должны полагаться на человеческую память – вспоминать названия и расположение документов. Чтобы найти нужный документ, пользователь обязан знать его название, *а также* то, где документ хранится. Например, если вы хотите отыскать таблицу, в которой рассчитали амортизацию займа на покупку дома, вам придется вспомнить, что вы сохранили ее в каталоге «Дом» под именем «аморт1». Если вы не вспомните ни того, ни другого, поиск документа окажется достаточно сложной задачей.

Системы извлечения на основе атрибутов

В ранних системах с графическим пользовательским интерфейсом, таких как самый первый Macintosh, позиционная система извлечения имела определенный смысл. Она была продиктована метафорой рабочего стола (вы ведь не пользуетесь указателем при поиске бумажки на рабочем столе), а на дискете емкостью 144 Кбайт помещалось не так уж много документов. Современные настольные системы могут вмещать в 500 000 раз больше документов (не говоря уже о том, что скрыто в недрах скромной локальной сети)! Тем не менее для управления этими данными мы все еще пользуемся старыми метафорами и прежней моделью извлечения. Мы продолжаем создавать программные системы извлечения данных в строгом соответствии с моделью реализации системы хранения, пренебрегая той мощью и простотой, которыми могла бы обладать система для *поиска* файлов, отличная от системы для их *сохранения*.

---

[1] Теодор Джуда – американский инженер-железнодорожник, стоявший у истоков строительства Тихоокеанской железной дороги. – *Примеч. ред.*

Система, основанная на извлечении по атрибутам, позволила бы нам искать документы по содержимому и осмысленным свойствам (скажем, по времени последнего изменения). Назначение подобной системы – дать пользователям механизм, позволяющий описывать предмет поиска естественным образом. Например, сотрудница отдела продаж, которой необходимо найти коммерческое предложение, недавно отправленное клиенту «Widgetco», могла бы выразить свою мысль так: «Покажи мне все документы в формате Word, связанные с Widgetco, которые я вчера редактировала или выводила на печать».

Качественная система извлечения по атрибутам позволяет пользователям выполнять также поиск по синонимам, по смежным темам и по атрибутам или тегам (меткам), присвоенным отдельным документам. У пользователя появляется возможность динамически определять наборы документов, имеющих общие атрибуты. Вернемся к нашему примеру с сотрудницей отдела продаж. Предположим, она рассылает письма с предложениями всем потенциальным клиентам. Каждое письмо является индивидуальным и при этом естественным образом объединено с файлами, относящимися к конкретному клиенту. Однако между этими письмами существует определенная связь, поскольку они служат одной цели – предложению делового сотрудничества. Было бы удобно, если бы пользователь мог найти и собрать вместе все такие письма, сохранив уникальность каждого и его связь с соответствующим клиентом. Файловая система, жестко связанная с местоположением – с единственным местом хранения, – вынуждена хранить документы на основании единственного атрибута (клиент или тип документа), а не набора характеристик.

Система может многое узнать о документе, если будет достаточно внимательна к нему. Если система поиска по атрибутам запомнит хотя бы часть этой информации, она облегчит ношу, лежащую на плечах пользователя. Например, программа могла бы легко запомнить следующее:

- приложение, создавшее документ;
- содержание и формат документа;
- последнее приложение, открывавшее документ;
- размер документа и то, является ли документ необычно маленьким или большим;
- лежал ли документ на протяжении долгого времени без движения;
- продолжительность последнего сеанса работы с документом;
- объем информации, добавленной или удаленной при последнем редактировании;
- был ли документ создан с нуля или путем копирования другого документа;
- часто ли документ редактируется;
- часто ли документ просматривается;

- был ли документ распечатан, и если да, то на каком устройстве;
- часто ли документ выводился на печать, и вносились ли в него изменения каждый раз непосредственно перед печатью;
- отправлялся ли документ по факсу, и если да, то кому;
- отправлялся ли документ по электронной почте, и если да, то кому.

Spotlight – механизм поиска в операционной системе Apple Mac OS X – организует эффективную систему извлечения по атрибутам (рис. 15.1). Пользователь может не только искать документы по осмысленным свойствам, но и сохранять поисковые запросы в качестве «умных папок» (Smart Folders), позволяющих, в частности, видеть в одной папке все документы, относящиеся к одному конкретному клиенту, а в другой – все коммерческие предложения (при этом Spotlight все же не в состоянии самостоятельно распознать коммерческие предложения, и пользователю придется приложить некоторые усилия, чтобы обозначить их). Следует заметить, что полезность Spotlight во многом определяется скоростью получения результатов. Это важное отличие Spotlight от встроенного поиска Windows было реализовано с помощью специального технического решения, в котором содержимое жесткого диска индексируется системой во время простоя компьютера.

Система извлечения на основе атрибутов способна находить документы, не требуя от пользователя предварительных действий по организации хранения документов, однако возможность **помечать** документы тегами или присваивать им атрибуты имеет значительную ценность.

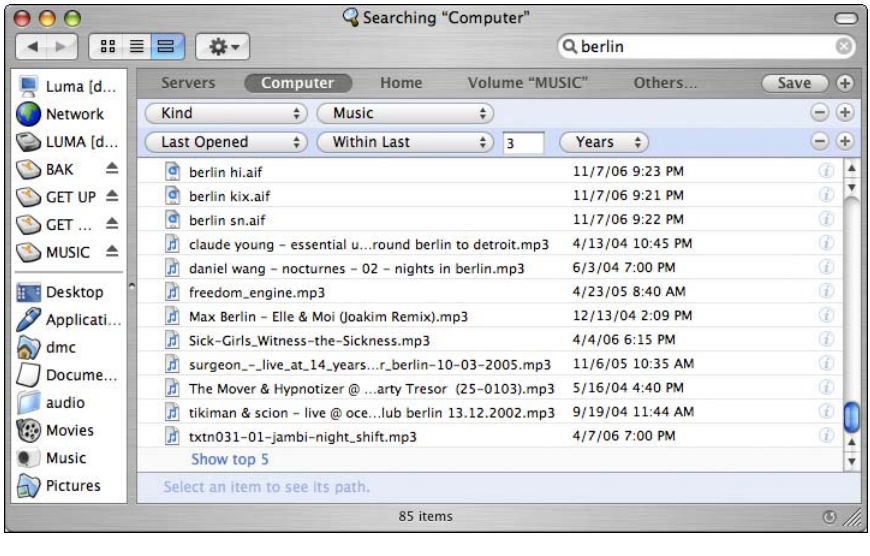


Рис. 15.1. *Spotlight, механизм поиска в Apple OS X, позволяет пользователям находить документы по осмысленным атрибутам, например по имени документа, его типу и дате последнего сеанса работы с ним*

Она не только позволяет пользователям заполнять пробелы в тех случаях, когда технология не способна самостоятельно выявить осмысленные атрибуты, но дает также людям возможность де-факто определять организационные схемы исходя из того, как они осознают и используют информацию. Механизм извлечения на основе таких меток часто называют «фолксономией» (то есть «народной таксономией»). Создание термина приписывается информационному архитектору Томасу Вандер Валу (Thomas Vander Wal). Фолксономии могут приносить особенно заметную пользу в социальных контекстах и совместной работе, где создают альтернативу глобальной таксономии, если нежелательно или неудобно заставлять всех следовать ей и думать в терминах нормализованной лексики. Этот подход применяется в системах поиска информации таких проектов, как Flickr, del.icio.us и LibraryThing (рис. 15.2), где люди имеют возможность просматривать и находить документы (фотографии и гиперссылки соответственно) на основе атрибутов, определенных самими пользователями.

Рис. 15.2. *LibraryThing – это веб-приложение, позволяющее пользователям вести каталоги своих книжных коллекций во Всемирной паутине с использованием системы тегов. Применение тегов ко всем книгам всех коллекций создало демократичную систему организации, основанную на том, как сообщество читателей описывает книги*

Реляционные базы данных и «цифровой бульон»

Программы, задействующие технологии баз данных, обычно предъявляют к пользователям два простых требования: Во-первых, пользователь должен заранее задать формат данных, а во-вторых, он должен впоследствии следовать заданному формату. Относительно пользователей программного продукта также справедливы два утверждения: во-первых, они редко способны заранее сформулировать, чего хотят, а во-вторых, даже если и способны, то, скорее всего, потом передумают.

Как организовать неорганизуемое

В век Интернета мы все чаще сталкиваемся с информационными системами, которые не проходят тест на соответствие реляционным критериям. Мы не только не можем определить информацию заранее, но даже не можем последовательно придерживаться какого-то мыслимого определения. Эту дилемму хорошо иллюстрируют, в частности, два явления, переживающие период бурного роста.

Первое из них – электронная почта. В то время как запись в базе данных имеет изначально заданное содержательное сходство с другими подобными ей записями и поэтому хранится в одной таблице с ними, сообщения электронной почты не вписываются в эту парадигму. Мы можем разделить почтовые сообщения на входящие и исходящие, но в этом мало проку. Предположим, вы получили письмо от Джерри насчет Салли и по поводу проекта «Аякс», который имеет отношения к фирме «Джонс Консалтинг» и к вашей совместной презентации на ближайшем совещании. Вы можете сохранить это письмо в папке «Джерри», или «Салли», или «Аякс» – но по-хорошему вы хотели бы сохранить его во всех трех папках. Через шесть месяцев у вас может появиться необходимость просмотреть это письмо по одной из множества непредсказуемых причин – и у вас должна быть возможность найти его независимо от причины.

Второе явление – это Всемирная паутина. Подобно бесконечному, неорганизованному, избыточному, бесхозному жесткому диску она сопротивляется любым попыткам структуризации. В Интернете доступны громадные объемы информации, но его размеры и явная неоднородность практически гарантируют, что сеть не поддастся какой-либо систематизации. Даже если бы Всемирную паутину можно было бы организовать, метод организации должен был бы находиться вовне, поскольку ее содержимое принадлежит миллионам личностей, ни одна из которых не признает власть над собой. В отличие от записей в базе данных, нельзя ожидать от документов в Интернете, что они будут содержать предсказуемые идентификационные пометки.

Сложности с базами данных

С базами данных связана еще одна проблема. Каждая запись в базе данных имеет один заранее определенный тип, а все однотипные записи сгруппированы. Запись в базе данных может быть представлением счета на оплату или клиента, но она никогда не является представлением счета на оплату и клиента *одновременно*. Аналогичным образом, поле записи может содержать имя клиента или номер его медицинской страховки, но оно никогда не содержит *и* имя, *и* номер страховки. Это фундаментальная концепция, лежащая в основе всех баз данных. Она служит жизненно важной цели – позволяет упорядочить систему хранения. К сожалению, она неспособна решить реальную проблему извлечения в примере с электронной почтой, приведенном выше. Недостаточно определить тип записи «электронное сообщение» для письма от Джерри. Мы должны каким-то образом назначить ему еще типы «Джерри», «Салли», «Аякс», «Джонс Консалтинг» и «Совещание». Мы должны иметь возможность добавить новый или изменить существующий тип даже после того, как письмо будет сохранено. Более того, тип «Аякс» может относиться к документам, не являющимся сообщениями электронной почты, например к планам проекта. Поскольку формат записи непредсказуем, значение, которое идентифицирует запись как относящуюся к проекту «Аякс», не может храниться вместе с самой записью. Это положение прямо противоречит принципам работы баз данных.

Базы данных способны предоставить более гибкие инструменты поиска и выборки записей, нежели поиск по простым типам данных. Они позволяют найти запись по ее содержимому, применяя заданные критерии поиска. Например, мы можем найти счет номер 77329 или клиента с идентифицирующей строкой «Goodyear Tire and Rubber». Тем не менее эти инструменты *не* помогают решить проблему идентификации электронного сообщения. Если мы разрешим пользователю помещать в запись ключевые слова «Джерри», «Салли», «Аякс», «Джонс Консалтинг» и «Совещание», мы должны будем заранее задать эти поля. Однако, как мы говорили, предварительное определение чего бы то ни было не гарантирует, что пользователь будет впоследствии придерживаться этого определения. Например, он может заняться поиском писем, относящихся к корпоративному пикнику. Добавление полей с ключевыми словами приведет нас к одной из фундаментальных загадок обработки данных: если вы предоставите пользователям десять полей, то кому-нибудь обязательно понадобится одиннадцатое.

Альтернатива с применением атрибутов

Итак, что же нам делать, если технология реляционных баз данных не годится? В ситуации, когда пользователям трудно задавать структуру информации заранее, как того требуют базы данных, найдется ли альтернативная система хранения и поиска, которая будет удовлетворять требованиям людей?

И снова решение требует разделения систем хранения и извлечения. Если в качестве системы извлечения воспользоваться указателем, то системой хранения информации может оставаться база данных. Мы можем представлять себе систему хранения в виде **«цифрового бульона»**, в который мы помещаем свои записи. «Бульон» принимает любые записи, независимо от размера, типа или содержимого. После ввода записи программа возвращает некое опознавательное имя, по которому можно найти запись. Все, что мы должны сделать, чтобы «цифровой бульон» выдал нам запись, – ввести ее опознавательное имя. Однако это только система хранения; необходима еще и система извлечения, управляющая этими опознавательными именами.

На помощь приходит извлечение по атрибуту. Мы можем создать указатель, который хранит ключевое значение вместе с копией опознавательного имени. Весь фокус заключается в том, что мы можем создать сколь угодно много указателей, каждый из которых будет содержать свой ключ и копию опознавательного имени. Например, если в «цифровом бульоне» содержатся все наши электронные письма, мы можем создать указатели для каждого из типов: «Джерри», «Салли», «Аякс», «Джонс Консалтинг» и «Совещание». Теперь, если мы захотим найти письмо, относящееся к совещанию, нам не придется вручную ворошить десятки папок, – один запрос вернет нам искомое.

Конечно, кто-то (или что-то) должен заполнить эти указатели, но это уже более рутинная задача из области проектирования взаимодействия. Внимания заслуживают два момента. Во-первых, система должна уметь читать сообщения электронной почты и автоматически извлекать из них информацию для указателя, например имена собственные, адреса в Интернете, адреса реальные, номера телефонов и прочие важные данные. Во-вторых, система должна предоставить пользователю возможность добавления специальных ссылок на сообщение, чтобы он мог явно указать, что данное сообщение относится к некой теме, независимо от того, упоминается ли данная тема в сообщении. Ввод с клавиатуры приемлем, но выбор из списка, перетаскивание и другие современные идиомы пользовательского интерфейса сделают решение этой задачи практически безболезненным для пользователя.

Подход, при котором важность системы хранения уменьшена, а система извлечения отделена от нее и значительно усилена, сулит большие выгоды. «Цифровой бульон» поможет нам управлять непредсказуемой информацией, которая занимает все больше места в нашем повседневном информационном пространстве. Мы можем предложить пользователям мощные инструменты управления информацией, не требуя, чтобы пользователи структурировали свою информацию заранее или чтобы они придерживались заданной структуры в будущем. Они ведь все равно не смогут это делать – так зачем же нам настаивать?

Вывод на естественном языке: идеальный интерфейс для извлечения по атрибутам

В предыдущих разделах этой главы мы обсуждали достоинства извлечения данных по атрибутам. Чтобы быть действительно успешной, такая система требует интерфейса, позволяющего пользователям легко ориентироваться в сложных и взаимосвязанных наборах атрибутов.

Один из возможных альтернативных подходов заключается в использовании естественного языка для формулирования запроса. Препятствием на этом пути, однако, становится то, что обычные современные компьютеры в большинстве случаев не способны эффективно обрабатывать запросы на естественном языке, касающиеся реальных рабочих ситуаций. Распознавание естественного языка возможно в лаборатории, в жестко контролируемых условиях, но не в реальном мире, где имеют место безграмотность, диалектизмы, разговорные выражения и прочие недоразумения. Так или иначе, программирование механизма распознавания естественного языка лежит за пределами возможностей и бюджета средней команды программистов.

Более приемлемый подход, успешно опробованный авторами, состоит в том, что мы назвали **выводом на естественном языке**. Использующая эту методику программа предлагает пользователю массив элементов управления, ограничивающих ввод. С их помощью пользователь выбирает нужные значения. Элементы расположены так, что образуют предложение на английском языке. Пользователь действует в рамках набора грамматически правильных вариантов, так что результатом проектирования становится ограниченная самодокументированная функция построения запроса. На рис. 15.3 приведен пример такого интерфейса.

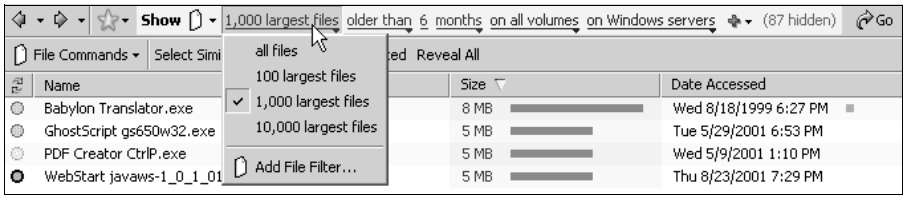


Рис. 15.3. *Пример интерфейса извлечения по атрибутам, основанного на естественном языке. Это фрагмент проекта, созданного фирмой Cooper для продукта Storage Manager по заказу Softek. Элементы управления, помогающие составить запрос к базе данных, предлагают естественный язык на выходе вместо того, чтобы пытаться принимать его в качестве входного. Щелчок по любой подчеркнутой конструкции раскрывает меню со списком вариантов. Пользователь конструирует предложение из динамического множества вариантов, гарантирующего корректный результат*

Интерфейс с выводом на естественном языке полностью подходит для составления запросов к традиционным реляционным базам данных. Большинству людей сложно работать с обычными запросами к базе данных, поскольку для их понимания необходимо знать булеву систему обозначений и мудреный синтаксис а-ля SQL. Мы уже обсуждали проблемы булевой нотации в главе 2. То, что программе требуются булевы запросы, еще не основание навязывать их составление пользователю.

Естественный язык не является языком булевой алгебры, и предложения в нем соединяются не союзами AND (И) или OR (ИЛИ), а фразами типа «все перечисленные» и «не все перечисленные». Пользователю проще выбрать из этих фраз, потому что они ясны и осмысленны, а когда он составит предложение, то сможет прочитать его, чтобы проверить правильность.

С точки зрения программиста, самое сложное в выводе на естественном языке состоит в том, что выбор одного из элементов в начале предложения очень часто меняет набор вариантов в остальных частях предложения. Это означает, что для эффективной реализации вывода на естественном языке грамматика предложений должна быть хорошо продумана заранее и что элементы управления должны появляться и скрываться динамически, в зависимости от того, что выбрано в остальных элементах управления. Кроме того, сами элементы должны быть в состоянии выводить или по меньшей мере загружать данные динамически.

Еще одним сложным вопросом является локализация. Если вы разрабатываете программу для пользователей, говорящих на разных языках, а порядок слов в их языках различается (как, например, в немецком и английском), то, возможно, вам потребуется многоязычная система грамматического ассоциирования.

Как системы поиска по атрибутам, так и интерфейсы с выводом на естественном языке требуют значительных усилий от проектировщиков и программистов, но зато приносят огромную выгоду пользователям, предоставляя им мощную и гибкую систему управления данными. Поскольку объем данных, которыми нам приходится управлять, растет экспоненциально, имеет смысл вкладываться в разработку этих более мощных целеориентированных инструментов управления данными уже сейчас.

16

Отмена

Отмена – замечательная функция, позволяющая отказаться от совершённого действия. Ценность этой простой и элегантной возможности очевидна. Тем не менее если рассмотреть имеющиеся реализации и применения этой функции с точки зрения целеориентированного подхода, обнаружатся значительные вариации ее назначения и принципа действия. Функция отмены чрезвычайно важна для пользователей и не так проста, как может показаться на первый взгляд. В этой главе исследуются различные представления пользователей об этой функции и разнообразные способы ее применения.

Пользователи и отмена

Отмена традиционно воспринимается как метод спасения пользователя в беде – эдакий рыцарь в сверкающих доспехах, или кавалькада, скачущая по холмам, или супергерой, влетающий в кадр в последнюю секунду.

Как вычислительная функция отмена не имеет никаких достоинств. Компьютеры не нуждаются в ней, поскольку никогда не ошибаются. Зато люди совершают ошибки постоянно, и эта функциональная возможность существует исключительно для них. Одно только это наблюдение немедленно приводит нас к заключению, что из всех функциональных возможностей программы модель отмены должна в наименьшей степени соответствовать модели реализации и в наибольшей степени – ментальной модели пользователей.

Люди не просто совершают ошибки. Ошибки – это часть их повседневного поведения. С точки зрения компьютера фальстарт, взгляд в неверном направлении, пауза, чихание, экспериментирование, разнообразные «ну» и «как бы» – все это ошибки. Однако с человеческой точки зрения они вполне нормальны. Ошибки человека настолько обыч-

ны, что если вы будете расценивать их как неправильное, «ошибочное» поведение, то это отрицательно скажется на дизайне вашего программного продукта.

Ошибки: ментальные модели пользователей

Пользователи, как правило, не верят или, по меньшей мере, не желают верить, что совершают ошибки. Иначе говоря, ментальная модель пользователя обычно не предполагает ошибок с его стороны. Следовать ментальной модели пользователя означает прощать его ошибки и отказаться от обвинений в его адрес. В то же время модель реализации основывается на безошибочном поведении центрального процессора, и следование модели реализации имеет в своей основе предположение, что вся вина лежит на пользователе. Большинство программных продуктов исходят из убеждения, что они безупречны, а любые проблемы возникают исключительно из-за оплошностей пользователей.

Решением этой проблемы будет полный отказ проектировщика пользовательского интерфейса от мысли, что пользователь может совершить ошибку. Следует предполагать, что любые свои действия сам пользователь считает допустимыми и разумными. Люди не любят признавать свои ошибки – и программа не должна противоречить настрою пользователя при взаимодействии с ним.

Отмена поощряет экспериментирование

Если мы будем проектировать программные продукты исходя из убеждения, что никакие действия пользователя не являются ошибочными, то увидим вещи в новом свете. Мы перестанем считать пользователя модулем кода или периферийным устройством, которое управляет компьютером, и будем относиться к нему как к исследователю, экспериментирующему с неизвестным. Мы поймем, что исследование неизбежно включает в себя попадание в безвыходное положение. Людям свойственно экспериментировать, идти разными путями и приоткрывать завесу неизвестного, чтобы понять, где границы их возможностей. Как они смогут узнать, что именно можно делать с помощью данного инструмента, если не поэкспериментируют с ним? Конечно, у разных людей склонность к экспериментам развита в разной степени, но большинство людей экспериментирует хотя бы в малой степени.

Программисты, которым хорошо платят за то, что они мыслят, как компьютеры, рассматривают такое поведение как последовательность ошибочных действий, подлежащих обработке с помощью кода. В соответствии с моделью реализации (которая неизбежно отражает точку зрения программиста) такие осторожные и невинные эксперименты представляют собой непрерывную серию «ошибок». С нашей же более просвещенной точки зрения, опирающейся на ментальную пользовательскую модель, эти действия естественны и нормальны. Приложение стоит перед выбором – либо дать отпор этим предполагаемым ошибкам,

либо оказать пользователю помощь в его исследованиях. Таким образом, функция отмены в первую очередь является инструментом интерфейса, поддерживающим эксперименты. Она позволяет пользователю отказаться от одного или нескольких действий, если он передумал.

Важнейшая выгода от функции отмены – чисто психологическая: благодаря ей у пользователей возникает уверенность в своих действиях. Гораздо легче войти в пещеру, когда вы уверены, что в любой момент сможете выйти из нее. Функция отмены – это ведущая на поверхность веревочная лестница, поддерживающая в пользователе желание продолжать эксперименты и гарантирующая, что он сможет вернуться из любого тупика.

Интересно, что пользователи не задумываются о функции отмены, пока она им не понадобится. Они ведут себя как домовладельцы, которых не заботит вопрос страхования имущества, пока не случится стихийное бедствие. Люди часто отправляются в пещеры неподготовленными и начинают искать веревочную лестницу – аналог функции отмены, – когда уже оказались в беде.

Проектирование функции отмены

Хотя пользователи нуждаются в функции отмены, она не отвечает напрямую никаким конкретным целям, которые ставят перед собой пользователи. Скорее она обеспечивает необходимое условие – наличие уверенности на пути к поставленной цели. Она не способствует достижению пользователями их целей непосредственно, но не позволяет негативным событиям сводить на нет все усилия пользователей.

В зависимости от ситуации и собственных ожиданий пользователи поразному представляют себе функцию отмены. Неподготовленный пользователь видит в этой функции спасительную красную кнопку, выручающую его в безнадежно запутанной и опасной авантюре. Более искушенный пользователь может думать об отмене как о хранилище удаляемых данных. Действительно опытный пользователь с логическим складом ума понимает, что это стек процедур, действие которых может быть отменено в порядке, обратном порядку их выполнения. Чтобы эффективно реализовать функцию отмены, мы должны обеспечить ее соответствие ментальным моделям всех наших персонажей.

Секрет проектирования успешной системы отмены состоит в том, чтобы она поддерживала типичные инструменты и избегала любых намеков (визуальных, звуковых или текстовых) на то, что необходимость ее применения является следствием ошибки пользователя. Отмена должна быть не столько средством исправления ошибок, сколько средством поддержки экспериментирования. Ошибки, вообще говоря, являются одиночными неправильными действиями. Экспериментирование же, наоборот, представляет собой длинную последовательность попыток, результаты которых следует сохранять либо отбрасывать.

Функция отмены хорошо работает, когда внутри программы она реализована глобально и ее влияние распространяется на любые действия пользователя, будь то непосредственное манипулирование или операции в диалоговом окне. В существующих реализациях функции отмены есть серьезный минус. Если пользователь сохраняет документ (скажем, таблицу Excel), он теряет возможность откатывать свои действия. Тот факт, что человек сохранил документ, чтобы избежать потерь при сбое компьютера, не обязательно означает, что ему стали неинтересны все внесенные изменения. Более того, с учетом емкости жестких дисков нет причин *не* сохранять буфер отмены вместе с документом.

Помимо прочего, отмена может создавать проблемы в случае документов с внедренными объектами. Если пользователь внесет изменения в электронную таблицу, внедренную в документ Word, вернется к этому документу Word и затем вызовет функцию отмены, то будет отменено последнее действие Word, а не последняя операция над электронной таблицей. Пользователям трудно воспринимать такую логику. Мы заставляем их отбросить ментальную модель единого документа и думать в терминах модели реализации, где есть один документ, который внедрен в другой, и у этих документов раздельные буферы отмены.

Типы и варианты отмены

Как это вообще свойственно миру программного обеспечения, адекватной терминологии для описания различных существующих типов отмены нет. Все типы отмены называются просто «отменой». Этот языковой пробел способствует застою в разработке новых и более удачных вариантов отмены. В разделе ниже мы дадим определения разным видам отмены и обсудим различия между ними.

Модифицирующие и процедурные действия

Отмена работает с действиями пользователя. Типичное действие пользователя в типичном приложении имеет процедурную составляющую (что сделал пользователь) и зачастую информационную составляющую (на какие данные повлияло действие). Когда пользователь запрашивает отмену, происходит обращение процедурной составляющей, то есть действие совершается в обратном направлении. Если действие содержало информационную составляющую – пользователь добавил, изменил или удалил данные, – данные будут удалены, изменены или восстановлены соответственно. Вырезание, вставка, рисование, набор с клавиатуры и удаление – все эти действия имеют информационную составляющую, и их отмена влечет за собой удаление или замену соответствующих фрагментов текста или изображения. Действия, имеющие информационную составляющую, мы будем называть **модифицирующими**.

Многие отменяемые действия, такие как форматирование абзаца или поворот графического объекта, являются преобразованиями, не затра-

гивающими данные. Оба указанных действия работают с данными, но не создают, не изменяют и не удаляют их (с точки зрения базы данных – пользователь, конечно, может и не разделять эту точку зрения). Подобные действия (включающие в себя только процедурную составляющую) назовем **процедурными**. Большинство существующих систем отмены не проводят границу между процедурными и модифицирующими действиями, а просто обращают самую последнюю операцию.

Отмена вслепую и поясняющая отмена

В типичном случае функция отмены вызывается с помощью пункта меню или элемента панели инструментов с фиксированной пиктограммой или подписью. Пользователь знает, что данная идиома отменяет последнюю операцию, но индикация того, какая операция была последней, отсутствует. Это называется **отменой вслепую**. Если же идиома включает в себя текстовое или визуальное описание операции, которая будет отменена, мы имеем дело с **поясняющей отменой**.

Например, если последним действием пользователь набрал слово «дизайн», то пункт меню гласит: «Отменить ввод: **дизайн**». Вообще говоря, поясняющая отмена намного приятнее отмены вслепую. Включить описание операции в пункт меню легко, но вот поместить его на панели инструментов гораздо труднее. Хорошим компромиссом является всплывающая подсказка (более подробно о панелях инструментов и всплывающих подсказках см. в главе 23).

Одиночная и множественная отмена

В настоящее время двумя самыми распространенными типами отмены являются одиночная и множественная. **Одиночная отмена** – самый простой вариант. Она отменяет последнее действие, выполненное пользователем, – процедурное или модифицирующее. Вызов функции отмены два раза подряд обычно приводит к «отмене отмены» и возвращает систему к тому состоянию, в котором она находилась до первой отмены.

Такая возможность очень полезна, поскольку проста в обращении. Ее интерфейс прост и понятен, его легко описать и запомнить. Пользователь получает ровно один бесплатный завтрак. Это наиболее часто реализуемый тип отмены, и в большинстве программ он уместен, если не сказать – оптимален. Для некоторых пользователей отсутствие этой простейшей функции отмены является достаточным основанием для полного отказа от программного продукта.

Пользователь обычно сразу замечает свои ошибки, поскольку результат выглядит не так, как ожидалось. Тогда он останавливается для того, чтобы оценить ситуацию. Если представление данных достаточно ясное, он видит ошибку, вызывает функцию отмены, чтобы все вернулось на свои места, и продолжает работу.

Множественная отмена может быть выполнена несколько раз подряд. Она отменяет последовательность операций в порядке, обратном их выполнению. Любая программа с простой функцией отмены должна запоминать последнюю операцию пользователя и, если необходимо, кэшировать изменившиеся данные. Если в программе реализована множественная отмена, ей приходится поддерживать стек операций, глубину которого опытный пользователь может задать в настройках приложения. При каждом вызове функции отмены выполняется модифицирующая отмена: отменяется самая последняя операция, данные восстанавливаются должным образом, и отмененная операция удаляется из стека.

Ограничения одиночной отмены

Самое значительное ограничение одноуровневой функциональной отмены проявляется тогда, когда пользователь устраивает «короткое замыкание», пытаясь исправить положение. Как правило, это происходит, если пользователь заметил свою ошибку не сразу. Например, он удаляет шесть абзацев текста, затем одно слово и после этого решает, что шесть абзацев были удалены по ошибке, и их необходимо вернуть на место. К сожалению, выполнение операции отмены вернет ему только одно слово, а шесть абзацев будут утрачены навсегда. Функция отмены не выполнила задачу, потому что следовала буквальной логике, а не соображениям практичности. Любому ясно, что шесть абзацев важнее одного слова, но программа легко отказывается от них ради этого слова. Слепота программы заставляет ее сохранить монету в 25 центов и выбросить пятидесятидолларовую банкноту только потому, что монетка была предложена последней.

В некоторых программах щелчок мышью, даже самый невинный, заставляет функцию одиночной отмены забыть последнее осмысленное действие пользователя. Хотя множественная отмена решает эти проблемы, она имеет свои недостатки.

Ограничения множественной отмены

Реакцией на минусы одиночной отмены было создание многоуровневой реализации той же самой модифицирующей отмены. Программа сохраняет каждое действие пользователя. При многократном обращении к функции отмены действия отменяются в порядке, обратном порядку их выполнения. В предыдущем сценарии с шестью абзацами текста, имея подобную функцию, пользователь может восстановить удаленное слово при первом вызове отмены и шесть удаленных абзацев при втором. Необходимость повторного удаления одного слова является небольшой платой за возможность восстановить шесть ценных абзацев. Интерфейсный налог в виде необходимости заново удалить одно слово практически незаметен и не принимается во внимание, как не принимается во внимание стоимость вызова машины скорой помощи: кто будет подсчитывать такие мелочи, когда речь идет о человече-

ской жизни? Однако факт остается фактом: механизм отмены построен на ошибочной модели, и при иных обстоятельствах отмена в строгом порядке «последним вошел, первым вышел» может привести к тому, что лечение принесет столько же страданий, сколько сама болезнь.

Вернемся к нашему пользователю, удалившему шесть абзацев. Предположим, что затем он открыл другой документ и выполнил в нем глобальный поиск с заменой. Чтобы вернуть на место недостающие шесть абзацев, пользователь должен будет вначале отменить довольно сложную операцию глобального поиска с заменой. На этот раз «вмешавшаяся» операция не столь незначительна, как удаление одного слова. Она потребовала сил и времени, и ее отмена, очевидно, является неприятной дополнительной нагрузкой. Было бы очень удобно, если бы пользователь мог выбирать, какую операцию из очереди следует отменить. Это позволило бы оставить нетронутыми промежуточные безошибочные операции.

Проблемы модели множественной отмены

Проблемы, свойственные множественной отмене, кроются не в ее поведении, а в ее модели представления. Большинство механизмов отмены спроектированы в жестокой функциональной манере. Они помнят, какие действия пользователь выполнял шаг за шагом, и разбивают их на отдельные функции. В лучших традициях модели представления, основанной на модели реализации, системы отмены пытаются отразить логику кода и заложенные в продукт структуры данных вместо того, чтобы следовать целям пользователей. Каждый щелчок по кнопке `Отмена` обращает действие размером ровно в одну функцию. Отмена, реализованная по этому принципу, является подходящей ментальной моделью для устранения большинства простых проблем, связанных с ошибками пользователя при вводе. Пользователи замечают ошибки сразу и сразу же их исправляют – как правило, не позже, чем через два-три действия. Но когда проблема становится более сложной, модель модифицирующей множественной отмены перестает хорошо масштабироваться.

Недостатки стека

Когда пользователь оказывается в логическом тупике (а не просто ошибается при вводе данных), он нередко делает несколько шагов в неизвестность, прежде чем осознает, что заблудился и ему необходима команда спасателей. К этому моменту он, возможно, уже выполнил целый ряд тесно связанных функций, из которых лишь часть требует отмены. Он, скорее всего, захочет отменить действия выборочно, не обязательно в строго обратном порядке. Что, если пользователь введет какой-то текст, отредактирует его, в затем решит отменить ввод части текста, но не действия по редактированию? Такую операцию нелегко объяснить и реализовать. Нил Рубенкинг (Neil Rubenking) привел вот такой фатальный пример: предположим, пользователь выполнил в тексте глобальную замену слова «шницель» на слово «котлета», а затем

глобально же заменил «кот» на «собака». Если попросить программу отменить первое действие без отмены второго, сможет ли она корректно обработать вхождения слова «собакалета»?[1]

В этой запутанной ситуации упрощенное представление функции отмены в виде одного линейного стека не столь уместно, как в более тривиальных случаях. Возможно, пользователь захочет изучить список своих действий и выбрать те, которые подлежат отмене (не обязательно идущие подряд), оставив остальные нетронутыми. Такой подход потребует реализации поясняющей отмены с более ясным представлением действий пользователя, нежели принято при реализации множественной отмены или отмены вслепую. Кроме того, в таком представлении должны быть в достаточной степени развиты средства выбора элементов. Еще более трудной задачей является представление операции в списке таким образом, чтобы пользователь ясно понимал, что именно он отменяет.

Повтор

Функция **повтора** возникла как следствие использования для функции отмены модели реализации, согласно которой операции отменяются в порядке, обратном их выполнению, и ни одна операция не может быть отменена без предварительной отмены всех корректных операций, выполненных после нее. В сущности, функция повтора отменяет отмену, и ее легко реализовать, если разработчик уже потратил время на создание функции отмены.

Функция повтора позволяет смягчить жестокость многократной отмены. Когда пользователь хочет откатиться на пять операций, он принимается щелкать по кнопке `Отмена`, ожидая момента, когда все наконец вернется к требуемому состоянию. В такой ситуации легко щелкнуть по кнопке лишний раз. Конечно, пользователь заметит, что отменил нужное действие. Функция повтора решает проблему, позволяя ему вернуть это действие.

Многие программы, в которых реализована одиночная отмена, воспринимают последнюю отмену как действие, которое само может быть отменено. В результате второе обращение к функции отмены эквивалентно вызову функции повтора.

Групповая множественная отмена

В редакторе Microsoft Word функция отмены реализована не совсем удачно, но такая реализация, к сожалению, получает все более широкое распространение. Речь идет о варианте множественной отмены, который мы назовем **групповой множественной отменой**. Это многоуровневая отмена, содержащая текстовое описание каждой операции

[1] В оригинальном тексте используются замены *tragedy* на *catastrophe*, *cat* на *dog* и слово *dogastrophe* как результат «мутации». – *Примеч. перев.*

в стеке. Вы можете изучить список последних операций и выбрать ту, которая подлежит отмене. Однако при этом вы отмените не только ее, но и все операции вплоть до этой точки включительно (рис. 16.1). Такой стиль множественной отмены применяется и в ряде продуктов компании Adobe.

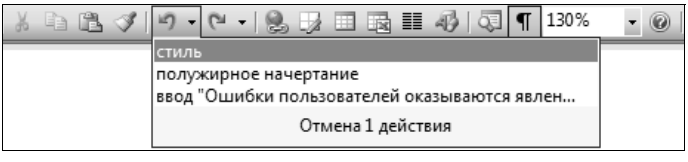


Рис. 16.1. *Функция Отменить/Повторить в Microsoft Word позволяет отменить несколько действий, но только группой. Вы не можете просто отменить операцию, выполненную за три операции до последней; то же самое верно и для функции повтора*

В результате вы не можете восстановить шесть удаленных абзацев без предварительной отмены всех последующих операций. После того как вы выберете одну или несколько операций для отмены, список, перечисляющий отмененные операции в обратном порядке, становится доступным в функции повтора. Повтор работает аналогично отмене: вы можете выбрать для повтора любое количество операций, и все операции вплоть до указанной будут повторены.

Программа отображает две визуальных подсказки о таком поведении. Если пользователь выберет в списке, скажем, пятый элемент с конца, то выделенными окажутся этот элемент и четыре последующих. Кроме того, появляется надпись «Отменить 5 действий». Тот факт, что разработчикам пришлось добавить текстовую надпись, свидетельствует о расхождении ментальных моделей у пользователей и у программистов. Пользователи полагали, что могут выбрать из списка любое из прошлых действий и отменить его независимо. Программа не предоставляла такой возможности – и тогда появились предупреждающие знаки. Это напоминает дверь с ручкой, провоцирующей входящего тянуть на себя, и надписью «От себя». Множественная отмена, несомненно, очень полезный механизм, но нет причин останавливаться на принятой реализации. Следует бросить ресурсы на реализацию такого варианта отмены, который позволяет пользователям отменять нежелательные действия выборочно, а не только в связке с действиями, выполненными после них.

Прочие модели для механизмов, схожих с отменой

Модель представления функции отмены в ее простейшем виде (одиночная отмена) соответствует ментальной модели пользователя: «Я только

что выполнил действие, которое теперь хочу отменить. С этой целью я щелкну по соответствующей кнопке». К сожалению, эта модель представления быстро расходится с ментальной моделью пользователя, как только ситуация становится сложнее. В этом разделе мы обсудим модели поведения, аналогичные поведению функции отмены, но работающие иначе, чем стандартные идиомы отмены и повтора.

Сравнение: как это будет выглядеть?

Помимо того, что спаренная функция Отмена/Повтор оказывает надежную поддержку сомневающимся в себе пользователям, она является удобным инструментом сравнения. Предположим, что вы хотите сравнить вид текста с рваным правым краем и выровненным правым краем. Открыв документ, вы выполняете выравнивание по правому краю. Потом вы возвращаетесь к прежнему виду с помощью функции отмены, а затем выполняете функцию повтора, чтобы снова увидеть выравнивание. Получается, что переключение между отменой и повтором реализует функцию **сравнения — «а что, если?»**. Просто так получилось, что эта функция представлена моделью реализации. Если бы возникла необходимость добавить такую функцию исходя из пользовательской ментальной модели, ее можно было бы представить в виде самостоятельного инструмента сравнения. Эта функция позволила бы пользователю неоднократно совершать то шаг вперед, то шаг назад и сравнивать состояния визуально.

Некоторые пульты дистанционного управления к телевизорам Sony имеют кнопку, переключающую каналы с текущего на предыдущий. Это очень удобно, когда вы смотрите две передачи одновременно. Функция перехода на предыдущий канал с помощью одной команды выполняет то же самое, что пара Отмена/Повтор, вдвое снижая уровень интерфейсного налога при той же функциональности.

Когда пара Отмена/Повтор используется для сравнения, она является одной функцией, а не двумя. Одна ее половина говорит: «Применить данное изменение», а другая: «Отменить данное изменение». Кнопка Сравнить могла бы более точно представить это действие пользователю. Хотя мы обсуждали этот инструмент на примере текстового редактора, функция сравнения была бы очень полезна в графическом редакторе или программе для рисования, где пользователь последовательно трансформирует изображения. Возможность легко и быстро сравнивать трансформированное изображение с исходным вариантом станет значительным подспорьем для компьютерного художника. Во многих продуктах подобная функциональность реализуется посредством миниатюр изображений.

Несомненно, функция сравнения не входит в число базовых. Кнопку переключения на предыдущий канал, скорее всего, использует меньшинство телезрителей. Так и функция сравнения была бы полезна только тем, кто пользуется приложением часто и помногу. Однако это

не умаляет ее достоинств, поскольку люди обычно помногу работают с графическими редакторами. Для подобных программ забота об опытном пользователе представляется разумной.

Специфическая отмена

Клавиша Backspace фактически выполняет функцию отмены, причем специализированную. Когда пользователь допускает опечатку, клавиша Backspace позволяет стереть ошибочный ввод. Когда пользователь допускает опечатку, а затем выполняет какую-нибудь функцию, не связанную с вводом, например форматирование абзаца, после чего несколько раз нажимает на клавишу Backspace, ошибочные символы стираются, а операция форматирования остается в силе. В зависимости от точки зрения такую возможность можно считать либо гибкой, так как она позволяет пользователю выполнять отмену специальным образом, либо потенциально опасной, поскольку пользователь может передвинуть курсор и нечаянно стереть не те символы, которые были введены последними.

Логика подсказывает, что второй случай представляет собой определенную проблему. Однако наблюдения показывают, что эта проблема возникает у пользователей редко. Такая изолированная модифицирующая отмена (невозможно подобрать термин попроще) естественна и проста в использовании, поскольку все происходит на глазах: пользователь видит, что именно будет удалено. Клавиша Backspace является классическим примером модифицирующей отмены. Она восстанавливает данные выборочно, игнорируя промежуточные операции. И все же, если вы представите себе механизм отмены с неким указателем, который можно передвигать, выбирая отменяемое действие, вам, вероятно, покажется, что такая схема явно неуправляема и вызовет непонимание у типичного пользователя. Опыт, однако, говорит нам, что при использовании клавиши Backspace ничего подобного не происходит. Она все делает как надо, потому что ее поведение соответствует ментальной модели курсора, которой придерживаются пользователи. Поскольку курсор является источником новых символов, вполне разумно, если он будет и центром удаления символов.

Обладая этим знанием, мы можем создать различные типы модифицирующей отмены, например функцию отмены форматирования, которая отменяла бы лишь команды форматирования, и другие действия **специфической отмены**. Если пользователь набрал некоторый текст, преобразовал начертание букв в курсивное, набрал дополнительный текст, увеличил отступ абзаца, набрал еще текст, а затем вызвал функцию отмены форматирования, то будет отменено только увеличение абзацного отступа. Следующий вызов функции отмены форматирования «снимет» курсив. Однако ни один из вызовов этой функции не повлияет на содержание набранного текста.

Что подразумевается под специфической отменой в программе, не работающей с текстом? В графическом редакторе, например, могут быть отдельные команды отмены для инструментов закраски, для трансформаций и для инструментов копирования и вставки. Ничто не мешает нам реализовать независимые функции отмены для каждого класса операций в приложении.

Инструментами закраски мы здесь называем все инструменты рисования (карандаши, перья, заливки, аэрозоли, кисти) и все инструменты для создания графических примитивов (прямоугольники, линии, эллипсы, стрелки). Трансформации – это все, что связано с манипулированием изображением: сдвиг, поворот, резкость, оттенок, контрастность, толщина линии. Инструменты копирования и вставки включают в себя лассо, прямоугольное и овальное выделение, клонирование, перетаскивание и прочие средства изменения положения объекта. В отличие от клавиши Backspace, примененной в отношении текста, отмена функции закраски в графическом редакторе будет выполняться по временно́му признаку независимо от того, какой объект выделен. Иными словами, в первую очередь будет отменена последняя функция закраски независимо от текущего выделения. В тексте естественным порядком следования данных является направление от левой верхней точки к правой нижней. Удаление данных в противоположном направлении является сильной и естественной ментальной моделью. При рисовании подобное соглашение отсутствует, так что любой порядок удаления, кроме порядка, обратного вводу, только запутает пользователя.

Возможно, более удачным подходом стала бы отмена только в пределах выделенной области. Пользователь выделяет графический объект и запрашивает, например, отмену трансформации. Последняя трансформация, примененная к *выделенному объекту*, будет отменена.

Большинство пользователей привыкли к модифицирующей отмене и сочтут специфическую отмену новаторской и, возможно, неудобной. Но повсеместное присутствие клавиши Backspace показывает, что модифицирующая отмена является заученным поведением, которое пользователи находят полезным. Если бы инструменты специфической отмены присутствовали в большом количестве программ, пользователи быстро адаптировались бы к ним. Со временем они начали бы удивляться отсутствию подобных инструментов в приложениях, как удивились бы, не обнаружив в текстовом редакторе поддержки клавиши Backspace.

Буферы удаленных данных

Когда пользователь работает над документом в течение продолжительного времени, у него может появиться желание иметь хранилище для удаленного текста. Возьмем, например, наши шесть пропавших абзацев. Если они отделены от нас парой сложных команд поиска и замены, то восстановить их с помощью функции отмены не легче, чем ввести заново. Пользователь думает: «Если бы программа просто запоми-

нала, что я удалил, и хранила это в специальном месте, я мог бы пойти туда и взять то, что мне нужно».

Пользователь при этом представляет себе некое хранилище информационных составляющих своих действий, а не просто стек процедур, то есть **буфер удаленных данных**. Пользователю нужен только пропавший текст – без связи с функцией, которая его удалила. Обычно модель представления заставляет пользователя не только отдавать себе отчет в каждом промежуточном действии, но и отменять их по очереди. Чтобы реализовать функциональность, в большей степени ориентированную на потребности пользователя, нам следует дополнить обычный стек отмены независимым буфером, хранящим все удаленные данные или текст. У пользователя должна быть возможность в любой момент открыть этот буфер в виде документа и восстановить нужный текст с помощью стандартных идиом копирования и вставки или перетаскивания. Если записи в этом буфере будут промаркированы именами документов и временем, навигация в нем окажется простой и наглядной.

Пользователь мог бы просматривать буфер удаленных данных в произвольном порядке, а не строго последовательно. Поиск недостающих шести абзацев был бы простой визуальной процедурой, не зависящей от количества и сложности шагов, предпринятых после удаления. Буфер удаленных данных следует предложить пользователю наряду с обычной функцией модифицирующей множественной отмены, поскольку он дополняет ее. Необходимо сохранять данные в этом буфере в любом случае. Подобная возможность была бы весьма полезна в большинстве программ, будь то приложение для электронных таблиц, программа рисования или генератор счетов-фактур.

Версии и откат

Иногда у пользователей возникает желание вернуться на достаточно большое расстояние, но при этом мелкие промежуточные действия не являются значимыми. Потребность в модифицирующей отмене остается, но в большинстве случаев нет необходимости различать отдельные составляющие действий для операций более давних, чем несколько последних. **Создание версий**, обсуждаемое более подробно в главе 17, сводится к созданию копии всего документа – подобно моментальному слепку реальности, создаваемому фотоаппаратом. Поскольку создание версий затрагивает весь документ в целом, оно обычно реализуется через прямое обращение к файловой системе. Самая существенная разница между созданием версий и прочими механизмами отмены заключается в том, что пользователь должен явным образом запросить создание версии – «моментального снимка» или копии документа. После этого он может спокойно продолжать работать с оригиналом. Если он впоследствии решит, что внесенные изменения нежелательны, то сможет вернуться к сохраненной копии – предыдущей версии документа.

Существует множество инструментов для программистов, поддерживающих концепцию промежуточных версий, но за пределами мира разработчиков она до недавнего времени не была известна. Продукт Writeboard (37signals) автоматически создает версии документа, над которым ведется совместная работа, позволяя пользователям сравнивать версии и, разумеется, возвращаться к любой из существующих (рис. 16.2).

Крайне важным моментом в эффективности системы версий является поведение команды «откат». Наряду с перечнем имеющихся версий документа, о котором идет речь, эта команда должна предоставлять дополнительную информацию о каждой из них, например время и дату создания, имя создавшего ее пользователя, размер документа и, возможно, примечания автора. У пользователя должна быть возможность оценить различия между версиями и в конечном итоге выбрать любую из версий для отката к ней. При этом текущее состояние документа

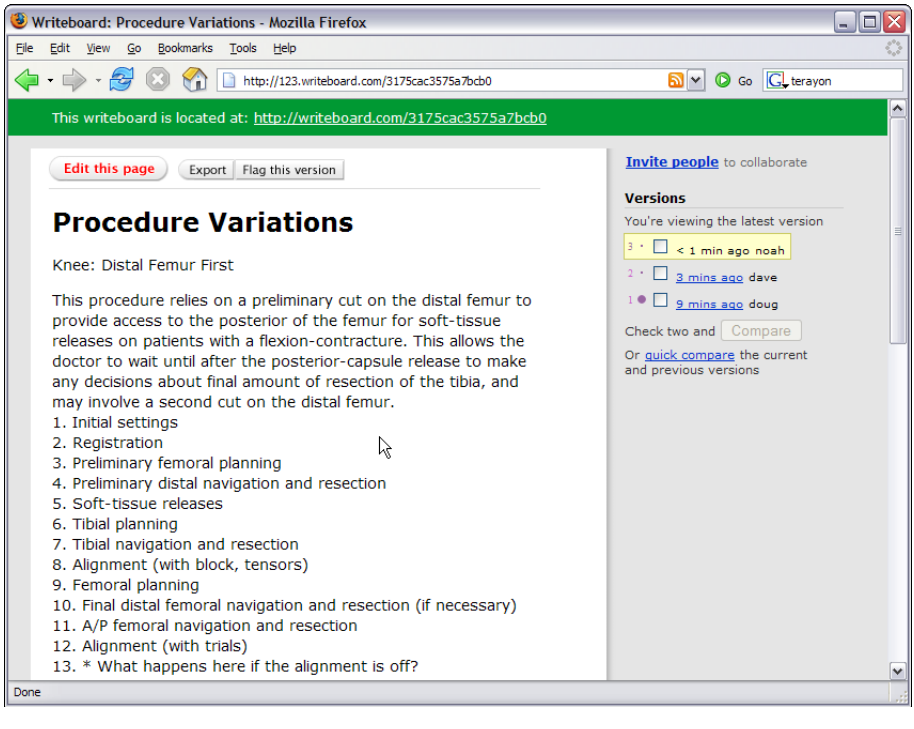


Рис. 16.2. *Продукт Writeboard компании 37signals позволяет нескольким людям работать над одним документом. Writeboard создает новую версию всякий раз, когда пользователь сохраняет внесенные в документ изменения, и позволяет пользователям сравнивать различные версии. Это может быть весьма полезно, поскольку совместная деятельность ведется без риска потерять проделанную любым из участников работу*

должно быть сохранено в виде еще одной версии, чтобы пользователь впоследствии мог к нему вернуться.

Замораживание

Замораживание – это действие, обратное созданию версий. Оно состоит в защите данных в документе от внесения изменений. Все, что было введено, не подлежит дальнейшей модификации, но новые данные добавлять можно. Существующие абзацы неприкосновенны, но между ними можно вставлять новые.

Такой подход более уместен при работе с графикой, чем с текстом. Он напоминает процесс напыления фиксажа: то, что было нарисовано до этого, останется неизменным, но рисовать поверх фиксажа можно. Изображения, размещенные на экране, блокируются и не подлежат редактированию. В то же время новые изображения могут быть свободно наложены на старые. Приложение Corel Painter реализует такую возможность в командах Wet Paint и Dry Paint.

Необратимые действия

Некоторые действия попросту невозможно отменить, поскольку они задействуют устройства, не находящиеся под прямым контролем программы. Например, после того как сообщение электронной почты отправлено, его нельзя вернуть. Однако многие операции лишь маскируются под необратимые, тогда как в действительности их нетрудно отменить. Например, при первом сохранении документа большинство программ позволяет пользователю выбрать имя файла. Но практически ни одна программа не дает возможность переименовать этот файл. Конечно, вы можете выполнить команду Сохранить как и указать другое имя, но это будет *другой* файл с новым именем, а старый файл с прежним именем никуда не денется. Почему операция указания имени файла не может быть отменена? Поскольку она не вписывается в традиционные представления о функции отмены, программисты не реализуют функцию, отменяющую действие именования файла.

Существуют также ситуации, когда невозможность отмены действия объясняется бизнес-правилами или политикой организации. Примерами могут служить финансовые транзакции или записи в истории болезни. В таких случаях вполне может оказаться, что функция отмены как таковая действительно не подходит, но вы можете содействовать достижению целей пользователей и обеспечить поддержку их ментальных моделей, предоставив способ коррекции или обращения операции с возможностью внешнего аудита действий пользователя.

Не пожалейте время на изучение своего приложения и попытайтесь найти обратимые действия, для которых не реализована отмена. Возможно, вы будете удивлены их количеством.

17

Новый взгляд на файлы и операцию сохранения

В мире цифровых технологий мышление, основанное на моделях реализации, сильнее всего проявляет свою мерзкую сущность в области управления файлами и в понятии «сохранение». Если вы когда-нибудь пытались научить свою маму обращаться с компьютером, то знаете, что слово *трудно* является недостаточно выразительным для того, чтобы передать всю серьезность проблемы. Начало не сулит ничего плохого: «Запусти текстовый редактор и набери письмо». Это вполне понятное для нее действие – то же самое, что писать на бумаге. Но, закончив писать, вы щелкаете по кнопке Закрыть – и появляется диалоговое окно с вопросом: «Сохранить изменения?» Вы оба раздосадованы. Она смотрит на вас и спрашивает: «Что это означает? Все ли в порядке?»

Эта проблема возникает по вине программного обеспечения, которое заставляет людей думать подобно компьютерам, безо всякой необходимости вынуждая их бороться с внутренними механизмами хранения данных. Это является проблемой не только для вашей мамы – даже опытные пользователи могут путаться и совершать ошибки. Люди тратят тысячи долларов на устройства и программы, а в результате вынуждены отвечать на вопросы вроде этого: «Вы действительно желаете сохранить документ, над которым трудились полдня?»,– и постоянно помнить о команде Сохранить как..., когда требуется всего лишь поработать с копией документа.

По опыту авторов, люди считают файловые системы компьютеров – механизмы хранения файлов приложений и данных на жестких дисках – очень сложными для понимания и применения. Это один из самых важных механизмов в компьютерах, и ошибки в работе с ним имеют малоприятные последствия. Большинство людей не понимают, в чем разница между оперативной памятью и дисковой памятью, но, к сожалению, традиционный способ проектирования программных

продуктов заставляет пользователей – и даже вашу маму – осваивать эту разницу и думать о своих документах исходя из того, как устроен компьютер.

Популяризация веб-приложений и других программ на основе баз данных создала отличную возможность для того, чтобы отказаться от необходимости думать в терминах модели реализации файловой системы компьютера. К сожалению, здесь возникла другая проблема – всякий раз когда пользователь изменяет документ или настройки, он обычно должен щелкнуть по кнопке Отправить или Сохранить изменения, и это снова вынуждает его думать о том, как работает система, – в данном случае в терминах клиент-серверной архитектуры.

В этой главе описан иной способ взаимодействия, относящегося к файлам и сохранению, – способ, лучше гармонирующий с ментальными моделями пользователей ваших продуктов. К счастью, не мы одни думаем в этом направлении.

Что не так с сохранением файлов?

Каждая запущенная программа одновременно существует в двух местах: в памяти и на диске. То же самое справедливо в отношении любого открытого файла. На сегодня это просто обязательное положение вещей – существующая технология применяет различные механизмы для быстрого доступа к данным (оперативная память) и для долгосрочного хранения этих данных (диски). Однако большинство пользователей этого не осознают. Наши ментальные модели (если не говорить о программистах) иные: существует один документ, который мы непосредственно создаем и в который вносим изменения.

Когда открывается диалоговое окно «Сохранить изменения?», изображенное на рис. 17.1, пользователь подавляет приступ страха и смущения и щелкает по кнопке Да в силу привычки. Диалоговое окно, на которое пользователи всегда реагируют одинаково, избыточно. Его следует удалить из интерфейса.

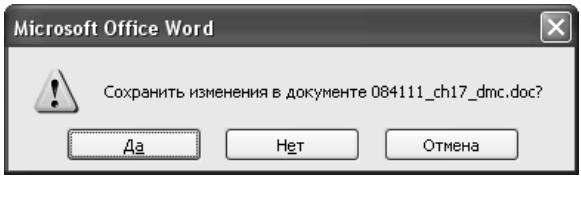


Рис. 17.1. *Вопрос, который редактор Word задает, когда вы закрываете файл с внесенными изменениями. Это диалоговое окно – результат того, что программист навязывает несчастному пользователю модель реализации файловой системы. Оно настолько не соответствует ожиданиям пользователей-новичков, что они зачастую непреднамеренно выбирают вариант «Нет»*

Диалоговое окно сохранения изменений исходит из неверного предположения о том, что сохранение изменений и отказ от сохранения – равновероятные варианты поведения пользователя. Диалоговое окно считает оба варианта одинаково важными, хотя кнопка Да нажимается на несколько порядков чаще кнопки Нет. Как говорилось в главе 10, здесь возможность перепутана с вероятностью. Пользователь может ответить *«нет»*, но почти всегда отвечает *«да»*. Ваша мама думает: «Если бы эти изменения были мне не нужны, разве я оставила бы их в документе?». Вопрос о необходимости сохранить изменения звучит для нее абсурдно.

Еще одна странность этого диалогового окна: почему оно спрашивает вас о необходимости сохранить изменения только тогда, когда вы уже закончили работу? Почему этот вопрос не был задан непосредственно после внесения изменений? Связь между закрытием документа и сохранением изменений отнюдь не естественна, хотя опытные пользователи и привыкли к ней.

Приложение открывает диалоговое окно сохранения изменений, когда пользователь закрывает файл или выходит из приложения, потому что настало время, когда программа должна согласовать копию документа, находящуюся в памяти, с копией, хранящейся на диске. Технология реализации этой функциональности связывает сохранение изменений с закрытием файла или программы, однако пользователь не видит никакой связи. Когда мы выходим из комнаты, мы не рассматриваем вариант отмены всех перестановок, которые в ней сделали. Когда мы ставим книгу на полку, мы не стираем заметки, которые сделали на полях.

Набравшись опыта, мы привыкаем использовать это окно в целях, для которых оно не предназначалось. Когда нет простого метода отменить большое количество изменений, мы пользуемся диалоговым окном сохранения изменений и отвечаем «нет». Если вы вдруг обнаруживаете, что внесли значительные изменения не в тот файл, вы вызываете это диалоговое окно в качестве запасного выхода для возврата к первоначальному состоянию. Это удобно – но все же это трюк. Существуют лучшие способы решения проблемы (например, функция возврата к определенной версии – Revert).

В чем же реальная проблема? Файловые системы в операционных системах современных персональных компьютеров (Windows XP, Windows Vista или Mac OS X) технически превосходны. Источником проблемы вашей мамы является ошибка разработчиков, которые скрупулезно перенесли прекрасную модель реализации на пользовательский интерфейс.

В реальности многим приложениям нет нужды иметь функции управления документами или файлами. Созданные компанией Apple приложения iPhoto и iTunes дают пользователя богатую и простую в применении функциональность, позволяя обычному человеку игнорировать

факт существования файлов. Список проигрывания в iTunes можно создать, изменять, раздавать, переносить на iPod, хранить годами, несмотря на то, что пользователь ни разу не сохранял его явным образом. Фотографии переносятся с фотоаппарата в iPhoto, после чего их можно сортировать, показывать, отправлять по электронной почте, печатать – и при этом пользователям никогда не приходится задумываться о файловой системе.

Проблемы модели реализации

Файловая система компьютера – это инструмент управления данными и программами на диске. Речь идет не только о крупных жестких дисках, где хранится большая часть вашей информации, но также о flash-дисках, компакт-дисках CD-R и DVD-R и сетевых дисках. Программы Finder в Mac OS и Проводник (Explorer) в Windows графически представляют файловую систему во всей ее красе.

Управление дисками и файлами не входит в список целей пользователей.

Хотя файловая система – это внутренний механизм, который не должен затрагивать пользователей, ее влияние повсеместно сказывается на пользовательском интерфейсе большинства приложений. Соскользнуть к мышлению в модели реализации очень легко, потому что в своих поисках более совершенных решений для работы с файловой системой проектировщики взаимодействия сталкиваются со сложными задачами. Особенности реализации файловой системы непосредственно влияют на нашу способность создавать полезные и осмысленные интерфейсы управления версиями документов, отношениями между документами и даже процедурной инфраструктурой наших приложений. Это влияние, вероятно, никуда не исчезнет, если мы не приложим усилия, чтобы изменить положение дел.

В настоящее время большинство программных продуктов обращаются с файловой системой точно так же, как и оболочка операционной системы (Проводник, Finder). Это все равно что заставлять автолюбителя обращаться с машиной так, как с ней обращается автомеханик. Хотя такой подход неудачен с точки зрения взаимодействия, он является стандартом де-факто, и попыткам улучшить ситуацию препятствует значительное сопротивление.

Закрытие документа и отказ от нежелательных изменений

Давнишние пользователи компьютеров привыкли думать, что закрытие документа является подходящим моментом для отказа от нежелательных изменений, внесенных по ошибке или просто для того, чтобы пова-

лять дурака. Это неверно, потому что наилучшим моментом для отказа от изменений является момент, когда они сделаны. Существует проверенная идиома, поддерживающая такой подход: функция отмены.

Сохранение документа

Во многих приложениях при первом сохранении документа или выполнении команды Сохранить как... из меню Файл вы получаете диалоговое окно Сохранение документа. Типичный пример такого окна приведен на рис. 17.2.

Это диалоговое окно выполняет две функции – функцию именования файла и функцию выбора папки для сохранения файла. Обе функции требуют от пользователя глубокого знания файловой системы и умения предсказывать, откуда файл будет удобно открывать впоследствии. Пользователь должен уметь придумывать допустимые и запоминающиеся имена файлов и разбираться в навигации по иерархии папок. Многие пользователи, разобравшиеся с именованием файлов, оставили все попытки разобраться с деревом папок. Они помещают документы на Рабочий стол или в каталог, предлагаемый приложением по умолчанию. Рано или поздно какое-нибудь неаккуратное действие

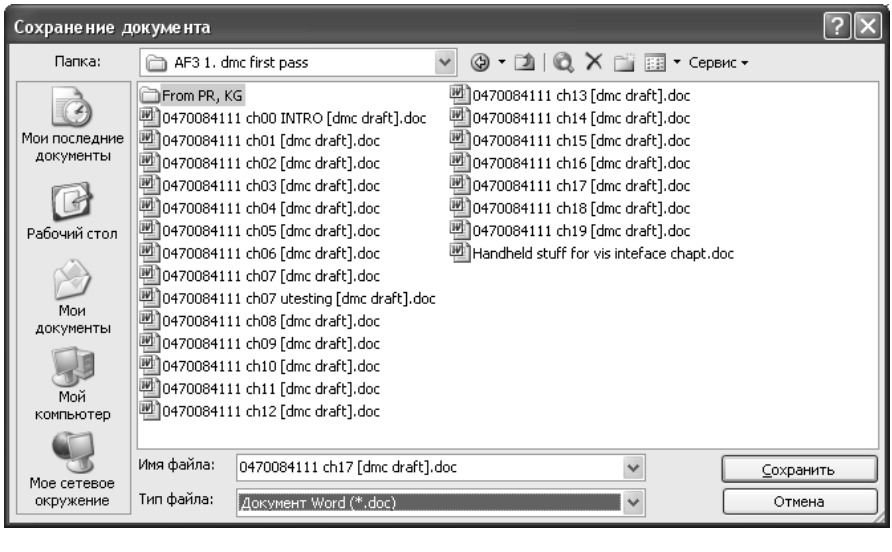


Рис. 17.2. *Диалоговое окно «Сохранение документа» выполняет две функции: оно позволяет дать файлу имя и поместить файл в папку, выбранную пользователем. Однако пользователи не понимают, что такое сохранение, и заголовок диалогового окна не соответствует их ментальной модели этой функции. Более того, если диалоговое окно позволяет вам именовать и размещать документ, вы, возможно, ожидаете, что оно позволит переименовать и переместить документ. К сожалению, наши ожидания разрушаются неудачной реализацией*

заставляет приложение забыть об этом умолчании – и пользователям приходится для поиска файлов вызывать специалиста.

Диалоговому окну «Сохранение документа» следует определиться со своим предназначением. Если смысл его существования – именование и размещение файлов, то оно плохо справляется с этой задачей. Назвав и определив файл, пользователь больше не может поменять его имя и местоположение – по крайней мере, с помощью этого диалогового окна, претендующего на функциональность, связанную с именованием и размещением файлов. Не может пользователь переименовать и переместить файл и с помощью какого-либо другого инструмента программы. В Windows XP это диалоговое окно вообще-то позволяет переименовывать файлы – но только не те, с которыми вы работаете в данный момент. Каково, а? Новичкам не повезло больше всех; опытные пользователи осваивают длинный и трудный путь: закрыть документ, открыть приложение Проводник, переименовать файл, вернуться в основное приложение, из меню Файл воззвать к диалоговому окну Открытие документа и открыть переименованный документ заново.

То, что пользователя вынуждают открыть Проводник, чтобы переименовать документ, не самое большое из зол, но здесь присутствует скрытая ловушка. Все начинается с приманки: Windows без проблем поддерживает одновременную работу нескольких приложений. Привлеченный этой возможностью, пользователь пытается переименовать файл в Проводнике, не закрыв документ в приложении. Этот вполне разумный поступок приводит в действие капкан – и стальные челюсти безжалостно смыкаются на ноге пользователя. Он получает грубый отпор в виде сообщения об ошибке, изображенного на рис. 17.3. Попытка переименовать открытый файл является нарушением правил совместного доступа, и операционная система отвергает ее, выдавая высокомерное сообщение об ошибке.

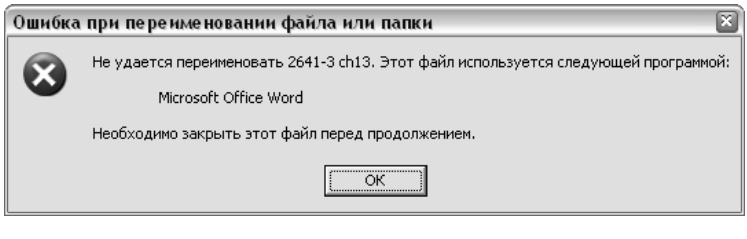


Рис. 17.3. *Если пользователь попытается с помощью Проводника переименовать файл, открытый в редакторе Word, выяснится, что Проводник слишком глуп, чтобы справиться с проблемой. Кроме того, он настолько груб, что выдает высокомерное сообщение об ошибке. Новичок, получивший резкий отпор и от редактора, и от операционной системы, может вообразить, что документ вообще невозможно переименовать*

Невинный пользователь всего лишь пытается переименовать свой документ, но обнаруживает, что заблудился в лабиринте тайн операционной системы. По иронии судьбы единственная сущность, одновременно отвечающая за этот открытый документ и обладающая достаточными полномочиями для его переименования, – то есть само приложение – отказывается даже от попытки сделать это.

Архивирование

Не существует никакой явной функции для создания копии, или архивирования, документа. Пользователь должен делать это с помощью диалогового окна «Сохранение документа», но в этой операции много непонятного. Если пользователь уже сохранил файл под именем «Альфа», то должен явным образом вызвать диалоговое окно «Сохранение документа» и поменять название. Файл «Альфа» закрывается и сохраняется на диске, а «Альфа (новый)» остается открытым и доступным для редактирования. Это действие не имеет смысла с точки зрения единственности реального документа и готовит пользователю весьма неприятную ловушку.

Вот вполне вероятный сценарий, который приводит к неприятностям. Предположим, пользователь работал с документом «Альфа» последние двадцать минут и теперь желает создать архивную копию на диске перед внесением значительных экспериментальных изменений в оригинал. Он вызывает диалоговое окно «Сохранение документа» и меняет имя на «Альфа (новый)». Программа помещает документ «Альфа» на диск, позволяя редактировать документ «Альфа (новый)». Однако пользователь за время работы ни разу не сохранял документ «Альфа», так что документ записан на диск без изменений, внесенных в течение последних двадцати минут! Эти изменения существуют лишь в документе «Альфа (новый)», который сейчас находится в памяти компьютера и с которым работает приложение. Приступив к его модификации в полной уверенности, что результат предыдущей работы хранится в файле «Альфа», пользователь подвергнет изменениям единственную копию этой информации.

Каждый знает, что можно «ввернуть» шуруп в доску при помощи молотка и забить гвоздь плоскогубцами, но опытный мастер понимает, что использование неподходящего инструмента в конце концов «выйдет боком». Либо сломается инструмент, либо будет безнадежно испорчена работа. Диалоговое окно «Сохранение документа» является неподходящим инструментом для создания копий и управления ими – и пострадает в конечном счете именно пользователь.

Модель реализации против ментальной модели

Модель реализации файловой системы входит в противоречие с ментальной моделью, которой придерживаются почти все пользователи.

Большинство пользователей представляют себе файлы похожими на реальные бумажные документы и наделяют их поведением реальных объектов. Говоря простыми словами, пользователи полагают, что в отношении всех документов справедливы два факта: во-первых, документ всегда один, а во-вторых, он принадлежит пользователю. Модель реализации файловой системы нарушает оба этих правила. Когда документ открыт, существуют две копии, и обе принадлежат открывшему приложению.

Как уже говорилось, каждый файл с данными, каждый документ и каждая программа во время работы с ними на компьютере присутствуют в двух местах одновременно – на диске и в оперативной памяти. В то же время пользователь представляет себе документ в виде книги или блокнота на полке. Время от времени блокнот берется с полки, и в него добавляются новые записи. Существует только один экземпляр блокнота: он находится либо на полке, либо в руках пользователя. При работе на компьютере диск служит полкой, а память, где происходит редактирование, эквивалентна рукам пользователя. Но в мире компьютеров блокнот не исчезает с полки – создается его копия, и именно она находится в оперативной памяти. Когда пользователь вносит изменения в документ, он на самом деле изменяет копию в оперативной памяти, в то время как оригинал хранится на диске нетронутым. Когда пользователь заканчивает работу и закрывает документ, программа оказывается перед выбором: либо заменить оригинал измененной копией, либо отбросить копию со всеми изменениями. С позиции программиста, рассматривающего все возможные варианты, события могут развиваться по любому из двух сценариев. С точки зрения модели реализации оба варианта равноправны. Но с точки зрения пользователя вообще не требуется никакого решения: он внес изменения и теперь откладывает документ в сторону. В реальном мире он взял бы блокнот с полки, сделал бы какие-то записи и положил его обратно на полку. Представьте себе полку, которая вдруг стала интересоваться, хочет ли он сохранить добавленные записи!

Прощаемся с моделью реализации

В этот момент серьезные читатели-программисты, вероятно, заерзали на своих местах. Они думаю, что мы замахнулись на святое. Непорочная копия на диске чудесна, и нам лучше воздержаться от призывов избавиться от нее! Спокойно: с реализациями наших файловых систем все в порядке (хотя мы с нетерпением ожидаем прихода более совершенного индексирования в систему Windows). Мы всего лишь должны скрыть от пользователя факт их существования. Мы по-прежнему можем предлагать ему все преимущества этой дополнительной копии, не разрушая при этом его ментальную модель.

Если мы будем визуализировать файловую структуру в соответствии с ментальной моделью пользователей, мы добьемся многого. Во-пер-

вых, работа пользователей станет более продуктивной. Если им не придется тратить силы и умственную энергию на управление файловой системой своих компьютеров, они смогут лучше сосредоточиться на своих задачах. И, разумеется, им не придется повторно выполнять многочасовую работу, потерянную из-за ошибки в сложной шахматной партии, связанной с управлением версиями файлов в современной операционной системе.

Во-вторых, мы научим наших мам действительно хорошо обращаться с компьютерами. Нам больше не придется отвечать на недоуменные вопросы о необъяснимом поведении интерфейса. Мы просто покажем программу и расскажем, как она позволяет работать с документом. И сообщим маме, что по завершении работы она сможет сохранить документ на диске точно так же, как если бы положила блокнот на полку. Наше осмысленное объяснение не будет прервано диалоговым окном сохранения изменений. Между прочим, мамы – типичные представительницы рынка массовой компьютерной продукции. Они владеют компьютерами и используют их, но не любят их, не доверяют им и используют их неэффективно.

Еще одним достоинством такого подхода будет то, что проектировщикам взаимодействия больше не придется учитывать в своих продуктах топорные особенности файловой системы. Мы сможем структурировать команды приложений в соответствии с целями пользователей, а не в соответствии с потребностями операционной системы. Отпадет необходимость называть самый левый пункт в строке меню словом «Файл». Это название – постоянно торчащие из-за фасада приложений уши технологии. Далее в этой главе мы обсудим некоторые альтернативы.

Изменение названия и содержания меню Файл противоречит давно установленному, хотя и неофициальному стандарту. Но выгода от этого изменения перевесит возможные отрицательные последствия. Опытным пользователям определенно придется приложить некоторые усилия, чтобы привыкнуть к новым идиомам, но эти усилия будут гораздо меньше, чем можно предположить. Ведь эти люди уже продемонстрировали свои способности к обучению и свою терпимость, сумев понять и освоить модель реализации. Им не составит труда изучить более удачную модель, и продуктивность их труда не упадет. Новички же немедленно почувствуют значительные преимущества этой модели. Мы, компьютерные специалисты, уже забыли, как высока гора, на которую мы взошли, но новички, которые только подходят к подножию Эвереста компьютерной грамотности, оказываются обескуражены. Если мы хоть что-то сделаем для уменьшения той высоты, на которую они вынуждены взбираться, это совершенно изменит положение вещей, и такой шаг позволит им в дальнейшем покорять гораздо более опасные вершины.

Проектирование с унифицированной файловой моделью

Правильно спроектированный программный продукт должен всегда обращаться с документом как с единственным экземпляром, а не как с двумя копиями, одна из которых хранится на диске, а вторая в памяти. Такой подход мы назовем **унифицированной файловой моделью**. В этой модели пользователям никогда не приходится сталкиваться с внутренними механизмами компьютера: запись данных в память и на диск – задача файловой системы.

Стандартный комплект инструментов управления файлами в большинстве приложений включает в себя команды Открыть, Сохранить и Закрыть, а также связанные с ними диалоговые окна «Сохранение документа», «Сохранить изменения» и «Открытие документа». Как мы убедились, все вместе они создают путаницу при выполнении одних задач и абсолютно непригодны для выполнения других. Ниже мы излагаем альтернативный подход к управлению документами, более точно соответствующий ментальной модели пользователей.

Помимо представления документа как единственного объекта этот подход включает в себя реализацию ряда операций, ориентированных на цели пользователя. Каждая такая операция, если у пользователя возникает в ней необходимость, должна быть представлена в интерфейсе самостоятельной функцией.

- Автоматическое сохранение документа
- Создание копии документа
- Создание версий документа
- Именование и переименование документа
- Размещение и перемещение документа в файловой системе
- Указание формата хранения документа
- Отмена отдельных изменений
- Отказ от всех изменений

Автоматическое сохранение документа

Сохранение документа – одно из самых важных действий, которому должен научиться каждый пользователь компьютера. Вызов этой функции означает принятие всех изменений, внесенных пользователем в копию документа, находящуюся в оперативной памяти, и запись этих изменений в копию документа на диске. В унифицированной модели мы упраздняем различия между этими двумя копиями в пользовательском интерфейсе, так что функция Сохранить должна исчезнуть из основных элементов интерфейса. Сказанное *не означает*, что она должна совсем исчезнуть из программы. Это по-прежнему необходимая операция.

Сохраняйте документы и настройки автоматически.

Приложение должно автоматически сохранять документы. Начать следует с того, что, когда пользователь заканчивает работу с документом и вызывает функцию закрытия, программа должна записать все изменения на диск, не требуя от пользователя подтверждения.

В идеальном мире этого было бы достаточно; однако компьютеры и программы дают сбои, иногда пропадает питание, и разнообразные другие непредсказуемые аварийные события могут приводить к потере результатов вашего труда. Если питание отключится до того, как вы закрыли документ, все изменения, внесенные в него, будут потеряны, поскольку он будет стерт из оперативной памяти. Копия на диске сохранится, но могут потеряться результаты нескольких часов работы. Чтобы этого не случилось, приложение должно также сохранять документ через определенные интервалы времени на протяжении всего сеанса работы. В идеале программа должна сохранять малейшие изменения, как только пользователь внесет их, то есть после любого нажатия на клавишу. В большинстве программ это вполне возможно. Другой вариант – накапливать небольшие изменения в памяти и сохранять их через разумные интервалы времени.

Важно, чтобы эта функция автоматического сохранения не влияла на скорость работы пользовательского интерфейса. Сохранение следует сделать фоновой задачей либо выполнять в те моменты, когда пользователь не взаимодействует с приложением. Никто не набирает текст непрерывно. Каждый время от времени останавливается, чтобы собраться с мыслями, перевернуть страницу или сделать глоток кофе. От приложения требуется лишь дождаться момента, когда пользователь прервется на пару секунд, и сохранить файл.

Автоматическое сохранение устроит практически всех пользователей. Однако некоторые (в том числе и авторы этих строк) настолько боятся сбоев и потери данных, что имеют привычку сохранять документ после каждого абзаца, а иногда и после каждого предложения (для чего пользуются комбинацией Ctrl + S). Все программы обязаны обеспечивать возможность сохранения вручную, но не должны *требовать* этого от пользователей.

Создание копии документа

В приложении должна присутствовать явно обозначенная функция Создать копию. Копия должна быть идентична оригиналу, но никоим образом не привязана к нему. Это означает, что последующие изменения в оригинале не влияют на эту копию. Новая копия должна автоматически получать некоторое стандартное имя, например «Копия Альфа», если исходный документа назывался «Альфа». Если документ

с таким именем уже существует, новая копия должна получить имя «Копия Альфа #2». (Другой разумный вариант – «Копия Альфа» и «Вторая Копия Альфа», однако здесь есть одна тонкость: оригиналы и файлы копий в списке файлов, по умолчанию отсортированном по алфавиту, не будут расположены рядом.) Копии следует помещать в тот же каталог, что и оригинал.

Существует соблазн сопровождать команду копирования диалоговым окном, но работа пользователя не должна прерываться. Приложение должно выполнять свою работу тихо, эффективно и разумно, не досаждая пользователю вопросами вроде: «Вы уверены, что хотите сделать копию?» С точки зрения пользователя это – простая команда. Если возникнут непредвиденные обстоятельства, программа должна самостоятельно принять конструктивное решение.

Именование и переименование документа

Имя документа должно отображаться в заголовке приложения. Если пользователь решит переименовать документ, у него должна быть возможность щелкнуть по имени и отредактировать его по месту. Что может быть проще и естественнее?

Размещение и перемещение документа в файловой системе

Большинство обрабатываемых документов уже существует. Их требуется открывать, а не создавать с нуля. Это означает, что их расположение в файловой системе уже известно. Хотя мы задумываемся о папке для документа в момент его создания или первого сохранения, ни одно из этих событий не является важным вне модели реализации. Новый файл следует поместить в какое-то разумное место, где пользователь сможет впоследствии найти его, например на Рабочий стол.

Помещайте файлы туда, где пользователи смогут их найти.

Конкретное место зависит от пользователей и типа проектируемого продукта. Для сложных монопольных приложений, ежедневно используемых людьми, иногда бывает уместно определять особое место для хранения связанных с этими приложениями документов, но в случае временных приложений и редко используемых монопольных приложений не стоит прятать файлы пользователей в укромных уголках файловой системы.

Если пользователь захочет явным образом указать место документа в файловой иерархии, он может запросить эту возможность через меню. Тогда появится диалоговое окно «Перемещение документа» с выделенным именем текущего документа. В этом диалоговом окне (более

удачно названном варианте окна «Сохранение документа») у пользователя будет возможность переместить документ туда, куда он пожелает. Приложение, таким образом, автоматически распределяет файлы, а диалоговое окно служит лишь для их перемещения.

Указание формата хранения документа

В нижней части существующего диалогового окна «Сохранение документа», изображенного на рис. 17.2, присутствует дополнительный раскрывающийся список, позволяющий пользователю указать формат хранения файла. Эта возможность не должна находиться в данном месте. Привязка формата к действию сохранения файла без нужды усложняет эту операцию. В редакторе Word, если пользователь без задней мысли изменит формат, операция сохранения и все последующие попытки закрыть документ будут сопровождаться пугающим и неожиданным диалоговым окном. Переопределение физического формата файла происходит относительно редко, а сохранение файла – дело обычное. Эти две функции не следует объединять.

С точки зрения пользователя физический формат документа, будь то RTF, ASCII или формат редактора Word, является характеристикой документа, а не файла на диске. Указание формата не должно быть увязано с действием сохранения документа на диск. Оно более уместно в диалоговом окне «Свойства» (документа), доступном через пиктограмму, расположенную рядом с именем файла документа.

Интерфейс этого диалогового окна должен ясно доводить до сознания пользователя тот факт, что выполнение функции чревато потерей важных данных.

В случае с графическими программами, где возможность сохранения изображений в различных форматах весьма желательна, для этой цели подходит диалоговое окно «Экспорт» (уже присутствующее в некоторых графических редакторах).

Отмена отдельных изменений

Если пользователь непреднамеренно внесет в документ изменения, которые должны быть аннулированы, ситуацию легко исправить с помощью уже существующей функции отмены (более подробно поведение отмены мы описывали в главе 16). Не следует прибегать к услугам файловой системы для создания суррогата отмены. Файловая система может служить механизмом реализации этой функции, но отсюда никак не следует, что сама функция должна быть представлена пользователю в терминах файловой системы. Идея непосредственного обращения к файловой системе для отмены изменений противоречит смыслу функции отмены.

Функция создания версий, описанная далее в этой главе, демонстрирует, как ориентированный на файлы подход к функции отмены может быть реализован в согласии с унифицированной файловой моделью.

Отказ от всех изменений

Хотя это редкий сценарий, но нужно оставлять пользователю возможность отказаться от всех изменений, сделанных после открытия или создания документа, и нам определенно не помешает предоставить ему такую возможность. Вместо того чтобы ставить пользователя перед необходимостью изучать файловую систему для достижения этой цели, предоставьте ему в главном меню функцию отказа от всех изменений.

Еще один хороший способ представить данную операцию – предложить функцию возврата к определенной версии, действие которой основано на системе версий, описанной ниже. Поскольку функция отказа от изменений предполагает значительную потерю данных, вы должны защитить пользователя недвусмысленными предупреждениями. Желательно, чтобы сама эта функция могла быть отменена, что сравнительно легко реализовать.

Создание версий документа

Создание **версии** аналогично команде копирования. Разница заключается в том, что версия находится в ведении приложения и для пользователя выглядит как все тот же документ. Пользователям должно быть понятно, что они могут вернуться к тому состоянию, которое документ имел на момент создания версии. Пользователь должен располагать возможностью видеть список версий и сопутствующую статистику, такую как время создания версии и ее размер. Одним щелчком пользователь может выбрать версию и сделать ее активным вариантом документа. Документ, который был активным на момент выбора версии, сам становится одной из версий. Кроме того, раз дисковое пространство сегодня не в дефиците, имеет смысл создавать версии регулярно, даже если пользователям это не приходит в голову.

Новое меню Файл

Наше новое меню Файл теперь выглядит так, как показано на рис. 17.4, и вот его функции:

- Функции Создать и Открыть остались прежними.
- Функция Закрыть закрывает документ без диалоговых окон и прочей суеты, предварительно сохранив все изменения.
- Команды Переименовать и Переместить открывают диалоговое окно, позволяющее пользователю переименовать текущий файл или переместить его в другую папку.
- Команда Создать копию создает новый файл, содержащий копию текущего документа.
- Команда Печать открывает единое диалоговое окно управления печатью.

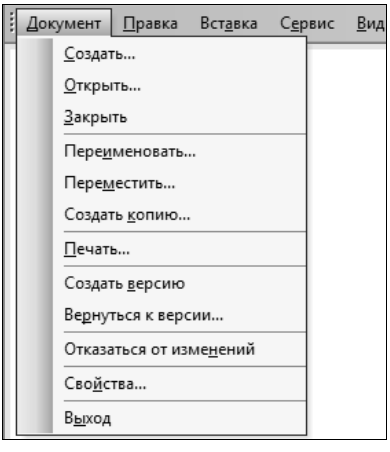


Рис. 17.4. *Пересмотренное меню Файл лучше отражает ментальную модель пользователя и не следует модели реализации, выбранной программистом. Файл существует в единственном экземпляре и принадлежит пользователю. Если пользователь захочет, он сможет создать версию или копию, переименовать файл, отменить все изменения или изменить формат файла. Ему больше не нужно разбираться в тонкостях и беспокоиться о соответствии копии в оперативной памяти и копии на диске*

- Команда Создать версию аналогична команде копирования, но приложение занимается сопровождением версий посредством диалогового окна, открывающегося по команде Вернуться к версии.
- Команда Отказаться от изменений отменяет все изменения, внесенные в документ с момента его открытия.
- Команда Свойства открывает диалоговое окно, позволяющее пользователю изменить физический формат документа.
- Команда Выход работает как обычно, закрывая документ и приложение.

Новое название для меню Файл

Теперь, когда у нас есть унифицированная модель хранения документа, заменяющая собой двойственную модель реализации (хранение на диске и в оперативной памяти), отпадает необходимость давать крайнему левому пункту главного меню приложения название Файл, отражающее модель реализации, а не пользовательскую модель. Существуют две разумных альтернативы.

Можно назвать это меню в соответствии с типом документа, с которым работает приложение. Например, в программе обработки электронных таблиц можно назвать его Таблица, а в программе, генерирующей счета, его можно назвать Счет.

Вторым вариантом является общее название Документ. Это разумный выбор для таких приложений, как текстовые процессоры, программы работы с электронными таблицами или графические редакторы. Однако он вряд ли подойдет для более специализированных нишевых приложений.

В противоположность этому те немногие программы, которые все-таки представляют содержимое дисков в виде файлов, такие как оболочки операционных систем и вспомогательные программы, должны иметь меню Файл, поскольку они обращаются с файлами как с файлами.

Индикация состояния

Если файловая система демонстрирует пользователю файл, который нельзя модифицировать из-за того, что он открыт другим приложением, она должна недвусмысленно указать на это обстоятельство. Достаточно пометки рядом с именем файла или красного цвета шрифта. Новый пользователь по-прежнему может нарваться на сообщение об ошибке (см. рис. 17.3), но у него, как минимум, будет визуальная подсказка о том, почему возникла ошибка.

В существующей модели в двух вариантах существуют не только файлы с данными, но и запущенные приложения. Когда пользователь запускает текстовый редактор через меню Пуск, на панели задач появляется кнопка приложения Word. Но если пользователь возвращается в меню Пуск, приложение Word там по-прежнему доступно! С точки зрения пользователя случилось вот что: он вытащил из ящика инструментов молоток – и обнаружил, что в ящике с инструментами по-прежнему лежит молоток.

Вероятно, такое поведение системы менять не следует: поддержка нескольких копий одного приложения – одна из сильных сторон компьютера. Но программное обеспечение должно облегчить пользователям понимание этого неинтуитивного действия. Скажем, в меню Пуск можно каким-то образом ссылаться на уже запущенное приложение.

Являются ли диски и файловые системы важным конструктивным элементом?

С точки зрения пользователя необходимость дисков ничем не обусловлена. С точки зрения инженера-электронщика есть три причины для использования дисков:

- диски дешевле твердотельной памяти;
- диски не теряют записанную на них информацию после выключения питания;
- диски являются физическим средством переноса информации с одного компьютера на другой.

Вторая и третья причины, конечно, серьезны, но указанные качества не являются исключительными характеристиками дисков: носители, созданные по другим технологиям, справляются с этими задачами не хуже. Существуют разные типы энергонезависимой памяти, которые не теряют записанную на них информацию после выключения питания. Некоторые типы памяти сохраняют данные при очень слабом электрическом питании и даже без него. Компьютерные сети и телефонные линии применяются для передачи данных на другие компьютеры – и часто с меньшими затратами, чем при использовании съемных дисков.

Первая причина, стоимость, является действительным объяснением существования дисков. Энергонезависимая твердотельная память намного дороже дисков.[1] К сожалению, у жестких дисков множество недостатков по сравнению с оперативной памятью. Они намного медленнее, чем твердотельные носители. Они еще и менее надежные, поскольку обладают движущимися частями. Они обычно потребляют больше электроэнергии и занимают больше пространства. Однако самая большая проблема дисков такова: компьютер, а точнее его центральный процессор, не может напрямую читать информацию с диска или записывать ее туда. Вспомогательные программы должны перенести информацию в оперативную память, прежде чем процессор сможет что-либо делать с этой информацией. Когда процессор закончил работу, вспомогательные программы переносят данные обратно на диск. Таким образом, обработка с использованием дисков неизбежно на несколько порядков медленнее и сложнее, чем работа с одной лишь памятью.

Диски – технологический трюк, а не конструктивный элемент.

Плата за использование дисков, проявляющаяся в виде медленной работы и повышенной сложности, настолько высока, что ничто, кроме их сравнительно ничтожной стоимости, не может заставить нас пользоваться ими. Диски не делают компьютеры лучше, мощнее, быстрее или проще в обращении. Наоборот: они делают компьютеры слабее, медленнее и сложнее. Диски – компромисс, так сказать, «ослабление» архитектуры компьютеров, основанной на твердотельной памяти. Если бы проектировщики компьютеров имели финансовую возможность использовать оперативную память вместо дисков, они бы сделали это без колебаний. Фактически они это уже делают в новейших моделях коммуникаторов и портативных компьютеров, в которых применяется память Compact Flash и другие твердотельные технологии памяти.

---

[1] Лавинообразный выход на рынок дешевых компактных ноутбуков с твердотельными дисками (так называемых нетбуков) свидетельствует о том, что эта разница перестала быть значимой. – *Примеч. ред.*

Дисковая технология наложила свой отпечаток на проектирование программного обеспечения, но это было сделано исключительно для удобства реализации, а не для улучшения обслуживания пользователей и не по какой-то другой причине, связанной с целями пользователей.

Время перемен

В пользу программного обеспечения, реализованного в соответствии с моделью файловой системы, можно привести лишь два аргумента: наши программы уже проектируются и создаются подобным образом, и пользователи к ним привыкли.

Ни один из этих аргументов не является весомым. Первый вообще несерьезен, поскольку новые приложения, использующие унифицированную файловую модель, могут свободно сосуществовать со старыми программами, придерживающимися модели реализации. Файловая система, лежащая в основе, вообще не меняется. Подобно тому, как панели инструментов быстро наводнили интерфейсы большинства программ в последние годы, унифицированная модель может быть реализована с успехом и под аплодисменты пользователей.

Второй аргумент – более коварный, поскольку те, кто его выдвигает, прячутся за спины пользователей, как за живой щит. Более того, если вы спросите самих пользователей, то они отвергнут новое решение, поскольку ненавидят перемены, особенно когда эти перемены касаются предмета, на изучение которого ушло много сил, в данном случае – файловой системы. Однако пользователи не всегда хорошо предсказывают успех того или иного решения, особенно если оно отличается от всего, что они уже испытали.

В 80-е годы компания Chrysler продемонстрировала покупателям эскизы автомобиля с принципиально новым дизайном – мини-вэна. Публика его единодушно отвергла. Будучи уверенной, что дизайн превосходен, компания не послушала пользователей и выпустила свой Caravan. И оказалась права: те же люди, которые поначалу отвергли дизайн, не только помогли этой модели стать одним из самых продаваемых мини-вэнов, но и сделали мини-вэн самым популярным автомобильным архетипом со времен появления кабриолета.

Пользователи не являются проектировщиками интерфейса, и от них нельзя ожидать понимания эффектов сдвига интерфейсной парадигмы. Однако рынок показывает, что люди с радостью отказываются от неудобного и плохо спроектированного программного продукта в пользу более простого в обращении, даже если не понимают причин, послуживших источником изменений.

18

Улучшаем ввод данных

В главе 12 мы говорили, что интерактивные продукты следует проектировать таким образом, чтобы своим поведением они походили на умных и тактичных людей. Но именно тогда, когда пользователю требуется осуществить ввод данных, интерактивные продукты проявляют минимум такта и сообразительности. Некоторые неприятные артефакты мышления, основанного на моделях реализации, не позволяют людям работать наиболее естественным для них способом. В этой главе мы обсудим проблемы, связанные с существующими способами обработки вводимых данных, и методы, позволяющие переориентировать эти процессы с нужд базы данных на нужды пользователей.

Целостность данных и информационный иммунитет

Одно из наиболее важных требований к правильно работающему программному обеспечению – чистота данных. Как гласит поговорка – «что посеешь, то и пожнешь». Как следствие, в том, что касается ввода и обработки данных, программисты обычно следуют простой логике: никогда не позволяй сомнительным данным попасть в приложение. Они воздвигают в пользовательском интерфейсе барьеры, чтобы сомнительные данные не проникли в систему и не нарушили чистое внутреннее состояние, называемое **целостностью данных**.

За требованием обеспечить целостность данных стоит представление о внешнем мире, заполненном хаотической информацией, любой фрагмент которой должен быть подвергнут фильтрации и очистке, прежде чем попадет в компьютер. Программа должна быть бдительной, подобно таможеннику на границе, постоянно выявляя плохие данные. Все данные проверяются на корректность в момент ввода. Всё поступающее извне изначально считается подозрительным, и лишь после жесткой

чистки те данные, которые прошли отбор и оказались внутри, считаются благонадежными. Преимущество такого подхода состоит в том, что после занесения информации в базу данных нет необходимости в повторных проверках допустимости и уместности этой информации.

Проблема того подхода – потребности базы данных ставятся выше интересов пользователя, который подвергается личному досмотру всякий раз, когда вводит очередную порцию данных. Заметим, что эта проблема нехарактерна для большинства персональных бизнес-приложений. Например, PowerPoint нисколько не заботится (и не имеет никакого представления) о том, правильно ли вы отформатировали свою презентацию. Но как только вы начинаете вести дела с большой корпорацией – будь вы клерк, вводящий информацию в систему управления проектом, или покупатель DVD в интернет-магазине, – вы сталкиваетесь с таможенным контролем.

Люди, заполняющие по долгу службы множество форм, знают, что данные никогда не поступают к ним в чистом виде, как того требует программа. Такая информация зачастую неполна и иногда неверна. Порой в интересах клиента приходится ускорять обработку. Но, столкнувшись с системой, жесткой в таких вопросах, клерк, ответственный за обработку информации, оказывается в тупике: он должен либо приостановить выполнение работы, либо как-то обойти систему, чтобы выполнить порученное. Если бы программа признала наличие человеческого фактора и отразила это в своем интерфейсе, все бы только выиграли.

Если оставить в стороне эффективность труда, у проблемы обнаружится еще один, более коварный аспект. Когда программа «досматривает» данные во время ввода, она недвусмысленно дает пользователю понять, что он ничтожен, а программа всесильна, что это он служит программе, а не наоборот. Очевидно, мы хотим видеть мир, в котором живут наши технологические новшества, совершенно не таким. Мы хотим, чтобы пользователь чувствовал свои возможности, знал, что компьютеры работают на благо людей. Мы должны вернуться к идеальному разделению цифрового труда: компьютер выполняет работу, а человек принимает решения.

К счастью, существует несколько способов защитить программное обеспечение от недопустимых данных. Вместо того чтобы отвергать эти данные, программист должен наделить систему *иммунитетом* к противоречиям и пробелам в информации. Это потребует написания более интеллектуального и более сложного кода, который сможет обрабатывать все отклонения, имеющиеся внутри данных, наделяя приложение своего рода **информационным иммунитетом**.

Информационный иммунитет

Чтобы реализовать иммунитет к вводимым данным, мы должны обучить приложение «смотреть под ноги» и при необходимости просить

помощи. Большинство программ выполняют арифметические действия, даже не взглянув на операнды. Программа исходит из допущения, что числовое поле содержит цифры, ибо так гласит принцип целостности данных. Если пользователь введет «девять» вместо «9», программа выдаст ошибку, тогда как человек воспримет число в такой форме, не моргнув глазом. Если бы программа просто посмотрела на данные перед выполнением действия, она бы поняла, что обычной математической функции здесь недостаточно.

Мы должны проектировать приложения, которые верят пользователю и понимают, что пользователь при необходимости все исправит и без нашего параноидального упорства. Однако приложения могут просить помощи у других приложений внутри компьютера. Существует ли модуль, способный перевести текст в число? Существует ли журнал действий, способный пролить свет на намерения пользователя?

Если никакие альтернативные варианты не проходят, приложение должно сопроводить данные комментарием. Если пользователь пожелает разобраться в проблеме, он обнаружит точное и полное описание того, что произошло и какие шаги предпринимала программа.

Да, если пользователь введет «фыва» вместо «9,38», программа не сможет выдать удовлетворительный результат. Однако немедленное прекращение обработки и поза «решите проблему прямо сейчас» – тоже не выход. Процесс ввода так же важен, как и конечный результат. Если пользовательский интерфейс спроектирован правильно, программа обеспечит визуальную обратную связь, когда пользователь вводит «фыва», так что вероятность, что он введет сотню ошибочных записей, очень мала. Вообще говоря, пользователи начинают вести себя глупо именно в тех случаях, когда программы считают их глупыми.

Когда пользователь вводит неверные данные, эти данные часто *близки* к верным. Приложения следует проектировать таким образом, чтобы они как можно эффективнее помогали пользователю исправлять ошибки. Так, если пользователь случайно набрал «TZ» в качестве двухбуквенного обозначения штата, а далее набрал «Dallas» в качестве названия города, то не надо быть семи пядей во лбу, чтобы догадаться, о каком штате идет речь («TX» – сокращение наименования штата Техас), и скорректировать ввод.

Как быть с недостающими данными?

Нехватка жизненно важных данных определенно идет вразрез с целями пользователей и назначением системы. Клерк, не указавший в счете сумму оплаты, порождает реальную проблему. Но при этом совершенно не обязательно, чтобы программа останавливала его, указывая на этот промах. Думайте о своем приложении как об автомобиле: пользователю вряд ли понравится, если рулевое колесо заблокируется из-за нехватки омывающей жидкости для лобового стекла. Приложения должны быть более гибкими. У пользователя на момент ввода может

не быть доступа к информации для всех обязательных полей, и, возможно, рабочий процесс предусматривает, что сначала вводятся имеющиеся данные, а затем, позже, по мере поступления, – недостающие. Разумеется, пользователи при этом должны быть *в курсе*, что отсутствует информация в тех полях, которые необходимо заполнить. Добиться этого можно посредством *обогащенной немодальной обратной связи* – нет ничего хорошего в полной остановке работы ради того, чтобы просто сообщить пользователю факты, которые, возможно, известны ему и без нас.

Рассмотрим для примера клерка, в чьи служебные обязанности входит заполнение справок-счетов. Он получает деньги за эту работу и провел тысячи часов, взаимодействуя с приложением. Он обладает шестым чувством по отношению к происходящему на экране и желает быть в курсе, если ему случится ввести ошибочные данные. Наиболее эффективной его работа будет в том случае, когда приложение воспользуется для уведомления об ошибках ввода пассивными визуальными и звуковыми подсказками.

Приложение может также оказывать клерку помощь: для ввода элементов данных, которые *обязательно* должны быть правильными, например номеров выпускаемых изделий, следует предлагать не просто текстовые поля для ручного ввода, а поля с автозавершением или ограничивающие элементы управления (например, раскрывающиеся списки). Для адресов и телефонных номеров естественным выбором будут интеллектуальные поля ввода, способные анализировать данные. Приложение должно обеспечивать ненавязчивую немодальную связь, помогая пользователю быть в курсе того, что происходит. Благодаря этому наш клерк сможет управлять ситуацией, и ему потребуется гораздо меньший надзор со стороны приложения.

Большинство систем обработки информации в действительности *устойчивы* к отсутствию части информации. Недостающее название, код, номер или цену почти всегда можно восстановить по остальным данным в записи. Если это не удается, данные всегда можно восстановить, обратившись к вовлеченным в транзакцию сторонам. Стоимость подобной коррекции высока, но не до такой степени, как стоимость падения продуктивности или стоимость центров технической поддержки. Наши системы обработки информации способны хорошо работать и в условиях нехватки данных. Программисты, создающие такие системы, просто не хотят делать дополнительную работу, связанную с реакцией на недостаточные данные, и потому объявляют целостность данных нерушимым законом. В результате тысячи клерков вынуждены взаимодействовать с программными продуктами, проявляющими жесткость и жестокость под предлогом поддержания целостности базы данных.

Контрпродуктивность подхода, при котором идиотами считают *всех* работников, чтобы защититься от тех немногих, кто ими является на

самом деле, очевидна. Такой подход понижает производительность каждого сотрудника, порождает значительную и дорого обходящуюся текучесть кадров, приводящую к сбоям в рабочем процессе, ухудшает моральное состояние коллектива, в результате чего повышается процент непреднамеренных ошибок у тех сотрудников, кто старается работать хорошо. Предполагать, что сотрудникам отдела обработки информации нельзя доверять, значит накликать на себя беду.

Стереотипное представление о клерке, который тупо набирает на клавиатуре данные из огромных стопок бумаги, сидя в душной комнате среди сотен таких же клерков, выполняющих такую же работу, испаряется на глазах. Ввод информации теперь становится деятельностью, все в меньшей степени рутинной и все в большей степени творческой, выполняемой умными, способными профессионалами, а с появлением интернет-коммерции – и самими клиентами. Иными словами, люди, которые занимаются вводом данных, все меньше готовы терпеть отношение к себе как к безропотным, необразованным и глупым батракам. Пользователи не станут сносить оскорбления от глупой программы, тем более если они за несколько секунд могут найти в Интернете другого производителя, который предоставит им интерфейс, относящийся к ним уважительно.

Ввод данных и сговорчивость

Если система слишком жесткая, она не способна моделировать процессы реального мира. Система, отвергающая реальность, в которой живут ее пользователи, бесполезна, даже когда все поля в форме заполнены. Что важнее: база данных или бизнес, который она поддерживает? Люди, администрирующие базы данных и создающие для них приложения ввода, зачастую молятся только центральному процессору. Это серьезный конфликт интересов, разрешить который может качественное проектирование взаимодействия.

Сговорчивость непросто встроить в компьютерную систему, поскольку для нее требуется значительно более мощный интерфейс. Клерк не может передвинуть документ в начало очереди, если не видит саму очередь, документ и место документа в очереди. Инструменты для извлечения документа из электронной стопки и помещения его на самый верх этой стопки должны быть простыми и очевидными в использовании. Сговорчивость требует наличия функции, переводящей записи в «подвешенное» состояние; аналогичные требования предъявляются также к функции отмены. Более серьезную проблему представляет собой тот факт, что сговорчивость таит в себе возможности для злоупотреблений.

Наилучшая стратегия против злоупотреблений – использование способности компьютера записывать все действия пользователя для последующего анализа, если таковой может потребоваться. Принцип прост: позволить пользователю делать все, что он пожелает, но регист-

рировать все его действия так, чтобы можно было восстановить полную картину.

Аудит и редактирование

Многие программисты полагают, что они обязаны информировать пользователя об ошибках, допущенных им при вводе данных. Конечно, программа обязана информировать *другие программы*, когда те допускают ошибки, но на пользователей это не распространяется. Клиент всегда прав, и программа должна принять сказанное пользователем независимо от того, что ей известно и что неизвестно. Это аналогично информационному иммунитету: все, что ввел пользователь, должно быть принято независимо от корректности этих данных с точки зрения программы.

Сказанное не означает, что приложение может умыть руки со словами: «Раз он отказывается от спасательного жилета, пусть тонет». Программа должна вести себя так, словно пользователь всегда прав, но отсюда не следует, что он действительно всегда прав. Люди постоянно ошибаются, и ваши пользователи не исключение. Программа может быть не виновата в ошибках пользователя, но она несет за них ответственность. Как же исправлять ошибки?

В ошибках может и не быть вашей вины, но вы несете за них ответственность.

Программа может выдавать предупреждения (если только они не прерывают работу ради дурацкого замечания), но, если пользователь захочет сделать что-то подозрительное, программе ничего не остается, кроме как принять этот факт и постараться уберечь пользователя от неприятностей. Как верный проводник, она должна следовать за хозяином в джунгли, не забыв прихватить снасти и большое количество воды.

Предупреждения должны быть очевидными и немодальными; они должны информировать пользователя о том, что он сделал, подобно спидометру, молчаливо отмечающему превышение скорости. При этом со стороны программы будет неразумно прерывать работу, как неразумно было бы спидометру перекрывать подачу топлива, когда стрелка переходит через отметку 110 километров в час. Вместо выдачи диалогового окна с сообщением об ошибке программа может окрашивать поле ввода для индикации подозрительных данных.

Когда пользователь делает что-то наверняка ошибочное по мнению приложения, лучший способ защитить пользователя (за исключением случаев, когда катастрофа неизбежна) – ненавязчиво дать ему понять, что, возможно, возникла проблема, и довериться его разуму, способному найти оптимальное решение проблемы. Если приложение само-

стоятельно рванет в бой и попытается исправить положение, оно может ошибиться и, таким образом, сведет на нет усилия пользователя. Более того, такой подход не позволяет пользователю учиться и не дает возможности избегать подобных проблемных ситуаций в будущем. При этом нашему приложению следует запоминать все действия пользователя и позволять аккуратно отменить любое из них, не удаляя сопутствующую информацию и давая пользователю понять, в чем, по мнению приложения, состоит проблема. По существу, приложение сохраняет контрольный журнал действий пользователя. Отсюда вытекает принцип: «Выполняйте проверку, а не редактирование».

Выполняйте проверку, а не редактирование.

Microsoft Word предоставляет нам блестящий пример проведения аудита, а также ужасающий обратный пример. Положительным примером служит то, как этот редактор выполняет проверку орфографии в реальном времени. По мере набора текста появляется красное волнистое подчеркивание, выделяющее слова с подозрительной орфографией (рис. 18.1). Щелчок правой кнопкой мыши по такому слову выводит на экран меню вариантов замены, но вы не обязаны что-либо выбирать, и ваша работа не прерывается диалоговыми окнами и другим модальным идиотизмом.

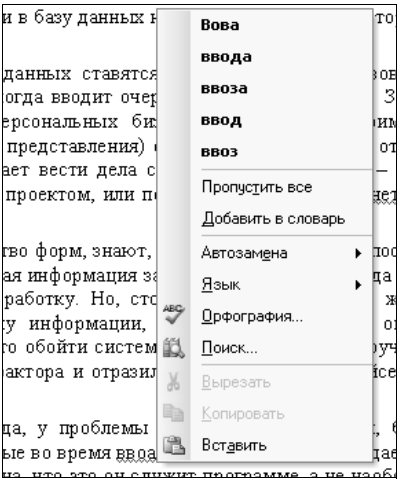


Рис. 18.1. *В Microsoft Word функция автоматической проверки орфографии подчеркивает ошибочные слова, предоставляя тем самым немодальную обратную связь. Щелчок правой кнопкой мыши выводит на экран меню, позволяющее выбрать один из вариантов правильного написания*

Зато функция Автозамена поначалу несколько раздражает. Когда вы вводите текст, она без уведомления изменяет слова, которые считает написанными неправильно. Эта функция оказывается очень полезной для исправления незначительных опечаток. К сожалению, исправления, сделанные программой, не отслеживаются, и пользователь не имеет ни малейшего представления, что из набранного было изменено. Было бы лучше, если бы редактор Word отмечал, что именно исправлено, – на тот случай, если он ошибся (что весьма вероятно при вводе технического текста, насыщенного специальными терминами и сокращениями).

Гораздо больше пугает функция Автоформат, которая пытается интерпретировать звездочки и цифры в тексте как сигнал для автоматического форматирования нумерованных списков и других элементов текста. Когда функция Автоформат срабатывает правильно, кажется, будто произошло чудо, однако часто она делает не то, что нужно, в то время как отмена ее действий затруднена. Функция Автоформат пытается быть умнее, чем следует. Мыслительные процессы нужно делегировать человеку. К счастью, эта функция может быть отключена (хотя найти, где она отключается, не так-то легко). Кроме того, в последних версиях Word присутствует специальное контекстное меню, позволяющее корректировать настройки функции Автоформат.

В реальном мире люди принимают частично заполненные и некорректно заполненные документы каждый день. Мы ставим отметку, что документ надо исправить, и возвращаемся к нему позже. Если мы забудем исправить ошибку, то исправим ее тогда, когда она в конце концов будет обнаружена. Даже если мы никогда ее не исправим, то не умрем от этого. Использование программ для повышения эффективности нашей работы по сбору данных – вполне обоснованная деятельность, и во многих случаях она соответствует нашим намерениям. (Никому не хочется указать неверный адрес доставки, приобретая дорогую вещь через Интернет.) Однако при этом возможно проектировать приложения таким образом, чтобы они были лучше приспособлены к тому, как подходят к решению подобных задач люди. Чисто техническая цель сохранения целостности данных не должна вызывать головную боль у наших пользователей.

19

Указание, выделение, непосредственное манипулирование

Современные графические пользовательские интерфейсы основаны на идее непосредственного манипулирования графическими объектами, находящимися на экране, – кнопками, ползунками и другими функциональными элементами управления, а также пиктограммами и прочими вариантами представления объектов данных. Возможность выбирать и изменять объекты на экране лежит в фундаменте проектируемых сегодня интерфейсов. Однако для такого манипулирования нужны достаточно гибкие механизмы ввода. В этой главе мы поговорим как об основах непосредственного манипулирования, так и о различных устройствах, благодаря которым оно становится возможным.

Непосредственное манипулирование

В 1974 году Бен Шнейдерман (Ben Shneiderman) предложил термин *непосредственное манипулирование* для описания стратегии проектирования интерфейсов, состоящей из трех важных частей:

- визуальное представление объектов, с которыми работает приложение;
- видимые жестикуляционные механизмы для манипулирования этими объектами (как противоположность текстовым командам);
- моментальное отображение результатов действий.

Менее строгое определение могло бы выглядеть так: непосредственное манипулирование есть щелчок по объекту и его перетаскивание. И хотя такое определение справедливо, с ним легко упустить из виду ключевой момент, тонко подмеченный Шнейдерманом. Обратите внимание, что два из трех пунктов его определения относятся к визуальной обратной связи, которую программа предоставляет пользователю.

Возможно, более точным было бы название «наглядное манипулирование», которое лучше подчеркивает важность того, что пользователь *видит* во время этого процесса. К сожалению, во многих случаях идиомы непосредственного манипулирования реализуются без адекватной визуальной обратной связи, и в результате взаимодействие не оставляет ощущения непосредственного манипулирования.

ПРИНЦИП проектирования

Полноценная визуальная обратная связь – ключ к успешному непосредственному манипулированию.

Еще одно важное соображение: пользователи могут непосредственно манипулировать лишь теми объектами, которые приложение вывело на экран. Иначе говоря, пользователю необходимо видеть то, чем он манипулирует. Если вы хотите внедрять эффективные идиомы непосредственного манипулирования в свои программные продукты, позаботьтесь о тщательной графической проработке и достаточной детализации при выводе на экран данных, объектов, элементов управления и курсоров.

Непосредственное манипулирование – простой, прямолинейный, удобный в применении и легко запоминающийся способ взаимодействия. Однако сказанное вовсе не означает, что, столкнувшись впервые с той или иной идиомой манипулирования, пользователи сразу интуитивно постигают ее суть и понимают, как ее применять. Часто пользователей приходится обучать идиомам непосредственного манипулирования. Однако сильная сторона этих идиом в том, что их легко освоить и трудно забыть. Это классический пример идиоматического проектирования. Благодаря своей наглядности и непосредственности это взаимодействие очень похоже на взаимодействие с объектами реального мира, поэтому приобретение соответствующих навыков дается нам просто.

Вот что сказано в отношении непосредственного манипулирования в руководстве Human Interface Style Guide (Руководство по стилю интерфейса для человека) компании Apple: «Пользователи хотят чувствовать, что управляют действиями компьютера». Сам пользовательский интерфейс Macintosh ясно демонстрирует, что Apple считает непосредственное манипулирование фундаментальным принципом проектирования хорошего пользовательского интерфейса. Однако Дон Норман, авторитет в области дизайна, ориентированного на пользователя, говорит: «Непосредственное манипулирование и системы, где инициатива принадлежит человеку, имеют свои недостатки. Хотя они обычно просты в использовании, приносят удовольствие и развлекают человека, для настоящей работы они часто малопригодны. Они требуют, чтобы пользователь непосредственно выполнял требуемое действие, а он может оказаться недостаточно компетентным для этого». Кому же мы должны верить?

Конечно же, обоим. Непосредственное манипулирование – исключительно мощный инструмент, но он может требовать от пользователей приобретения навыков эффективного использования. Многие идиомы непосредственного манипулирования требуют координации движений и понимания происходящего.

К примеру, перемещение файлов между папками в Проводнике (Windows Explorer) может быть сложной задачей, требующей ловкости и предусмотрительности. Помните о подобных барьерах, когда проектируете идиомы непосредственного манипулирования. Некоторый объем непосредственного манипулирования, как правило, полезен, однако можно и перестараться – здесь многое зависит от контекстов использования продукта персонажами. Следует задаваться вопросами о том, манипуляция какими объектами пользователям действительно необходима, а для каких объектов лучше реализовать косвенное манипулирование.

Большинство идиом непосредственного манипулирования можно отнести к одной из семи категорий:

- указание;
- выделение;
- перетаскивание;
- манипулирование элементами управления;
- палитры инструментов;
- манипулирование объектами (например, перемещение, изменение формы и размера);
- создание связей между объектами.

На протяжении этой главы мы поочередно обсудим все перечисленные категории, а начнем с основ – с устройств указания (таких как мышь), обеспечивающих ввод в современных графических пользовательских интерфейсах.

Устройства указания

Непосредственное манипулирование объектами на экране становится возможным благодаря использованию **устройства указания**. Очевидно, лучший способ указать на что-либо – использовать пальцы. Они очень удобные, у любого читателя они наверняка под рукой. Единственный недостаток пальцев – слишком тупые кончики для того, чтобы точно осуществлять целеуказание на экранах высокого разрешения; к тому же, большинство экранов высокого разрешения не способны распознать, что на них указывают. Вследствие этого ограничения нам приходится использовать разнообразные устройства указания, наиболее популярным из которых является мышь.

Передвигая мышь по столу, вы видите, как на экране соответствующим образом перемещается символ – курсор. Передвиньте мышь влево – и курсор переместится влево; отодвиньте мышь от себя – и курсор переместится вверх по экрану. Когда вы впервые взяли мышь в руки, у вас сразу возникло ощущение, что мышь и курсор как-то связаны. Это ощущение крайне легко запомнить и трудно забыть. В фильме «Звездный путь IV. Путешествие домой» («Star Trek IV: The Voyage Home») Скотти (один из лучших инженеров XXIV века) попадает на Землю XX века и пытается работать на компьютере. Он берет мышь рукой, подносит ко рту и пытается говорить в нее, как в микрофон. Сцена забавна и вместе с тем правдоподобна: мышь не имеет никаких визуальных подсказок о назначении, помогающих понять, что она является устройством указания. Но как только нам продемонстрируют, что движения курсора на экране соответствуют движениям мыши, понимание приходит моментально. Как мы уже сказали, все идиомы приходится изучать – но хорошую идиому достаточно изучить всего один раз. С этой точки зрения мышь определенно является хорошей идиомой.

Разумеется, проектировщику необходимо принимать во внимание и ряд других типов устройств указания, включая трекболы, сенсорные панели (touchpads, trackpads), планшеты и сенсорные экраны. Следует помнить, что если первые два типа устройств ведут себя подобно мыши (имея другие эргономические показатели), то планшеты и сенсорные экраны несколько отличаются от нее. Мышь является устройством относительного указания: перемещение мыши вызывает перемещение курсора *относительно его текущей позиции;* планшеты же обычно дают абсолютное позиционирование: каждой точке на планшете *напрямую соответствует совершенно определенная точка экрана*. Если оторвать перо от планшета в левом верхнем углу и снова прикоснуться пером к планшету уже в правом нижнем углу, курсор немедленно перескочит из левого верхнего угла экрана в правый нижний угол. То же верно и для сенсорных экранов.

Работа с мышью

При перемещении мыши по экрану есть отчетливое различие между маленькими и большими расстояниями. В близкий пункт назначения мышь можно передвинуть, не отрывая кисть от стола; в остальных случаях кисть приходится приподнимать и переносить. Передвигая курсор при неподвижном запястье, вы используете точную моторику мышц пальцев. Когда вы отрываете кисть от стола, чтобы совершить перемещение на большее расстояние, то используете грубую моторику мышц руки. Переход от грубой моторики к точной и обратно создает дополнительную нагрузку. Чтобы выполнить его, необходимо согласовать движения двух групп мышц, что требует определенной ловкости, на достижение которой у пользователя компьютера уходят силы и время. (Это напоминает рисование: чтобы научится прилично рисовать, требуется практика.) Профессиональные машинистки не любят опера-

ции, заставляющие их убирать руки из исходного положения на клавиатуре, поскольку такие действия требуют переключений между группами мышц. Аналогично этому перемещение указателя мыши через весь экран к нужному элементу требует переключения с точной моторики на грубую и затем обратно на точную. Не заставляйте пользователей делать это постоянно.

При щелчке кнопкой мыши также задействована тонкая моторика – без нее нажатие на кнопку приводило бы к непроизвольному смещению мыши и указателя, мешающему выполнить намеченное действие. Пользователь должен научиться располагать кисть на столе в удобной позиции, затем переключаться в режим тонкой моторики, чтобы поместить курсор в нужное место, и удерживать это положение при щелчке кнопкой. Когда указатель мыши находится далеко от нужного элемента управления, пользователю приходится сначала использовать грубую моторику, чтобы подвести указатель к элементу управления, а затем переходить к точной, чтобы завершить операцию. Некоторые элементы управления, в частности полосы прокрутки, усложняют задачу, заставляя пользователей переключаться между грубой и точной моторикой несколько раз, чтобы завершить взаимодействие (рис. 19.1).

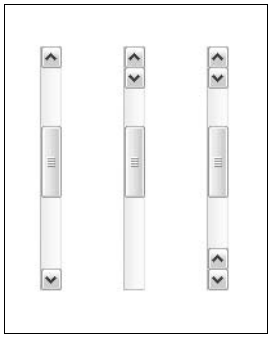


Рис. 19.1. *Всем знакомая полоса прокрутки (слева) является одним из самых сложных в использовании элементов графического пользовательского интерфейса. Чтобы переключиться с прокрутки вверх на прокрутку вниз, пользователь вынужден совершить переход от точной моторики, необходимой при выполнении щелчка по верхней кнопке, к грубой моторике для перевода указателя мыши на противоположный конец полосы прокрутки. Затем пользователь переключается обратно на точную моторику, чтобы точно позиционировать мышь и щелкнуть по нижней кнопке. Если полосу прокрутки слегка модифицировать так, как показано на среднем рисунке, то есть расположить кнопки рядом, то проблема исчезнет. (Полосы прокрутки Macintosh можно настроить, расположив кнопки со стрелками внизу полосы прокрутки.) Полоса прокрутки, изображенная справа, выглядит слегка перегруженной визуально, но обеспечивает наибольшую гибкость взаимодействия. Колесико прокрутки на устройстве ввода также является замечательным решением этой проблемы. Более подробная информация о полосах прокрутки содержится в главе 21*

Проектировщикам крайне важно обращать пристальное внимание на способности пользователей, их подготовку, контексты использования – только так можно принять осмысленное решение об уровне сложности моторики, которая должна поддерживаться интерфейсом. В данном случае это поиск тонкого равновесия между снижением сложности и уменьшением требуемых от пользователя усилий с одной стороны и созданием полезных и мощных инструментов – с другой. Практически всегда разумно группировать инструменты, используемые совместно.

Мышь вызывает проблемы не только у менее ловких людей, но временами и у многих опытных пользователей, в особенности у тех, кто много печатает слепым методом. В решении широкого круга задач, связанных с интенсивной обработкой данных, клавиатура превосходит мышь. Очень неприятно, когда требуется снять руки с клавиатуры лишь для того, чтобы переместить курсор посредством мыши и тут же снова вернуться к клавиатуре. На заре персональных компьютеров не было никаких устройств ввода кроме клавиатуры, сегодня же единственным применимым устройством ввода часто оказывается мышь. Программы должны обеспечивать полную поддержку как мыши, так и клавиатуры в качестве средств управления навигацией и выделением.

В задачах, связанных с навигацией и выделением, обеспечивайте поддержку как мыши, так и клавиатуры.

Значительный процент пользователей испытывает определенные сложности при работе с мышью, так что для проектирования успешных продуктов нам требуется учитывать интересы этой группы пользователей наравне с интересами тех, кто владеет мышью виртуозно. Это означает, что для каждой идиомы, связанной с мышью, должна быть хотя бы одна альтернатива, не требующая использования мыши. Конечно, это возможно не всегда. Было бы глупо пытаться реализовать рисование без мыши. Однако в большинстве приложений для корпоративного использования и для бизнеса клавиатурные команды вполне уместны.

Кнопки мыши

Создатели мыши пытались понять, сколько кнопок требуется, но так и не пришли к согласию. Некоторые считали, что достаточно одной кнопки, другие отстаивали вариант с двумя кнопками. Третьи предлагали модель с несколькими кнопками, которыми можно было бы щелкать по отдельности или в сочетаниях, так что пять кнопок обеспечивали бы 32 различных сочетания.

В конечном итоге фирма Apple создала однокнопочную мышь для Macintosh, корпорация Microsoft – двухкнопочную, а UNIX-сообщество (в частности, Sun Microsystems) предпочло трехкнопочную. Обширное пользовательское тестирование Apple выявило, что для новичка опти-

мальное число кнопок – одна, и так однокнопочная мышь была увековечена в пантеоне истории Apple. Это прискорбно, потому что правая кнопка мыши обычно выходит на арену, как только человек покидает ряды новичков и становится вечным середняком. Мышь с одной кнопкой жертвует удобством большинства пользователей ради простоты для начинающих. Не так давно компания Apple признала важность контекстных меню, вызываемых по правой кнопке мыши, и теперь ее компьютеры поставляются с двухкнопочной мышью.

Левая кнопка мыши

Говоря в общем, левая кнопка используется для всех основных функций непосредственного манипулирования, таких как активизация элементов управления, выделение, рисование и т. п. Основное предназначение левой кнопки – активизация или выделение. Для стандартных элементов управления, таких как кнопки или флажки, щелчок левой кнопкой мыши означает нажатие на кнопку или установку/сброс флажка. Щелчок левой кнопкой в области данных обычно подразумевает выделение. Идиомы выделения мы обсудим позже в этой главе.

Правая кнопка мыши

Долгое время Microsoft и другие компании просто игнорировали существование правой кнопки. Лишь некоторые отважные программисты привязывали к ней какие-либо функции, но эти функции были дополнительными, необязательными или предназначались для более опытных пользователей. Когда компания Borland International сделала правую кнопку инструментом доступа к диалоговому окну свойств объекта, реакция отрасли была двойственной, хотя само решение встретило, как говорится, хороший прием. Положение дел изменилось с появлением Windows 95, в которой Microsoft наконец последовала примеру Borland. Сегодня правая кнопка мыши играет важную и крайне полезную роль, обеспечивая непосредственный доступ к свойствам и другие контекстно-зависимые функции и действия над объектами.

Средняя кнопка мыши

Вообще говоря, нельзя полагаться на наличие у мыши пользователя средней кнопки, если только речь не идет о специализированном приложении, для которого будут куплены все требуемые аппаратные средства. Как следствие, в большинстве приложений средняя кнопка используется лишь для быстрого доступа к определенным функциям. В своем руководстве по стилю Microsoft утверждает, что среднюю кнопку следует «назначать операциям или функциям, уже представленным в интерфейсе», то есть повторяет сказанное некогда о правой кнопке. У нас есть знакомые, активно использующие среднюю кнопку и готовые присягнуть, что это очень удобно. Пользуясь настройками драйвера мыши, они определяют среднюю кнопку как эквивалент двойного щелчка левой кнопкой.

Колесо прокрутки

Колесо прокрутки – одно из самых полезных нововведений в области устройств указания. Существуют различные его варианты, но обычно это небольшое колесико, встроенное в мышь и расположенное под средним пальцем пользователя. Вращение колесика вперед приводит к прокрутке окна вверх, а вращение назад – к прокрутке вниз. Колесо прокрутки замечательно тем, что позволяет пользователям избежать сложностей, присущих взаимодействию с полосами прокрутки (см. рис. 19.1).

Служебные клавиши

Использование **служебных клавиш** в сочетании с мышью позволяет расширить идиомы непосредственного манипулирования. К служебным относятся клавиши <Ctrl>, <Alt>, <Command> (на компьютерах Apple) и <Shift>.

Как правило, эти клавиши изменяют функцию взаимодействий, связанных с выделением и перетаскиванием. Так, в Проводнике (Windows Explorer) удержание клавиши <Ctrl> при перетаскивании файла изменяет функцию с перемещения на копирование. Эти клавиши также повсеместно применяются для подстройки поведения мыши – удержание клавиши <Shift> при перетаскивании часто ограничивает движение курсора одним направлением (вертикалью или горизонталью). О подобных соглашениях мы поговорим далее в этой главе.

Apple с давних пор придерживалась четко сформулированных стандартов сочетания служебных клавиш с мышью, и потому решения в этой области в целом выглядят последовательными. В мире Windows никто не взял на себя задачу создания аналогичных четких стандартов на интерфейсы, однако возникли некоторые соглашения (которые часто похожи на применяемые Apple).

Хорошая идея – использование курсорных подсказок для динамического отображения смысла служебных клавиш. Когда нажата служебная клавиша, следует изменить курсор, отразив таким образом новую функцию идиомы.

Используйте курсорные подсказки для отражения смысла служебных клавиш.

Указание и щелчок мышью

На самом нижнем уровне имеются две атомарных операции, которые можно выполнить с помощью мыши: вы можете перемещать ее, указывая на различные объекты, и можете щелкать ее кнопками. Любое другое действие, выполняемое при помощи мыши, является сочетанием некоторого количества этих операций. Полный набор операций

с мышью, в которых не применяются служебные клавиши, приведен ниже. В скобках даны краткие названия этих операций, которые мы будем использовать в дальнейшем обсуждении.

- Указание (указание).
- Указание, щелчок левой кнопкой, отпускание (щелчок).
- Указание, щелчок правой кнопкой, отпускание (щелчок правой кнопкой).
- Указание, щелчок левой кнопкой, перетаскивание, отпускание (щелчок и перетаскивание).
- Указание, щелчок левой кнопкой, отпускание, щелчок левой, отпускание (двойной щелчок).
- Указание, щелчок левой кнопкой, щелчок правой кнопкой, отпускание, отпускание (аккордный щелчок).
- Указание, щелчок левой кнопкой, отпускание, щелчок, перетаскивание, отпускание (двойной щелчок и перетаскивание).

Опытный пользователь, возможно, выполняет все семь действий, но обычному пользователю известны только первые пять.

Указание

Эта простая операция является краеугольным камнем графического пользовательского интерфейса и основой всех операций с мышью. Пользователь перемещает мышь, пока курсор на экране не окажется над нужным объектом, тем самым указав на него. Объекты интерфейса могут реагировать на факт указания, даже если щелчка не было. Часто объекты, допускающие непосредственное манипулирование, слегка меняют свой внешний вид, чтобы обозначить данное свойство, когда курсор проходит над ними. Такое поведение объектов называется **отзывчивостью** и подробно обсуждается далее в этой главе.

Щелчок

Удерживая курсор над целью, пользователь нажимает на кнопку мыши и отпускает ее. Как правило, за этим действием закреплено изменение состояния управляющего элемента либо выделение (выбор) объекта. В матрице, содержащей текст или состоящей из ячеек, щелчок означает «перенести точку выделения в это место». В случае кнопки изменение состояния означает, что пока кнопка мыши нажата, а указатель расположен на элементе управления, экранная кнопка будет находиться в нажатом состоянии. Когда пользователь отпустит кнопку мыши, экранная кнопка сработает и будет выполнено связанное с ней действие.

Одиночный щелчок выбирает информационный объект либо изменяет состояние элемента управления.

Если же пользователь, продолжая удерживать кнопку мыши в нажатом положении, уберет указатель с элемента управления, экранная кнопка вернется в исходное состояние (при этом фокус ввода будет оставаться на ней, пока пользователь не отпустит кнопку мыши). Когда пользователь отпускает кнопку мыши, фокус ввода снимается и ничего не происходит. Это удобный путь отступления для пользователя, если он изменил свое решение или же понял, что ошибся кнопкой. Механика событий «кнопка мыши нажата» и «кнопка мыши отпущена», возникающих в процессе щелчка, подробно обсуждается далее в этой главе.

Щелчок и перетаскивание

Эта гибкая операция имеет много вариантов применения, включая выделение, изменение формы, изменение положения, рисование и перетаскивание объектов. Все это мы будем обсуждать на протяжении этой главы и в оставшейся части книги.

Как и в случае со щелчком, часто важно оставить путь отступления для пользователей, которые запутались или ошиблись. Хороший пример на эту тему – полоса прокрутки в Windows. Она позволяет осуществлять прокрутку, даже если курсор не расположен непосредственно над полосой (представьте себе, как сложно было бы пользоваться этим элементом управления, если бы он вел себя подобно кнопке). Однако, если увести курсор слишком далеко от полосы прокрутки, она вернется в положение, в котором была до щелчка. Такое поведение имеет смысл, поскольку прокрутка на большое расстояние требует использования грубой моторики, что затрудняет удержание указателя мыши на узкой полосе прокрутки. Если указатель отходит слишком далеко, полоса прокрутки делает разумное предположение, что пользователь не собирался выполнять прокрутку, а имел в виду что-то другое. В некоторых приложениях границы влияния указателя мыши установлены очень близко к полосе прокрутки, в результате чего она ведет себя слишком темпераментно, срывая планы пользователей.

Двойной щелчок

Поскольку двойной щелчок состоит из двух одиночных, логично предположить, что первое составляющее действие этой операции совпадает с действием одиночного щелчка. Именно так и происходит, если мышь указывает на экранный объект: один щелчок выделяет объект, а двойной щелчок выделяет объект и выполняет некоторую операцию над объектом.

Двойной щелчок означает одинарный щелчок с последующим действием.

Эта базовая интерпретация пришла к нам от Xerox Alto/Star через Macintosh и остается стандартом для современных графических поль-

зовательских интерфейсов. При этом игнорируется тот факт, что двойной щелчок вызывает трудности у менее ловких пользователей – для некоторых он является мучительным действием, а для кого-то – и вовсе невыполнимым. Решение этих проблем с доступностью состоит в том, чтобы, реализуя идиомы двойного щелчка, сопровождать каждую из функций эквивалентными идиомами одиночного щелчка.

В то время как для файлов и пиктограмм приложений двойной щелчок имеет ясное значение, для большинства управляющих элементов он лишен смысла, и «лишний» щелчок просто игнорируется либо (что случается чаще) интерпретируется как еще один независимый щелчок. В зависимости от того, о каком элементе управления идет речь, такая трактовка может стать как благом, так и проблемой. В случае с элементом типа кнопка-выключатель может оказаться, что вы вернули ее в исходное состояние (сперва быстро включили, а затем тут же выключили). Если элемент управления исчезает с экрана после первого щелчка, как, например, кнопка OK в диалоговом окне, то результат становится абсолютно непредсказуемым, так как второй щелчок может попасть на кнопку, расположенную в исходном окне.

Аккордный щелчок

Аккордный щелчок – это щелчок сразу двумя кнопками, хотя в реальности пользователю необязательно нажимать и отпускать их строго одновременно. Чтобы щелчок распознавался как аккордный, вторая кнопка должна быть нажата до того, как будет отпущена первая.

Существует два вида аккордных щелчков. Первый и простейший: пользователь наводит курсор на объект и щелкает сразу двумя кнопками. Эта довольно неуклюжая идиома не нашла широкой поддержки в современных программах, хотя некоторые изобретательные и отчаянные программисты посчитали ее подходящей заменой клавише <Shift> в процессе выделения.

Второй вариант заключается в использовании аккордного щелчка для отмены перетаскивания. Перетаскивание начинается с удержания одной кнопки, а затем пользователь подключает к работе вторую кнопку. Хотя эта техника выглядит менее понятной, чем первый вариант, она нашла более широкое применение в приложениях. Она отлично подходит для отмены операций перетаскивания, и мы обсудим ее более подробно чуть позже в этой главе.

Двойной щелчок и перетаскивание

Это еще одна идиома «только для специалистов». Безошибочно выполнить двойной щелчок и перетаскивание столь же трудно, как похлопывать себя по голове, одновременно поглаживая по животу. Как и тройной щелчок, эта операция полезна только в специализированных монопольных приложениях. Применяйте ее как расширенный вариант выделения. Например, двойным щелчком в редакторе Word

выделяется целое слово. Можно расширить эту функцию, разрешив выделение по словам с помощью двойного перетаскивания.

В большом монопольном приложении, имеющем разные варианты выделения, реализация подобной идиомы вполне уместна. Однако в подавляющем большинстве продуктов мы рекомендуем ограничиться более простыми операциями с мышью.

События «кнопка отпущена» и «кнопка нажата»

Всякий раз, когда пользователь щелкает кнопкой мыши, программа должна обработать два отдельных события: «кнопка нажата» и «кнопка отпущена». Конкретная интерпретация этих событий меняется от платформы к платформе и от продукта к продукту. В пределах конкретного продукта (а в идеале – в пределах платформы) требуется строгая унификация этих действий.

Если щелчок выделяет объект, это всегда должно происходить по событию «кнопка нажата». Щелчок кнопкой может оказаться первым шагом в перетаскивании, а перетаскивание невозможно без выделения объекта.

Если курсор находится над объектом или данными, событие «кнопка нажата» должно приводить к выделению этого объекта или данных.

С другой стороны, если курсор расположен не над данными, которые можно выделить, а над элементом управления, нажатие на кнопку должно *наметить* изменение состояния этого элемента управления. И лишь когда элемент «увидит», что кнопка мыши отпущена, происходит собственно смена состояния (рис. 19.2).

Когда курсор находится над элементом управления, событие «кнопка нажата» означает подготовку к действию, событие «кнопка отпущена» приводит к выполнению действия.

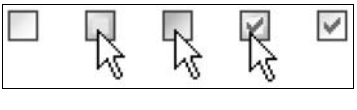


Рис. 19.2. *Эти изображения демонстрируют обратную связь и изменение состояния флажка в операционной системе Windows XP. На первом изображении показан невыделенный флажок; на втором флажок реагирует на курсор мыши, расположенный над ним; на третьем показана реакция на щелчок (кнопка нажата); на четвертом видно, что происходит, когда кнопку отпускают, но не уводят курсор с флажка (кнопка отпущена); на последнем изображении показан флажок, с которого увели курсор. Обратите внимание: здесь присутствует визуальная обратная связь по щелчку, но состояние флажка не меняется, пока кнопка не будет отпущена*

Описанный механизм дает пользователю возможность избежать последствий неосторожного щелчка. Например, после щелчка по экранной кнопке пользователь может увести указатель за ее пределы и отпустить кнопку мыши там; экранная кнопка вернется в неактивное состояние. Сходным образом ведет себя флажок: когда пользователь нажимает кнопку мыши, флажок демонстрирует свою готовность, однако сама «галочка» не появится, пока пользователь не отпустит кнопку мыши.

Указание и курсор

Курсор – это видимое представление положения мыши на экране. По общепринятому соглашению он имеет вид маленькой стрелки, указывающей влево и вверх, однако программа может как угодно изменять его вид при условии, что он остается ограниченным по размеру (32×32 пиксела в Windows XP). Поскольку курсор часто используется для указания на небольшие объекты, требуется точность указания в один пиксел и способ определить, на какой именно пиксел наведена мышь. Поэтому курсор любой формы имеет так называемую **горячую точку** (hotspot) – пиксел, определяющий активную точку указателя. Естественно, что у стандартной стрелки горячей точкой является ее острие. Независимо от формы курсора он всегда имеет горячую точку размером в один пиксел.

Как уже говорилось, ключ к успешному непосредственному манипулированию – в богатой визуальной обратной связи. Пользователям должно быть очевидно, какие аспекты интерфейса поддаются манипулированию, какие являются информационными, а какие служат просто украшениями. Обратная связь через курсор мыши является особенно важной для создания эффективных идиом взаимодействия.

Отзывчивость и подсказки

Возвращаясь к концепции ожидаемого назначения, предложенной Норманом (см. главу 13), отметим исключительную важность визуальных подсказок о назначении различных элементов интерфейса. Области экрана или объекты, которыми пользователь может манипулировать, мы называем **отзывчивыми**. Так, экранная кнопка отзывчива, потому что ее можно «вдавить» курсором мыши. Отзывчив любой объект, который можно перетащить, отзывчива любая ячейка электронной таблицы и любой символ текстового документа.

В большинстве случаев нужно особо сообщать пользователю об отзывчивости объекта. Единственная ситуация, в которой не следует это делать, – это представление сложной функциональности, ориентированное исключительно на подготовленных пользователей, когда нет необходимости заботиться об их способности обучаться и пользоваться приложением. В этих случаях есть возможность сэкономить экранное

пространство и не рассеивать фокус внимания ради передачи отзывчивости, найдя для них более полезное применение. Однако не следует выбирать этот путь без веских на то оснований.

Передавайте отзывчивость элементов интерфейса с помощью визуальных подсказок.

Существует три основных способа сообщить пользователю об отзывчивости объекта: при помощи статической визуальной демонстрации ожидаемого назначения прямо на самом объекте, путем динамического изменения его визуальных свойств для отражения его ожидаемого назначения или изменением визуальных свойств курсора, когда он проходит над объектом или взаимодействует с ним.

Объектные подсказки

Статическая объектная подсказка – это передача отзывчивости объекта через его собственные статические визуальные свойства. Например, псевдотрехмерный вид экранной кнопки является статической объектной подсказкой, поскольку демонстрирует физическое ожидаемое назначение – возможность нажатия (рис. 19.3).

В интерфейсах, насыщенных объектами и элементами управления, количество визуальных деталей, необходимых для реализации статических объектных подсказок, может выйти за пределы разумного. Если все на экране имеет трехмерный вид, демонстрируя ожидаемое назначение, интерфейс начинает походить на сад скульптур. Кроме того, статические объектные подсказки накладывают определенные ограничения на минимальный размер объекта: объекты должны быть достаточно крупными для визуального отражения ожидаемого назначения. Для преодоления этих сложностей требуется использование динамических визуальных подсказок.

Динамические визуальные подсказки работают следующим образом: когда курсор проходит над отзывчивым объектом, объект видоизменяется (рис. 19.3). Это происходит до того, как будет нажата какая-либо кнопка мыши, при одном лишь наложении курсора и объекта. Обычно

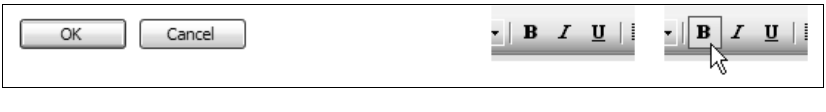


Рис. 19.3. *Кнопки слева – пример статических визуальных подсказок. О возможности их нажатия сообщает объемная визуализация. Инструментальные кнопки-значки на панели справа – пример динамических визуальных подсказок. Переключатель полужирного текста на первый взгляд не кажется кнопкой, но наведение на него курсора мыши изменяет его внешний вид, отражая ожидаемое назначение*

это называется «перекатыванием» (rollover). Хороший пример такого поведения дают **кнопки-значки** (butcon) на панелях инструментов: они не имеют постоянного визуального отражения ожидаемого назначения, указывающего на возможность нажатия, но при наведении курсора на любую кнопку-значок ожидаемое назначение визуализируется. В результате пользователь получает убедительную подсказку о том, что элемент управления ведет себя как кнопка, а отсутствие постоянного визуального отражения ожидаемого назначения резко снижает визуальную нагруженность панели инструментов.

Визуальный активный отклик должен появляться, когда кнопка мыши нажата (но не отпущена) внутри области элемента управления. Элемент управления наглядно показывает, что готов изменить состояние (см. рис. 19.2). Это важное действие, о котором разработчики, создающие собственные элементы управления, часто забывают.

Нажимаемая кнопка должна перейти из «выпуклого» состояния в «вогнутое», флажок должен подсветить рамку, но не показывать галочку. Активный отклик – важный механизм обратной связи для любого элемента управления, который меняет состояние или выполняет действие. Этот механизм сообщает пользователю, что произойдет определенное действие, как только он отпустит кнопку. Активный отклик является также важной составляющей механизма отмены действия. Когда пользователь нажал экранную кнопку, она реагирует, становясь вдавленной. Если пользователь выведет курсор за пределы экранной кнопки, продолжая удерживать кнопку мыши, экранная кнопка должна вернуться в свое обычное выступающее состояние. Если после этого пользователь отпустит кнопку мыши, экранная кнопка не будет активирована (что логично вытекает из отсутствия визуальной реакции в этот момент).

Курсорные подсказки

Курсорные подсказки передают отзывчивость элементов интерфейса через изменение вида курсора, когда он проходит над объектом или определенной областью экрана. К примеру, когда курсор проходит над границей окна, он превращается в двунаправленную стрелку, обозначающую ось, вдоль которой можно растягивать окно. Это единственное визуальное отображение ожидаемого назначения, показывающее, что окно можно растягивать.

Применяйте курсорные подсказки для передачи отзывчивости элементов интерфейса.

Курсорные подсказки должны, прежде всего, давать ясно понять, что объект является отзывчивым. Часто бывает полезно указывать еще и тип возможного непосредственного манипулирования.

Вообще говоря, элементы управления следует сочетать со статическими или динамическими визуальными подсказками, а *отзывчивые* (поддающиеся манипулированию) данные – скорее с курсорными подсказками. К примеру, сложно сделать так, чтобы плотные табличные данные визуально сообщали о своей пластичности, не мешая простоте восприятия, поэтому наиболее эффективным методом здесь будут именно курсорные подсказки. Некоторые элементы управления малы, и пользователям сложно сразу понять, что они действуют как кнопки, и курсорные подсказки жизненно необходимы для успешной работы таких элементов управления. Разделители колонок и экрана в Microsoft Excel – хорошие примеры этого подхода, как можно видеть на рис. 19.4.

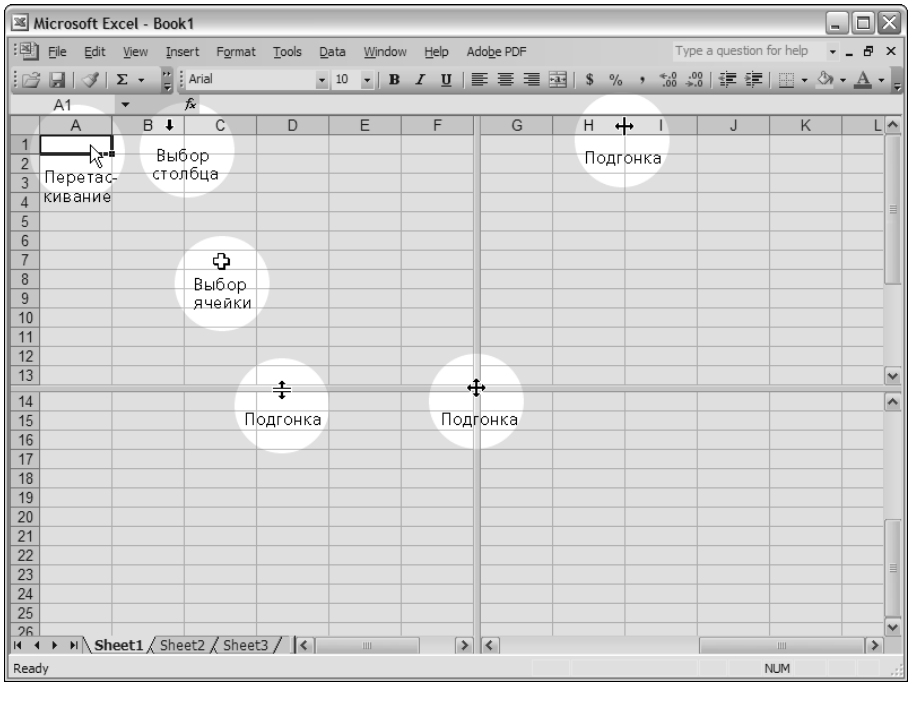


Рис. 19.4. *Курсорные подсказки используются в Excel для обозначения элементов управления, которые сами не сообщают пользователю о своей отзывчивости. Ширину отдельных колонок и высоту строк можно устанавливать, перетаскивая короткие вертикальные линии между парами колонок и строк – курсор при этом превращается в двунаправленную горизонтальную стрелку, сообщая об отзывчивости и указывая допустимые направления перетаскивания. То же верно и для разделителей экрана. Когда курсор оказывается над невыделенной редактируемой ячейкой таблицы, он превращается в курсор-плюс, а когда он расположен над выделенной ячейкой, то указывает на возможность перетаскивания*

Индикация ожидания посредством курсора

Существует вариант курсорной подсказки, известный как **индикация ожидания**, – он часто применяется, когда программа совершает действие, временно лишающее ее возможности реагировать на запросы пользователя, например выполняет серьезные вычисления или открывает файл. В данном случае курсор используется для визуального указания на занятость приложения. В Windows курсор превращается в знакомые всем песочные часы. Другие операционные системы выводят на дисплей механические часы, крутящиеся шарики или чашки с дымящимся кофе.

Когда эта идиома только появилась в графических пользовательских интерфейсах, занятость одного приложения приводила к тому, что курсор менялся для всех приложений. Это запутывало пользователей и сбивало их с толку. В современных многозадачных операционных системах этот недостаток устранен, однако предоставлять как можно больше информации об источниках любой задержки или причинах недостаточно быстрого реагирования на запросы пользователя по-прежнему важно.

Выделение

Выбор объекта или элемента управления называется **выделением**. Это простая идиома, обычно реализуемая с помощью указания требуемого объекта и щелчка по нему (хотя существуют и другие методы, связанные с клавиатурой и кнопками). Выделение часто служит основой для более сложных взаимодействий – выбрав объект, пользователь переходит в контекст, подходящий для выполнения действий над этим объектом. Последовательность событий, подразумеваемая такой идиомой, называется **порядком «объект – глагол»**.

Порядок команд и выделение

В основе любого пользовательского интерфейса лежит некоторый способ выражать команды. Практически с каждой командой связан **глагол**, описывающий действие, и **объект**, над которым выполняется это действие (говоря техническим языком, это операция и операнды).

Если задуматься, станет понятно, что любая команда может быть выражена двумя способами: либо сначала глагол, затем объект, либо сначала объект, затем глагол. Обычно эти два способа называют соответственно порядком **«глагол – объект»** и порядком **«объект – глагол»**. В современных пользовательских интерфейсах используются оба варианта.

Порядок «глагол – объект» соответствует порядку формирования команд на естественном языке. Как следствие, системы, управляемые через командную строку, вполне логично копируют эту языковую структуру в своем синтаксисе (к примеру, чтобы удалить файл в системе UNIX, нужно выполнить команду `rm filename.txt`).

Когда графические пользовательские интерфейсы только появились, стало очевидно, что порядок «глагол – объект» не годится. Не имея жестких рамок и формальных структур, присущих командной строке, графические интерфейсы должны увязывать различные взаимодействия внутри одной команды посредством *состояния*. Если пользователь выбрал глагол, система должна перейти в состояние (режим), указывающее на ожидание того, что пользователь выберет объект, к которому будет применен глагол. В простейшем случае пользователь выбирает один объект, и все заканчивается хорошо. Однако если пользователю требуется выполнить действие над многими объектами, система может узнать об этом, лишь если пользователь заранее сообщит количество операндов либо введет еще одну команду, указывающую, что выбраны все нужные объекты. Оба варианта взаимодействия крайне неуклюжи и требуют, чтобы пользователь выражался совершенно неестественным образом, который сложно освоить. То, что замечательно работает в четко структурированной языковой среде, распадается на части в более свободном мире графических пользовательских интерфейсов.

Порядок «объект – глагол» избавляет нас от необходимости беспокоиться о завершении списка операндов. Пользователь выбирает объекты для операции, после чего сообщает, какое действие (глагол) следует выполнить для этих объектов. Затем приложение выполняет указанную функцию для выбранных данных. Преимущество такого подхода в том, что пользователю оказывается легко выполнить несколько глаголов для одного сложного выделения. Вторая выгода – когда пользователь выбрал объект, приложение может показать только допустимые команды, сокращая когнитивную нагрузку и объем визуальной работы, которая требуется для поиска нужной команды (в визуальном интерфейсе все команды должны быть представлены визуально).

Обратите внимание, что в уравнение проникла новая переменная, которая не нужна и не существует в мире, где правит порядок «глагол – объект». Эта новая переменная называется **выделение**. Поскольку идентификация объектов и выбор глагола перестали быть частями единого взаимодействия, нам требуется механизм для указания выбранных операндов.

На абстрактном уровне модель «объект – глагол» может быть малопонятной, однако выделение – идиома очень простая для понимания и после одной демонстрации запоминается надолго (щелчок по сообщению электронной почты в Outlook с последующим удалением быстро становится привычным действием). При разговоре на обычном языке кажется странным начинать с объекта. В то же время в своих нелингвистических действиях мы часто руководствуемся этой моделью. Так, мы сначала берем в руки банку, а затем открываем ее.

В интерфейсах, не задействующих непосредственное манипулирование (например, в некоторых модальных диалоговых окнах), не всегда требуется выделение. В диалоговых окнах естественным образом реа-

лизована команда выполнения действия над списком объектов – кнопка OK. Здесь пользователи могут выбрать сначала функцию, а затем один или несколько объектов.

При реализации непосредственного манипулирования порядок «объект – глагол» подходит больше, однако и здесь существуют ситуации, в которых более полезным или удобным оказывается порядок «глагол – объект». Это ситуации, когда нет возможности или смысла указывать объекты вне контекста команды. Так, в картографических приложениях пользователь не всегда может выбрать из списка адрес, для которого нужна карта (в то время как в его записной книжке эта функция должна присутствовать обязательно); ему удобнее сказать «я собираюсь построить карту для следующего адреса...»

Дискретное и непрерывное выделение

Выделение – достаточно простое понятие, однако стоит обсудить два его основных варианта. Поскольку обычно, говоря о выделении, подразумевают выделение объектов, эти варианты возникают на основе двух обширных категорий выделяемых данных.

В некоторых случаях данные представлены в виде отдельных визуальных объектов, каждым из которых можно манипулировать независимо от остальных. Пиктограммы на рабочем столе и векторные объекты в графических приложениях представляют собой как раз такие данные. Выделяются эти объекты обычно независимо от того, как они соотносятся друг с другом в пространстве. Они представляют собой **дискретные данные**, и их выделение также **дискретно**. Дискретные данные не обязательно однородны, и дискретное выделение не обязательно является непрерывным.

С другой стороны, некоторые приложения представляют свои данные в виде матрицы, состоящей из большого количества непрерывных фрагментов данных. Текст в текстовом редакторе или ячейки электронной таблицы состоят из сотен и тысяч схожих небольших объектов, образующих единое целое. Эти объекты часто выделяются смежными группами, и мы называем их **непрерывными данными**, а соответствующее выделение – **непрерывным**.

Как непрерывное, так и дискретное выделение поддерживают выделение в один щелчок и выделение перетаскиванием. При одиночном щелчке обычно выделяется минимально возможный дискретный элемент, а щелчок с перетаскиванием позволяет выделить большее количество элементов, однако есть и другие существенные различия.

Смежные (непрерывные) элементы данных, из которых состоит текст в документе текстового редактора, обладают естественным порядком. Нарушение порядка следования букв уничтожает смысл документа. Символы следуют один за другим, образуя осмысленный континуум, и выделение слова или абзаца имеет смысл в контексте данных, тогда

как случайное, несвязное выделение в общем случае лишено смысла. Хотя теоретически возможно разрешить дискретное выделение, например выделение нескольких абзацев, разбросанных по тексту, их визуализация и необходимость защититься от непреднамеренных нежелательных операций над ними породят проблем больше, чем принесут пользы.

С другой стороны, у дискретных данных нет присущего им порядка. И хотя дискретные объекты можно упорядочить многими различными осмысленными способами (скажем, файлы можно отсортировать по дате их изменения), отсутствие сквозного внутреннего связующего признака означает, что пользователи, вероятно, захотят выполнять дискретное выделение (например, удерживая клавишу <Ctrl>, вразнобой выбирать файлы из списка). Разумеется, пользователям может понадобиться и непрерывное выделение по тому или иному организующему признаку (например, выбор старых файлов из конца списка, упорядоченного по времени). Полезность обоих подходов особенно очевидна в приложениях для работы с векторной графикой (Illustrator или PowerPoint). В одних случаях пользователю требуется выполнить непрерывное выделение объектов, расположенных рядом, в других нужно выделить лишь один объект.

Взаимное исключение

Обычно при выделении предыдущее выделение снимается. Такое поведение называется **взаимным исключением**, поскольку одно выделение исключает другое. Как правило, пользователь щелкает по объекту, тот становится выделенным и остается таковым, пока пользователь не выделит что-нибудь еще. Взаимное исключение – правило как для дискретного, так и для непрерывного выделения.

Некоторые приложения позволяют снять выделение с объекта повторным щелчком по нему. Это может привести к курьезной ситуации, когда ничего не выделено и при этом отсутствует точка ввода. Вы сами должны решить, приемлема ли подобная ситуация в вашем продукте.

Кумулятивное выделение

Взаимное исключение часто уместно в операциях непрерывного выделения, потому что в противном случае пользователь рискует не увидеть результат своих действий, если выделение окажется прокручено за пределы экрана. Представьте себе, что можно выделить несколько независимых абзацев текста в разных местах большого документа. Такая возможность может быть полезной, но окажется практически неконтролируемой: пользователям будет легко попасть в ситуацию, когда они изменяют данные непреднамеренно, поскольку не видят всего набора данных, участвующих в операции. Проблему создает прокрутка, а не непрерывное выделение, однако большинство программ, обрабатывающих недискретные данные, позволяет их прокручивать.

А вот в дискретном выделении от взаимного исключения можно отказаться. Пользователь может выделить несколько независимых объектов, последовательно щелкая по ним. Это называется **кумулятивным выделением**. Например, список позволяет пользователю выделить столько элементов, сколько потребуется. Чтобы снять выделение с объекта, следует щелкнуть по нему еще раз. После того как все нужные объекты выделены, по завершающему глаголу выполняется действие над ними.

В большинстве систем с дискретным выделением по умолчанию реализовано взаимное исключение, а кумулятивное выделение можно выполнить только с помощью служебной клавиши. В Windows для непрерывного выделения применяется в основном клавиша <Shift>, тогда как <Ctrl> применяется для дискретного выделения. Например, в графических редакторах, щелкнув по объекту и выделив его, можно добавить к выделению еще один объект. Для этого необходимо щелкнуть по нему, удерживая нажатой клавишу <Shift>.

Интерфейсы, поддерживающие непрерывное выделение, в общем случае не должны допускать кумулятивное выделение (либо обязаны предоставлять средство обзора, делающее кумулятивное выделение управляемым), однако должны позволять *расширять* существующее выделение. Для этих целей, опять же, применяются служебные клавиши. В редакторе Word можно выделить фрагмент текста, если установить курсор в начальную точку, а затем, удерживая клавишу <Shift>, щелкнуть в конечной точке.

В некоторых списках, а также в Проводнике системы Windows (в обоих примерах данные дискретны) кумулятивное выделение выглядит несколько странно. Для выполнения «нормального» дискретного выделения задействуется клавиша <Ctrl>, но затем для *расширения* выделения используется <Shift>, словно это не дискретные, а непрерывные данные. В большинстве случаев такой выбор сбивает пользователя с толку, поскольку конфликтует с общепринятой идиомой дискретного кумулятивного выделения.

Групповое выделение

Операция щелчок-перетаскивание также является основой для группового выделения. В случае непрерывных данных она приводит к расширению выделения от точки, где пользователь нажал кнопку мыши, до точки, где он ее отпустил. Эта операция тоже может быть модифицирована служебными клавишами. Например, в редакторе Word щелчок при нажатой клавише <Ctrl> выделяет целое предложение, так что перетаскивание при нажатой клавише <Ctrl> расширяет выделение предложениями. Монопольные приложения должны обогащать взаимодействие такими вариантами выделения, когда это уместно. Опытные пользователи в конце концов запоминают и применяют такие приемы, если они достаточно просты в исполнении.

В случае набора дискретных объектов щелчок и перетаскивание обычно означают перемещение объекта. Однако если щелкнуть в области между объектами, а не по какому-то конкретному объекту, такой щелчок будет иметь специальный смысл – он создаст **рамку выделения**, изображенную на рис. 19.5.

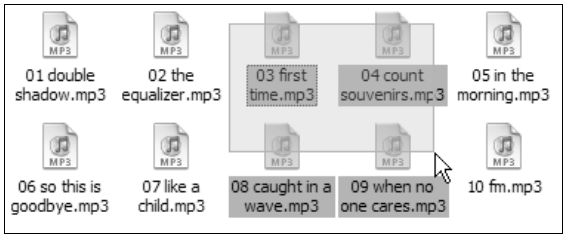


Рис. 19.5. *Если в момент нажатия на кнопку мыши указатель мыши не был расположен на конкретном объекте, щелчок и перетаскивание обычно порождают рамку выделения: все захваченные этой рамкой объекты выделяются, когда пользователь отпускает кнопку мыши. Эта идиома знакома пользователям всех графических и многих текстовых редакторов. Пример на этом рисунке взят из Проводника. Рамка была протянута из левого верхнего угла вправо вниз*

Рамка выделения динамически изменяет свой размер; ее верхний левый угол находится в точке, где пользователь нажал на кнопку мыши, а правый нижний – в точке, где он отпустил кнопку. Когда пользователь отпускает кнопку мыши, все объекты, захваченные рамкой, выделяются как единая группа.

Вставка и замещение

Как мы установили, выделение показывает, какими объектами будет оперировать выполняемая далее функция. Если выполнение этой функции приводит к созданию или вставке новых объектов или данных (посредством клавиатурных сокращений или команды Вставить), эти новые объекты или данные каким-то образом добавляются к выделенным. При дискретном выделении, когда выделен один или несколько объектов, поступающие данные передаются выделенным объектам, которые обрабатывают их соответствующим образом. Это может привести к **замещению**, при котором новые данные заменяют собой выделенный объект. В других случаях выделенный объект может воспринимать поступающие данные как входную информацию для некоторой заданной функции. Например, в PowerPoint, если выделена фигура, ввод с клавиатуры означает создание текстовой аннотации к этой фигуре.

В то же время в непрерывном выделении поступающие данные всегда замещают выделенные. Когда вы набираете текст в редакторе или в поле ввода, вводимый текст заменяет выделенный. Непрерывное выделение обладает уникальной особенностью: оно может просто указы-

вать на место между двумя элементами непрерывных данных, а не на какой-то конкретный элемент. Это место называется **точкой вставки**.

В текстовом редакторе **знак вставки** (как правило, вертикальный мигающий отрезок, обозначающий, куда будет введен следующий символ) отмечает позицию между двумя символами в тексте, но не выделяет ни один из них. Поместив курсор мыши в другую точку текста и щелкнув, можно перенести знак вставки в эту точку, но если перетащить указатель мыши, расширяя выделение, то знак вставки исчезнет и заменится непрерывным выделением текста.

В электронных таблицах также применяется непрерывное выделение, но реализовано оно немного иначе, чем в текстовых редакторах. Выделение является непрерывным, поскольку ячейки таблицы образуют матрицу данных, однако понятие места между двумя ячейками отсутствует. Один щелчок выделяет одну ячейку целиком. В настоящее время понятие точки вставки для электронной таблицы отсутствует, однако открывающиеся здесь для проектировщика интерфейса возможности весьма интересны (например, выделив горизонтальную линию между ячейками, пользователь мог бы начать ввод с клавиатуры, что приводило бы к созданию новой строки и заполнению ячейки за одну операцию).

Вполне возможна комбинация этих двух идиом. Так, сортировщик слайдов[1] в PowerPoint позволяет выделять как точку вставки, так и отдельный слайд. Щелчок по слайду выделяет его, а щелчок между слайдами приводит к появлению между ними мигающего знака вставки.

Если программа поддерживает точку вставки, объекты должны выделяться щелчком с перетаскиванием. Чтобы выделить хотя бы один символ в текстовом редакторе, пользователь должен протащить указатель мыши по этому символу. В результате пользователь выполняет много щелчков и операций перетаскивания в процессе нормальной работы с программой. Побочный эффект этого – затрудняется передача любой идиомы перетаскивания. Это легко увидеть на примере редактора Word, где перетаскивание текста включает в себя первоначальный щелчок и перетаскивание указателя мыши для выделения фрагмента, перемещение указателя мыши внутрь выделенного фрагмента и последующий щелчок с перетаскиванием фрагмента на новое место. Чтобы выполнить аналогичную операцию в Excel, вам предстоит сначала найти специальную активную область (один-два пиксела шириной) на границе выделенной ячейки. Для перемещения дискретного выделения пользователь щелкает по объекту и перетаскивает его единым движением. Чтобы облегчить бремя перетаскивания при выделении, в текстовых редакторах часто предусмотрены альтернативные пути непосредственного манипулирования, например двойной щелчок, выделяющий целое слово.

---

[1] Представление документа, вызываемое в PowerPoint командой Сортировщик слайдов меню Вид. – *Примеч. ред.*

Визуальная индикация выделения

Выделенные объекты должны быть однозначно и отчетливо представлены пользователю как таковые. Выделенный объект должен бросаться в глаза на экране, заполненном другими объектами, выделение должно быть недвусмысленным и не скрывать детали объекта, видимые при его нормальном состоянии.

Выделение должно быть визуально отчетливым и недвусмысленным.

В частности, нужно сделать так, чтобы пользователи без труда отличали выделенные объекты от невыделенных. При этом недостаточно просто наделить объекты разными цветовыми характеристиками. Помните, что значительная часть населения страдает цветовой слепотой, так что цвет – не единственное, что должно характеризовать выделение.

Исторически сложилось так, что для передачи выделения используется инверсия (преобразование белых пикселов в черные, а черных в белые). Хотя это весьма заметный визуальный ход, он может не очень легко читаться, особенно в полноцветных интерфейсах. Среди других средств выделения – цветной фон, очертания объектов, псевдотрехмерное отображение, маркеры и анимированные контуры.

В приложениях для черчения, рисования, анимации и презентаций, где пользователи имеют дело с визуально насыщенными объектами, выделение легко потерять. Здесь лучшее решение – добавить к объекту индикаторы выделения, а не просто изменить его визуальные свойства. В большинстве графических редакторов реализуется именно такой подход с **маркерами** – небольшими прямоугольниками, окружающими объект и предоставляющими точки для воздействия на него.

Когда речь идет о выделениях сложной формы (как в приложении для обработки изображений вроде Adobe Photoshop), маркеры могут сбивать пользователя с толку и теряться на экране. Однако существует один способ гарантировать, что выделение всегда будет заметно вне зависимости от используемых цветов: использовать движение.

Одна из первых программ для компьютеров Macintosh, MacPaint, предложила замечательную идиому: выделенный объект очерчивался простым пунктиром, и фрагменты пунктирной линии синхронно двигались вокруг объекта. Пунктир напоминал колонну муравьев и потому получил красочное название: **марширующие муравьи**. Сегодня эта идиома обычно называется **область (marquee)**.[1]

---

[1] Термин *marquee* (фр.) является очень образным: этим словом назывались мерцающие вывески старых кинотеатров. В русской версии Adobe Photoshop эта идиома получила нейтральное название «область», ставшее стандартом де-факто. – *Примеч. науч. ред.*

В Adobe Photoshop данная идиома применяется для отображения областей выделения на фотографиях и работает чрезвычайно хорошо (опытные пользователи могут одним нажатием клавиши включать и выключать эффект, получая возможность анализировать работу без отвлекающих факторов). Анимацию несложно реализовать, хотя она и требует некоторой проработки, а эффект действует независимо от смешения цветов и насыщенности фона.

Перетаскивание

Из всех идиом непосредственного манипулирования ни одна не характеризует графический пользовательский интерфейс лучше, чем перетаскивание – щелчок по объекту, перемещение по экрану при нажатой клавише мыши и отпускание кнопки в точке назначения. Удивительно, но перетаскивание применяется не так часто, как нам кажется, и определенно еще не исчерпало свой потенциал.

В частности, популярность среды Всемирной паутины и миф о том, что веб-интерфейсы – это синоним исключительной простоты использования, сдерживают развитие идиомы перетаскивания в настольных приложениях, поскольку разработчики ошибочно копируют ущербный интерфейс веб-броузеров в другие, гораздо менее подходящие для него контексты. По счастью, веб-технологии совершенствуются, и сегодня программисты могут реализовать богатые возможности перетаскивания даже в броузере; и хотя задача пока еще не всегда тривиальна, похоже, что у мощных, выразительных командных идиом на всех платформах начинается вторая жизнь.

Перетаскивание можно определить как «щелчок по объекту и перенос его в другое место», хотя подобное определение узковато для такой широкой идиомы. Более точным определением перетаскивания будет такое: «щелчок по некоторому объекту и перенос его с целью преобразования».

Компьютеры Macintosh стали первой успешной системой, предложившей идиому перетаскивания. С этой идиомой были связаны большие ожидания, которые не воплотились в реальность по двум простым причинам. Во-первых, перетаскивание не было общесистемной функцией, а применялось только в программе Finder. Во-вторых, Macintosh в те времена был однозадачным компьютером, и идея перетаскивания объектов из одного приложения в другое еще много лет выпадала из поля зрения разработчиков.

К заслугам Apple следует отнести то, что она описала операцию перетаскивания в своем первом руководстве по пользовательским интерфейсам. Корпорация Microsoft, расположенная по другую сторону баррикад, не только не включила средства перетаскивания в ранние версии Windows, но даже не описала эту процедуру в документации для программистов. Впрочем, в конце концов Microsoft не только наверстала

упущенное, но и первой предложила новые применения для этой идиомы: плавающие инструментальные панели и паркуемые палитры.

Обычно термин «непосредственное манипулирование» используется для описания всех видов идиом взаимодействия с графическим пользовательским интерфейсом, но, когда речь заходит о перетаскивании, полезно выделить два уровня «непосредственности». Во-первых, есть настоящие идиомы непосредственного манипулирования, в которых перетаскивание означает помещение объекта куда-либо, например перемещение файла из одного каталога в другой, открытие файла в конкретном приложении (переносом пиктограммы файла на пиктограмму приложения) или упорядочение объектов на холсте в графическом редакторе.

Во-вторых, имеются косвенные идиомы перетаскивания: пользователь перетаскивает объект в определенную область или на другой объект, чтобы выполнить какую-то функцию. Такие идиомы менее популярны, но могут быть очень полезными. Хороший пример – автоматизация в Mac OS X (рис. 19.6)

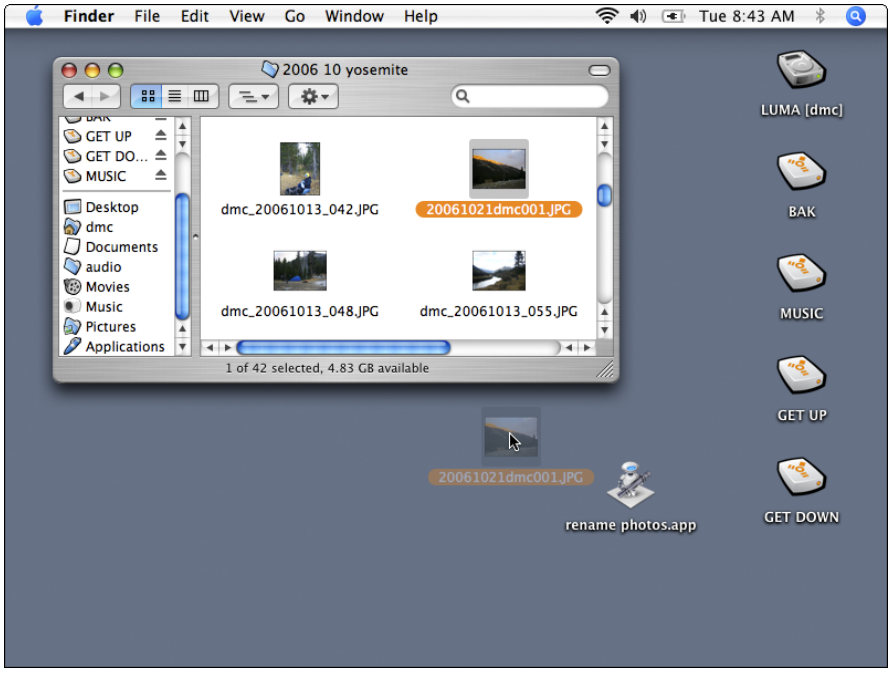


Рис. 19.6. *Инструмент Automator, предложенный Apple в рамках Mac OS X, позволяет пользователям описывать рабочие процессы (такие как переименование изображения), а затем представлять эти процессы в виде пиктограмм. Создав пиктограмму переименования, пользователь может перетаскивать на нее файлы и папки, чтобы выполнять переименование. И хотя такая функциональность, строго говоря, не является непосредственным манипулированием, она дает достаточно прямой способ воспользоваться командой*

Визуальная обратная связь процесса перетаскивания

Как уже говорилось, интерфейс должен визуально намекать на отзывчивость – либо статически, своим внешним видом, либо активно, реагируя на проходящий по нему курсор средствами анимации. Идея, что объекты можно перетаскивать, легко усваивается идиоматически. Единожды изучив поведение приложения, пользователь уже не забудет, что пиктограмма, выделенный текст или иной объект допускают непосредственное манипулирование. Однако он может забыть детали, и поэтому очень важно обеспечить обратную связь *после* начала перетаскивания. Новичку или пользователю, редко применяющему эту операцию, иногда требуется также дополнительная помощь (например, в виде текстовых подсказок, встроенных в интерфейс). Всепрощающие взаимодействия и функция отмены вдохновляют пользователей бесстрашно экспериментировать с идиомами непосредственного взаимодействия.

Как только пользователь щелкает кнопкой мыши по объекту, этот объект становится источником на всё время операции перетаскивания. Пока пользователь перемещает мышь, удерживая кнопку, курсор может проходить над множеством разнообразных объектов. Следует ясно давать понять, какие из этих объектов представляют собой осмысленные цели перетаскивания. Пока кнопка не отпущена, такие объекты являются **потенциальными целями**. В операции перетаскивания может быть только один источник и одна цель, но потенциальных целей может быть множество.

Единственная задача любой потенциальной цели состоит в визуальной индикации того факта, что горячая точка указателя мыши находится на ней. В свою очередь, это означает, что потенциальная цель примет к обработке объект-источник (или хотя бы адекватно отреагирует), если пользователь отпустит кнопку мыши. Подобная индикация по природе своей является активной визуальной подсказкой.

Потенциальные цели должны визуально демонстрировать свою готовность принимать объекты.

Самым ненадежным способом визуальной индикации готовности к обработке объекта-источника является изменение внешнего вида курсора. Первоочередная задача курсора – представлять перетаскиваемый объект, поэтому индикацию готовности принять объект-источник лучше оставить потенциальной цели.

Курсор при перетаскивании должен визуально ассоциироваться с объектом-источником.

Важно избегать путаницы в отношении этих двух визуальных функций. К сожалению, Microsoft допускает такую путаницу в Windows: здесь курсор почему-то используется для индикации объектов, которые *не* являются потенциальными целями. Такое решение, по-видимому, было принято для облегчения написания кода приложений, а не из каких-то соображений проектирования. Гораздо проще изменить форму курсора, чем сделать так, чтобы потенциальные цели отображали свою готовность принять объект. Однако задача указателя мыши – представлять объект-источник, а не потенциальную цель.

Будто нарочно пытаясь окончательно запутать ситуацию, Microsoft использует курсорную индикацию посредством зловещего перечеркнутого кружка – универсальное обозначение запрета. Данный символ – неприятная идиома, поскольку напоминает пользователям о том, чего они не могут делать. Это негативная обратная связь. Пользователь легко может истолковать этот знак как «Не отпускайте кнопку мыши, иначе случится нечто непоправимое» вместо «Если вы отпустите кнопку мыши, ничего не произойдет». Сочетание знака запрета с неуместной курсорной подсказкой – неудачная комбинация двух слабых идиом; ее следует избегать, что бы ни писала по этому поводу компания Microsoft в своих руководствах по стилю.

Когда пользователь в конце концов отпустит кнопку мыши, текущая потенциальная цель становится **целью**. Если же пользователь отпустит кнопку где-то между потенциальными целями или на недопустимой потенциальной цели, то никакой цели не будет и операция перетаскивания закончится безрезультатно. Отсутствие звукового сигнала или визуальной активности будет удачной индикацией такого завершения операции. Это не отмена операции, поэтому нет надобности выводить индикатор отмены.

Индикация возможности перетаскивания

Активная курсорная подсказка, указывающая на возможность перетаскивания, является плохим решением. Современная ориентация на объектно-ориентированный подход приводит к тому, что перетаскиваемых объектов становится больше, чем статичных. Непрерывное мелькание и смена формы курсора будет скорее отвлекать пользователя, чем помогать ему. Одно из возможных решений – исходить из предположения о том, что любые объекты можно перетаскивать, и предоставить пользователю возможность экспериментировать. Этот метод оправдывает себя в Проводнике (Windows Explorer) и в Finder (Macintosh). Без курсорных подсказок возможность перетаскивания может оказаться неочевидной, так что, возможно, есть смысл рассмотреть текстовые и всплывающие подсказки как альтернативные методы индикации.

Необходимо визуализировать момент, когда пользователь захватил объект-источник и операция перетаскивания началась. Самым визуально богатым методом является применение полноценной анимации

процесса перетаскивания, в реальном времени демонстрирующей перемещение объекта. Однако этот метод может оказаться сложным для реализации, неприятно замедлять работу и визуально усложнять интерфейс. Проблемой становится то, что операция перетаскивания требует довольно точного инструмента целеуказания. Допустим, объект-источник – квадрат со стороной 6 сантиметров, а цель – квадрат со стороной 1 сантиметр. Источник не должен перекрывать цель, но в данном примере он велик настолько, что может заслонить собой несколько таких потенциальных целей, и для точного указания целевого объекта, на который будет перенесен объект-источник, мы должны использовать горячую точку курсора мыши. На практике это означает, что перетаскивание прозрачного контура источника или миниатюры гораздо эффективнее анимированного перетаскивания точного изображения информационного объекта. Кроме того, перетаскиваемый объект не должен закрывать собой собственно курсор: острие стрелки курсора потребуется для точного указания горячей точки.

Перетаскивание контура подходит и для большинства операций изменения положения, поскольку контур при этом перемещается относительно объекта-источника, остающегося в исходном положении.

Индикация потенциальных целей

Путешествуя по экрану и перенося с собой контур объекта-источника, курсор проходит над разными потенциальными целями. Эти цели должны визуально демонстрировать свою готовность быть потенциальной целью. Своей визуальной реакцией потенциальная цель сообщает пользователю, что способна выполнить конструктивные действия с объектом, который будет на ней отпущен. (Разумеется, приложение должно быть достаточно сообразительным, чтобы выявлять осмысленные сочетания источников и целей.)

Имеется одно обстоятельство, столь очевидное, что о нем нередко забывают; оно состоит в том, что потенциальными целями могут быть только объекты, видимые пользователю. Работающему, но скрытому приложению нет нужды заботиться об индикации своей готовности быть целью. Как правило, на экране находится не так уж много объектов – десятка два, не больше. Следовательно, задача реализации проще, чем кажется.

Цели-вставки

В ряде приложений имеется возможность отпустить объект между другими объектами. Перетаскивание текста в редакторе Word относится как раз к такой операции, аналогичной операции упорядочивания в списках или массивах. В таких случаях на фоне, «позади» графических объектов приложения либо внутри непрерывных данных визуализируется особый тип визуальной подсказки; это и есть **цель-вставка**.

Хороший пример реализации такого перетаскивания дает сортировщик слайдов в PowerPoint. Пользователь может перетащить любой слайд в другую позицию внутри презентации. В ходе перетаскивания между слайдами появляется цель-вставка – вертикальная черная черта, похожая на увеличенный текстовый курсор. Редактор Word также демонстрирует цель-вставку при перетаскивании текста. Здесь пользователь видит не только «заряженный» данными указатель мыши, но и вертикальный серый курсор, отмечающий точное положение между символами, в которое будет помещен отпущенный фрагмент текста.

Если объекты можно перетаскивать и располагать между другими объектами, программа должна показывать цель-вставку. Как и в случае с потенциальной целью при перетаскивании типа «источник – цель», программа должна визуально обозначать, где можно отпустить перетаскиваемый объект.

Визуальная обратная связь, сигнализирующая о завершении операции

Если цель и источник сумели договориться, выполняется соответствующая операция. Важнейшим моментом на этой стадии является визуальная индикация факта выполнения операции. Если операция заключается в перемещении, объект должен исчезнуть из первоначального положения и появиться в конечном. Так, если мы перетаскиваем файл из одного каталога в другой, объект-источник должен исчезнуть с прежнего места и появиться на новом. Если цель представляет собой не контейнер для данных, а функцию (как в случае с пиктограммой печати), она должна показать, что приняла объект и обрабатывает его (выводит документ на печать). Это можно делать посредством анимации или иного изменения внешнего вида.

Прочие особенности перетаскивания

При первом знакомстве идиома перетаскивания кажется простой, однако при частом использовании и в некоторых ситуациях она может породить определенные сложности. Как это часто бывает, итеративный процесс разработки программных продуктов обнажил эти недостатки, и изобретательные проектировщики нашли ряд ловких решений.

Автоматическая прокрутка

Как должна реагировать программа, когда пользователь перетаскивает объект за пределы окна приложения? Разумеется, объект переносится на новое место, но где это место – внутри приложения или за его пределами?

Рассмотрим в качестве примера Microsoft Word. Что хочет сказать пользователь, перетаскивая выделенный фрагмент текста за пределы видимого в окне текста? «Я хочу перенести этот фрагмент в другую программу» или «Я хочу поместить его в другое место, но оно сейчас не

видно на экране»? В первом случае следует поступить так, как описано в предыдущих разделах. Если же пользователь имеет в виду второе, то приложение должно выполнить **автоматическую прокрутку** в направлении перетаскивания, чтобы пользователь мог отпустить выделенный фрагмент в другом месте документа, скрытом в настоящий момент.

Автоматическая прокрутка является очень важным дополнением к перетаскиванию. В тех случаях, когда цель может выходить за пределы экрана, программе нужна автоматическая прокрутка.

ПРИНЦИП проектирования

Любая цель перетаскивания, до которой можно добраться прокруткой, должна быть доступна посредством автоматической прокрутки.

В ранних реализациях автоматическая прокрутка выполнялась, когда пользователь перетаскивал объект за пределы окна приложения. Такой подход обладал двумя фатальными недостатками. Во-первых, как выйти за пределы окна, занимающего весь экран? Во-вторых, как приложение может отличить желание перенести объект в другое приложение от намерения вызвать автоматическую прокрутку?

Компания Microsoft нашла разумное решение этой проблемы. Автоматическая прокрутка начинается *внутри* окна приложения, а не *снаружи*. Автоматическая прокрутка в направлении перетаскивания начинается, когда курсор с перетаскиваемым объектом приближается к границам окна, подлежащего прокрутке, оставаясь при этом внутри окна. Если курсор мыши оказывается на расстоянии 30 пикселов от нижнего края области текста, Word прокручивает содержимое окна вверх. Если указатель подходит на такое же расстояние к верхней границе области текста, Word прокручивает содержимое вниз.

По счастью, недавно разработчики начали реализовывать автоматическую прокрутку с изменяемой скоростью (рис. 19.7): скорость прокрутки возрастает при приближении курсора к границе окна. К примеру, если курсор расположен на расстоянии 30 пикселов от верхней границы, текст прокручивается вниз со скоростью одна строка в секунду. На расстоянии 15 пикселов скорость возрастает до двух строк в секунду, и так далее. В результате пользователь может достаточно точно контролировать автоматическую прокрутку и применять ее в различных ситуациях.

Другой важной особенностью автоматической прокрутки является задержка во времени. Если автоматическая прокрутка начинается сразу, как только указатель мыши войдет в чувствительную зону у края окна, медлительный пользователь может вызвать автоматическую прокрутку непреднамеренно. Чтобы этого не произошло, следует начинать автоматическую прокрутку только после того, как указатель мыши задержится в зоне автоматической прокрутки на достаточно продолжительное время – около половины секунды.

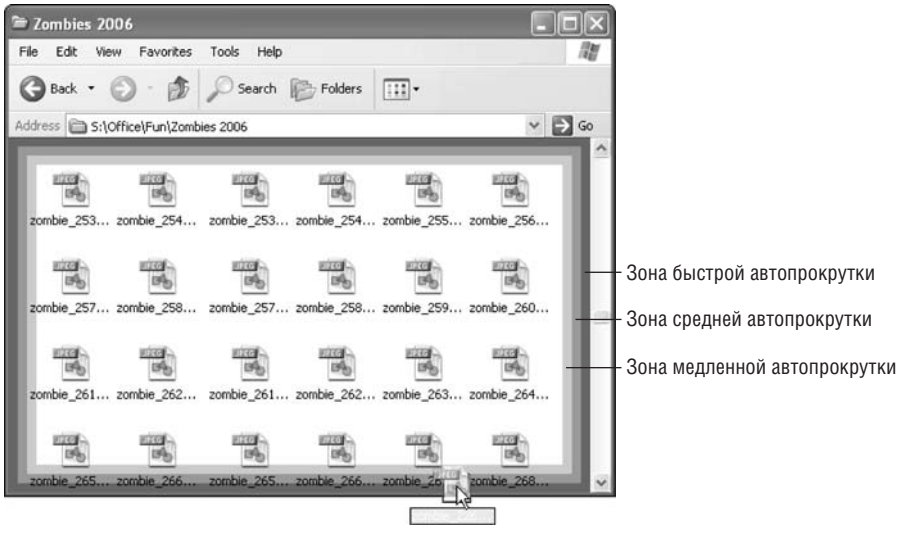


Рис. 19.7. *Это изображение демонстрирует вариант реализации автоматической прокрутки с изменяемой скоростью внутри приложения Проводник. К сожалению, в Windows XP автоматическая прокрутка работает с одной скоростью, которую невозможно изменять. Было бы лучше, если бы автоматическая прокрутка ускорялась при приближении курсора к границе окна (хотя ограничение скорости также важно сохранить – никому не будет пользы от слишком быстрой прокрутки). Однако отдадим должное Microsoft: мысль о том, что прокрутку следует включать при приближении курсора к границе внутри окна, а не при ее пересечении, оказалась правильной*

Если пользователь перетащит курсор целиком за пределы прокручиваемого окна редактора Word, никакой автоматической прокрутки не будет. Вместо нее будет выполнена операция переноса, которая закончится в другой программе. Например, если курсор с перетаскиванием перейдет из окна Word в окно PowerPoint и пользователь отпустит кнопку мыши, то выделенный фрагмент текста будет вставлен в слайд PowerPoint в ту позицию, на которую укажет мышь. Более того, если курсор остановится в трех или четырех миллиметрах от любой границы окна PowerPoint, PowerPoint начнет автоматическую прокрутку в соответствующем направлении. Это очень удобная функциональная возможность, поскольку из-за ограничений, накладываемых современными экранами, мы нередко обнаруживаем, что объект, который мы уже принялись перетаскивать, попросту негде отпустить.

Борьба с дребезгом при перетаскивании

Если объект можно как выделить, так и перетащить, очень важно, чтобы курсор мыши отдавал предпочтение операции выделения.

Крайне трудно щелкнуть по объекту, не сдвинув при этом указатель мыши на пару пикселов, поэтому часто выполняемая операция выделения объекта не должна ошибочно приниматься за начало операции перетаскивания. Пользователи редко пытаются передвинуть объект на два пиксела. (А если в некоторых случаях и пытаются, например в приложениях для рисования, полезно сделать порог смещения более высоким, чтобы предотвратить случайное перемещение.)

В аппаратуре применяются кнопки, содержащие механические контакты, в которых часто проявляется так называемый эффект **дребезга**, обусловленный тем, что тонкие контактные пластинки вибрируют, когда на них кто-то нажимает. В электрических устройствах, вроде дверного звонка, такая вибрация, длящаяся миллисекунды, несущественна, но в микроэлектронике подобное явление, приводящее к повторным щелчкам, может играть заметную роль. В устройствах, включающих в себя механические переключатели, реализована специальная логика, игнорирующая «лишние» переключения, если они имеют место в пределах нескольких миллисекунд после первого. Вот почему ваша стереосистема, например, не отключается через тысячную долю секунды после того, как вы ее включили. Описанная ситуация возникает и в сверхчувствительной схеме работы мыши. Решение, как и в случае переключателей, состоит в **компенсации дребезга**.

Чтобы избежать проблем, программа должна устанавливать **порог смещения**, чтобы все системные сообщения о перемещении мыши, поступившие после сообщения о нажатии на кнопку, игнорировались, если перемещение происходит в небольших пределах, например три пиксела. Тем самым обеспечивается некоторая защита от случайного перетаскивания. Если пользователь в состоянии удержать мышь в пределах трех пикселов в момент щелчка, щелчок интерпретируется как команда выделения, а все мелкие паразитные перемещения игнорируются. Так достигается компенсация дребезга. Если же мышь выйдет за пределы трехпиксельного порога, программа сможет уверенно приступить к операции перетаскивания. Это проиллюстрировано на рис. 19.8. Всегда, когда объект допускает как выделение, так и перетаскивание, необходима компенсация дребезга.

ПРИНЦИП проектирования

Компенсируйте дребезг перетаскивания.

В некоторых приложениях может потребоваться более сложная реализация порога смещения. В программах трехмерной графики часто необходимы пороги смещения по экрану вдоль сразу трех координатных осей в проекции. Сходная ситуация возникла однажды при разработке генератора отчетов для одного из наших клиентов. Пользователь имел возможность сортировать столбцы в отчете, перетаскивая их по горизонтали. Например, он мог поставить столбец с именами слева от

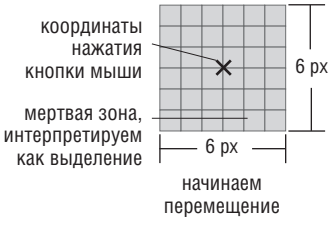


Рис. 19.8. *Для любого объекта, допускающего и выделение, и перетаскивание, следует компенсировать дребезг. Когда пользователь щелкает по объекту, это действие следует интерпретировать как выделение, а не перетаскивание, даже если пользователь случайно сдвинет мышь на пару пикселов, пока кнопка еще нажата. Программа должна игнорировать всякое движение в пределах мертвой зоны, простирающейся на три пиксела в каждую сторону от точки щелчка. Если указатель мыши сдвинется более чем на три пиксела от точки щелчка, объект должен считаться «сдвинутым» и операция выделения должна смениться перетаскиванием. Это называется порогом смещения*

столбца с фамилиями, ухватившись мышью за любое место столбца. Это была наиболее востребованная идиома перетаскивания. Была, однако, и другая операция перетаскивания, которая применялась намного реже. Она позволяла объединять значения в одном столбце со значениями в соответствующих строках другого столбца. Например, можно было объединить столбец адреса со столбцом, в котором указывался штат (рис. 19.9).

Мы хотели следовать ментальной модели персонажа и позволить ему перетаскивать значения одного столбца на значения другого, но это конфликтовало с обычным переупорядочиванием столбцов. Мы решили проблему, разделив операции перетаскивания на горизонтальные и вертикальные. Если пользователь перетаскивал столбец вправо или влево, это означало перемещение столбца как целого. Когда пользователь перетаскивал столбец вверх или вниз, это означало его намерение объединить значения ячеек этого столбца со значениями другого.

Поскольку горизонтальное перетаскивание было преобладающей операцией, а вертикальное выполнялось сравнительно редко, мы настроили порог смещения для предпочтения горизонтальных движений. Вместо квадратной мертвой зоны мы создали зону в форме катушки, изображенную на рис. 19.10. Благодаря четырехпиксельному порогу горизонтального смещения от пользователя не требовалось широких движений, чтобы началось обычное перетаскивание по горизонтали, а случайное перетаскивание по вертикали компенсировалось. Чтобы выполнить гораздо более редкую операцию вертикального перетаскивания, пользователь должен был сдвинуть курсор на восемь пикселов вдоль вертикальной оси, не отклонившись в сторону более чем на четыре пиксела. Такое движение оказалось вполне естественным и легко запоминаемым.

Name	Address	City
1 Ginger Beef	342 Easton Lane	Waltham
2 C. U. Lator	339 Disk Drive	Borham
3 Justin Case	68 Elm	Albion
4 Creighton Barrel	9348 N. Blenheim	Five Island
5 Dewey Decimal	1003 Water St.	Freeport

Name	Address/City
1 Ginger Beef	342 Easton Lane Waltham
2 C. U. Lator	339 Disk Drive Borham
3 Justin Case	68 Elm Albion
4 Creighton Barrel	9348 N. Blenheim Five Island
5 Dewey Decimal	1003 Water St. Freeport

Рис. 19.9. *Этот генератор отчетов предлагал пользователю интересную возможность слияния значений в двух столбцах путем перетаскивания одного из них на другой. Такая операция непосредственного манипулирования конфликтовала с другой, более частой операцией переупорядочивания столбцов перетаскиванием (например, перемещением столбца City влево от столбца Address). Мы применили специальный двухмерный порог смещения для разрешения конфликта*

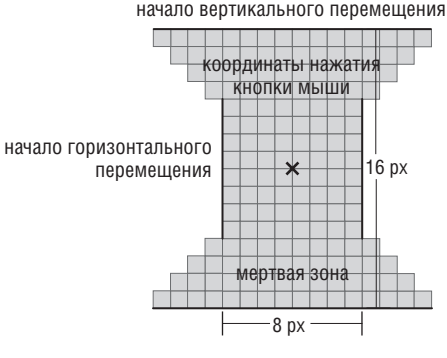


Рис. 19.10. *Этот порог смещения в форме катушки позволил реализовать в программе предпочтение горизонтального перетаскивания вертикальному. В данном приложении горизонтальное перетаскивание применялось чаще, чем вертикальное. Такой порог смещения препятствовал непреднамеренному перемещению по вертикали. Однако если пользователь действительно хотел выполнить вертикальное перетаскивание, решительное движение по вертикали заставляло программу переключиться в соответствующий режим и не отягощать пользователя лишними действиями*

Такой асимметричный по осям порог может использоваться и в других целях. В программе Visio реализована сходная идиома, позволяющая отличить рисование прямой линии от рисования кривой.

Точная прокрутка

Слабая пригодность мыши в качестве устройства точного указания очевидна и особенно ярко проявляется при перетаскивании объектов в программах рисования. Ужасно трудно перетащить объект точно в нужную позицию, особенно учитывая, что разрешение экрана – 72 пиксела на дюйм, а перемещения мыши масштабируются в соотношении шесть к одному. Чтобы переместить указатель мыши на один пиксел, вы должны сдвинуть ее саму на одну пятисотую дюйма. Это нелегко.

Проблема решается добавлением функции **точной прокрутки**, позволяющей пользователю изменять соотношение между перемещением мыши и курсора на экране. Если в ходе операции перетаскивания пользователю понадобилось более точное маневрирование, он включает режим точной прокрутки. Любая программа, в которой может потребоваться точное выравнивание, должна предлагать пользователю возможность точной прокрутки. Это касается в первую очередь всех программ для черчения и рисования, презентационных программ и программ манипулирования изображениями.

Любая программа, в которой требуется точное выравнивание, должна предлагать пользователю верньер.

Существует несколько вариантов этой идиомы. Обычно мышь переводится в режим верньера удержанием служебной клавиши во время перетаскивания. В этом режиме перемещению мыши на десять пикселов будет соответствовать перемещение объекта на один пиксел.

Другим эффективным методом является использование во время перетаскивания клавиш со стрелками. Удерживая кнопку мыши нажатой, пользователь может передвигать выделенный объект клавишами со стрелками на один пиксел вверх, вниз, влево или вправо. Как и всегда, перетаскивание заканчивается, когда пользователь отпускает кнопку мыши.

Проблема с такой реализацией верньера состоит в том, что в момент высвобождения кнопки рука пользователя может дрогнуть и сдвинуть мышь на один-два пиксела – и столь тщательно установленный объект сместится в самый последний момент. Справиться с этой проблемой можно, **лишив мышь чувствительности** после первого нажатия на клавишу в режиме верньера. Это реализуется путем игнорирования всех перемещений мыши в пределах, установленных неким порогом, равным, скажем, пяти пикселам. В конечном итоге получается, что пользователь может начать с размашистых движений мышью, потом выполнить окончательную подгонку позиции с помощью клавиш со

стрелками, а затем отпустить кнопку мыши без риска сдвинуть объект на экране. Если уже после переключения в режим верньера пользователю понадобится снова перейти к размашистым движениям, он просто выводит мышь за пределы зоны, установленной порогом, тем самым отключая режим.

Если, как принято в графических редакторах, клавиши со стрелками не используются, их можно задействовать для точного позиционирования выделенного объекта. Это означает, что пользователю не придется держать кнопку мыши нажатой. Такой подход реализован в Adobe Illustrator, Photoshop и PowerPoint. В последнем приложении клавиши со стрелками передвигают выделенный объект на один шаг сетки, по умолчанию равный примерно двум миллиметрам. Если удерживать клавишу <Alt>, то каждое нажатие на клавишу со стрелкой будет перемещать объект на один пиксел.

Манипулирование элементами управления

Элементы управления – базовые строительные блоки современных графических пользовательских интерфейсов. Мы подробно обсудим их в главе 21, а в разговоре о непосредственном манипулировании следует обсудить взаимодействия с мышью, необходимые для работы с рядом элементов управления.

Многие элементы управления, в частности меню, требуют от пользователя выполнения довольно сложной операции перетаскивания вместо простого щелчка. Эта операция непосредственного манипулирования создает дополнительную нагрузку на пользователя из-за чередования движений, требующих грубой и точной моторики, – щелчка, перетаскивания и отпускания кнопки. Хотя меню используются не так часто, как панели инструментов, они все же популярны, особенно у новичков или людей, редко пользующихся продуктом. И здесь мы обнаруживаем одну из самых непостижимых загадок проектирования графических пользовательских интерфейсов: меню используется преимущественно новичками, но при этом в физическом плане является одним из самых сложных элементов управления.

У этой проблемы есть лишь одно решение, которое состоит во введении дополнительных идиом для выполнения тех же операций. Если функция доступна через меню и будет использоваться относительно часто, обязательно предоставьте пользователю другие идиомы для ее вызова, которые не будут включать в себя перетаскивание, такие как кнопки на панели инструментов.

Одной из приятных особенностей Windows, позаимствованных последними версиями Mac OS, является возможность работы с меню посредством серий одиночных щелчков, а не щелчков с перетаскиванием. Вы щелкаете по меню – и оно раскрывается. Вы наводите курсор на нужный пункт меню и щелкаете. Пункт выделяется, а меню закры-

вается. Microsoft развила эту идею и реализовала **режим меню,** который включается после щелчка по любому меню. В этом режиме все меню верхнего уровня и все их элементы активны, словно вы выполняете щелчок и перетаскивание. Когда вы проводите по ним указателем мыши, все меню поочередно раскрываются, и для этого не требуется нажимать на кнопки мыши вообще. Это явление может привести в замешательство, если вы не знакомы с ним, но, когда первый шок пройдет, такое поведение покажется более удобным, поскольку оно меньше нагружает кисть.

Инструменты палитры

Во многих приложениях для черчения и рисования после выбора пользователем инструмента из палитры курсор видоизменяется и при щелчке с перетаскиванием выполняются функции, специфические для этого инструмента. Инструменты обладают собственным уникальным поведением, которое требует отдельного обсуждения. С точки зрения поведения инструменты палитры можно разбить на два класса: модальные инструменты и инструменты с нагруженным курсором.

Модальные инструменты

Модальные инструменты действуют следующим образом. Пользователь выбирает инструмент из списка или на специализированной панели инструментов, обычно называемой палитрой. Рабочая область окна программы целиком переходит в режим работы с выбранным инструментом: программа выполняет только действия, связанные с ним. Курсор при этом обычно видоизменяется, отображая активный инструмент.

Когда пользователь щелкает и выполняет перетаскивание в рабочей области, инструмент выполняет свою работу. Если это, например, аэрограф, программа переходит в режим рисования аэрографом и позволяет лишь распылять краску. Инструментом можно пользоваться столько, сколько потребуется, пока не понадобится другой инструмент. Если пользователь захочет применить, например, ластик, то ему придется вернуться к панели инструментов и выбрать там ластик. Программа перейдет в режим работы ластиком и будет способна только стирать нарисованное, пока пользователь снова не сменит инструмент. Как правило, палитра предлагает среди прочего инструмент выделения, что позволяет пользователю вернуться к указателю мыши, способному выполнять выделение (так, например, обстоят дела в Adobe Photoshop).

Модальные инструменты хороши как средство выполнения **действий** с рисунком (например, стирание ластиком), а также в качестве средств размещения **фигур** (например, эллипсов). Курсор может превратиться в ластик и стирать то, что было нарисовано, а может превратиться в эл-

липс и создать столько эллипсов, сколько требуется. Событие «кнопка мыши нажата» фиксирует угол или центр будущей фигуры (или ее ограничивающего прямоугольника), пользователь выполняет перетаскивание, чтобы придать фигуре нужный размер и пропорции, а событие «кнопка мыши отпущена» подтверждает, что рисование закончено.

Модальные инструменты не создают проблем в программе вроде Paint, где выбор инструментов рисования невелик. Зато в более сложных графических приложениях, таких как Adobe Photoshop, модальность становится разрушительной. По мере того как пользователь совершенствует свои навыки обращения с инструментами, процент времени, уходящего на выбор того или иного инструмента (то есть интерфейсные налоги), возрастает весьма заметно. Модальные инструменты являются прекрасными идиомами для предъявления новичку функциональных возможностей программы, но они не способны удовлетворить запросы более опытных пользователей сложных программ. К счастью, Photoshop предоставляет искушенным пользователям богатый набор клавиатурных команд.

Трудность управления приложением, предлагающим обилие модальных инструментов, вызвана не столько модальностью как таковой, сколько количеством инструментов. Иными словами, эффективность падает, когда количество предлагаемых пользователю инструментов становится слишком большим. Рабочим комплектом из пяти и более модальных инструментов, как правило, трудно управлять. Если бы количество инструментов Adobe Illustrator можно было сократить с двадцати четырех до восьми, проблемы пользовательского интерфейса этой программы перестали бы вызывать у пользователей головную боль.

Чтобы компенсировать изобилие модальных инструментов, продукты вроде Adobe Illustrator предлагают специальные служебные клавиши для оперативного переключения режимов. Для ограничения направления перетаскивания широко применяется клавиша <Shift>, но Adobe Illustrator помимо этого вводит много нестандартных служебных клавиш и предлагает использовать их нестандартным образом. Например, если удерживать нажатой клавишу <Alt> при перетаскивании объекта, перетаскиваться будет *копия* этого объекта. При этом клавиша <Alt> применяется также и для перевода выделяющего инструмента из режима выделения узлов кривых в режим выделения объектов целиком. Различие в действиях пользователя между этими способами использования клавиш довольно трудноуловимое: если вы щелкаете по объекту и затем нажимаете клавишу <Alt>, вы будете перетаскивать копию объекта; если же вы нажмете клавишу <Alt> и *после этого* щелкнете по объекту, выделится весь объект, а не одна его вершина. А потом, чтобы окончательно вас запутать, приложение требует, чтобы вы *отпустили* клавишу <Alt>, иначе будет выполнено перетаскивание копии. Выполнение такого простого действия, как выделение всего объекта и перетаскивание его на новое место, превращается в замысловатую процедуру: вы нажимаете клавишу <Alt>, наводите курсор на

объект, щелкаете по нему, но движение не начинаете, затем отпускаете клавишу <Alt> и перетаскиваете объект в другое место!.. О чем думали разработчики этого интерфейса?

Предполагается, что сочетания клавиш добавляют мощности интерфейсу, но их трудно выучить, трудно держать в голове и трудно применять. Если вы профессиональный дизайнер, работающий с Illustrator по восемь часов в день, вы сумеете обратить эти недостатки в достоинства, подобно тому, как профессиональный гонщик обращает взрывной характер заряженного автомобиля в преимущество на гоночной трассе. Однако пользователь, работающий с Illustrator от случая к случаю, будет чувствовать себя как автолюбитель за рулем гоночного автомобиля, темпераментного и не подходящего для обычной поездки.

Инструменты с нагруженным курсором

Когда речь идет об инструментах с **нагруженным курсором**, пользователь точно так же выбирает инструмент или фигуру из палитры, но на этот раз курсор не переключается в постоянный режим (до следующего переключения) работы с этим инструментом, а **нагружается** экземпляром выбранного объекта.

Щелкнув один раз внутри области рисования, пользователь создаст в точке щелчка экземпляр объекта. Нагруженный указатель плохо подходит для выполнения функций (хотя Microsoft повсюду использует его для функции копирования формата), но отлично работает с графическими объектами. В частности, он широко применяется в приложении PowerPoint. Пользователь выбирает прямоугольник в графической палитре – и курсор превращается в модальный инструмент, нагруженный ровно одним прямоугольником.

Во многих программах с нагруженными курсорами, например в PowerPoint, пользователь не всегда может поместить объект простым щелчком. Ему приходится перетаскиванием растягивать ограничивающий прямоугольник, чтобы задать размер будущего объекта. В некоторых приложениях, например Visual Basic, работают оба метода. Одиночный щелчок при нагруженном курсоре создает один экземпляр объекта с размером по умолчанию. Новый экземпляр находится в выделенном состоянии и окружен **маркерами** (которые мы обсудим в следующем разделе), готовый к немедленному изменению размера и формы. Такой двойной режим, включающий в себя либо одиночный щелчок для размещения объекта с размером по умолчанию, либо растягивание ограничивающего прямоугольника для размещения объекта с произвольным размером, безусловно, является самым гибким и очевидным и понравится большинству пользователей.

Иногда программы с нагруженным курсором забывают изменить его внешний вид. Например, Visual Basic превращает указатель в перекрестие, когда он нагружен, а Delphi никак не меняет его. Когда курсор переходит в модальный режим, то есть когда щелчок приводит

к созданию объекта, очень важно визуально отражать это состояние. К нагруженному указателю нужны также хорошие идиомы отмены, иначе как вы разгрузите курсор, ничего не повредив?

Манипулирование объектами

Как и элементами управления, объектами данных на экране (особенно графическими объектами в программах рисования и моделирования) можно манипулировать путем щелчка с перетаскиванием. Объекты (не пиктограммы, которые мы обсуждали ранее в этой главе) подвергаются перетаскиванию при выполнении трех основных операций: перемещение, изменение размера и изменение формы.

Перемещение

Перемещение – это простое действие, состоящее из щелчка по объекту и перетаскивания его в другое место. Наибольшие сложности при реализации перемещения вызваны тем, что оно узурпирует место других идиом непосредственного манипулирования: функция перемещения требует, чтобы пользователь выполнил щелчок и перетаскивание, и тем самым делает перетаскивание недоступным для других применений.

Самым распространенным способом разрешения этого конфликта является отведение под функцию перемещения определенной области объекта. Например, в Windows или Macintosh вы можете переместить окно, ухватившись за его заголовок. Остальная часть окна не является активной с точки зрения задачи перемещения, и идиома перетаскивания остается доступной внутри окна, как того и ожидает пользователь. Единственным намеком на то, что окно можно перемещать, является цвет его заголовка и некоторая трехмерность. Эта неброская подсказка является чистейшей воды идиомой (к счастью, весьма эффективной).

В общем случае следует обеспечивать более явные визуальные подсказки об отзывчивости некоторой области интерфейса. Для заголовка окна можно было бы использовать курсорную подсказку или заливку заголовка рельефной текстурой, которая говорит о его «перетаскиваемости».

Чтобы переместить объект, сначала нужно его выделить. Вот почему выделение должно происходить по событию «кнопка мыши нажата». В этом случае пользователь сможет сразу выполнить перетаскивание, то есть ему не придется сначала отпускать кнопку мыши, чтобы выделить объект, а затем снова нажимать, чтобы начать движение, – для перемещения объекта достаточно будет просто нажать и потянуть мышь. Щелкнуть по объекту и тут же перетащить его легким движением руки – гораздо более естественное движение.

Однако это создает проблемы с перемещением непрерывных данных. Например, в редакторе Word компания Microsoft реализовала неуклюжую операцию «щелчок – ожидание – щелчок» для перетаскивания фрагментов текста: вы должны сперва щелкнуть и перетащить указатель мыши, чтобы выделить фрагмент, затем подождать примерно секунду и снова щелкнуть – и тогда только приступать к перетаскиванию выделенного текста. Решение неудачное, однако для непрерывных данных хорошей альтернативы нет. Если бы разработчики из Microsoft согласились обойтись без использования служебных клавиш для расширения выделения, эти клавиши можно бы задействовать для выделения и переноса предложения в одно движение. Впрочем, это не решило бы проблему выделения и перемещения произвольных фрагментов текста.

Для ограничения перетаскивания одной осью (горизонтальной либо вертикальной) часто используется служебная клавиша (скажем, <Shift>). Такого типа перетаскивание называется **направляемым**. Направляемое перетаскивание очень полезно в приложениях для графики, особенно при рисовании аккуратных диаграмм. Первые пять-десять пикселов пути курсора определяют преобладающее направление перетаскивания. Если пользователь начинает перетаскивание в большей степени по горизонтали, то в дальнейшем оно будет ограничено именно горизонтальной осью. Некоторые приложения позволяют пользователю менять решение (и направление перетаскивания) в середине пути, отклоняя курсор на определенное расстояние.

Другой способ помочь пользователям при перемещении объектов на экране – **направляющие**. В наиболее популярном варианте (например, Adobe Illustrator) это специальные линии, которые пользователь может применять в качестве вспомогательных при размещении объектов. Как правило, приложению можно предписать выполнять «привязку» к направляющим: когда в режиме привязки объект оказывается в пределах определенного расстояния от направляющей линии, приложение считает, что объект следует выровнять по этой направляющей. Скорректировать такое перемещение, как правило, можно с клавиатуры.

Новаторская и полезная вариация этого подхода – интеллектуальные направляющие (Smart Guides), созданные OmniGraffle. Они обеспечивают визуальную обратную связь и помогают располагать объект исходя из (весьма разумного) предположения о том, что пользователям требуется выравнивать объекты, создавая колонки и ряды с равномерными интервалами. Продукт Google SketchUp (более подробно описываемый далее) сходным образом помогает пользователю с трехмерными эскизами.

Изменение размера и формы

Когда речь идет об окнах графического пользовательского интерфейса, реальной функциональной разницы между изменением размера и изменением формы нет. Пользователь может в один прием скорректировать размеры окна и соотношение сторон – достаточно перетащить элемент управления, расположенный в правом нижнем углу окна. Можно также потянуть за любую границу окна. Взаимодействия такого рода обычно поддерживаются очевидными курсорными подсказками.

Описанные идиомы хороши для изменения размеров окна, но в случае с графическим объектом в программе рисования или моделирования важно сообщить пользователю о том, какой объект выделен и где следует щелкнуть, чтобы изменить размер или форму этого объекта. Идиома изменения размеров графического объекта должна визуально выделяться среди элементов рисунка и отличаться от элементов объекта, к которому она относится. Кроме того, она не должна заслонять сам объект и область вокруг него. Наконец, элемент управления, ответственный за изменение размеров, не должен мешать выполнению этой операции.

Есть популярная идиома, удовлетворяющая всем этим требованиям. Она состоит из восьми маленьких черных квадратиков, расположенных по углам прямоугольного объекта и в центре каждой его стороны. Эти квадратики, изображенные на рис. 19.11, называются **маркерами размера**, или просто **маркерами**.

Маркеры полезны и проектировщикам, поскольку позволяют обозначать выделение. Это возникающий естественным путем симбиоз, поскольку объект обычно должен быть выделен перед изменением его размера.

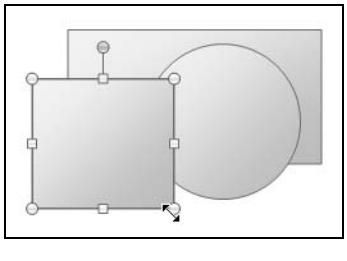


Рис. 19.11. *У выделенного объекта есть восемь маркеров: в каждом углу и в центре каждой стороны. Маркеры обозначают выделение и являются удобной идиомой изменения размеров и формы объекта. Иногда маркеры реализуются с инверсией цвета пикселов, но в многоцветном окружении они могут затеряться в «шуме». Изображенные на рисунке маркеры Microsoft PowerPoint 2007 обладают некоторым объемом, который контрастирует со слайдом, облегчая восприятие*

Маркер, расположенный в центре стороны, двигает только эту сторону и оставляет неподвижными прочие. Маркеры в углах перемещают обе прилегающие стороны, что вполне ожидаемо с позиций интуиции.

Маркеры часто заслоняют объект, к которому относятся, и потому не очень удобны в качестве постоянно присутствующих элементов управления. Вот почему мы не видим их на окнах верхнего уровня (впрочем, окна в некоторых версиях интерфейса Open Look от Sun имеют нечто подобное). В таких ситуациях для изменения размеров более уместна идиома рамки или «уголка». Если выделенный объект не помещается на экране, маркеры могут оказаться за пределами видимости. Находясь за пределами экрана, они не только недоступны для непосредственного манипулирования, но и бесполезны в качестве индикаторов выделения.

Как и в случае перетаскивания, процесс изменения размера часто направляют с помощью служебных клавиш. Это еще один пример идиомы направляемого перетаскивания: клавиша <Shift> дает возможность сохранить соотношение сторон объекта при изменении размеров. Такая возможность весьма удобна. В некоторых случаях бывает полезно ограничивать изменение размера вертикалью, горизонталью либо определенным отношением сторон.

До сих пор в нашем разговоре о маркерах предполагалось, что маркируемый объект является прямоугольником или легко вписывается в прямоугольник. Хорошо, если пользователь рисует организационную диаграмму, а что делать с изменением формы более сложных объектов? Существует мощная и полезная разновидность маркера – **вершинный маркер**.

Во многих программах объекты на экране изображаются с помощью **полилиний**. **Полилиния** – термин, употребляемый в программировании графики для обозначения ломаной линии, заданной массивом узлов (вершин). Если первая и последняя вершины совпадают, полилиния является замкнутой и образует многоугольник. Когда такой объект выделяется, программа помещает маркер в каждую вершину полилинии (а не в восемь точек описанного вокруг нее прямоугольника, как в случае с простыми объектами). Пользователь может перетаскивать любую вершину полилинии независимо от других и изменять форму объекта локально, не затрагивая объект в целом. Это проиллюстрировано на рис. 19.12.

Объекты произвольной формы в программе PowerPoint изображаются с помощью полилиний. Если щелкнуть по такому объекту, появится ограничивающий многоугольник с восемью стандартными маркерами. Если же щелкнуть по объекту правой кнопкой мыши и выбрать из контекстного меню пункт Начать изменение узлов (Edit Points), то ограничивающий прямоугольник исчезнет – и вместо него появятся вершинные маркеры.Важно, что доступны обе идиомы: первая необходима для пропорционального масштабирования объекта, а вторая – для точного редактирования его формы.

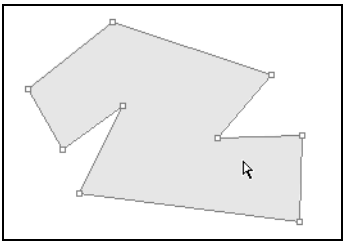


Рис. 19.12. *Вершинные маркеры обязаны своим названием тому, что у каждой вершины многоугольника имеется один маркер. Пользователь может щелкнуть по любому маркеру и перетащить его, изменяя форму многоугольника в рамках одного сегмента. Эта идиома полезна главным образом для графических приложений*

Если рассматриваемый объект состоит из кривых, а не прямых линий, оптимальным механизмом изменения размеров являются маркеры Безье. Подобно вершине полилинии маркер Безье отмечает точку объекта, но также еще и форму кривой, проходящей через точку. Эффективное управление кривыми Безье требует значительных навыков, и такую функциональность лучше оставить для специализированных приложений рисования и моделирования.

Манипулирование трехмерными объектами

Точная работа с трехмерными объектами представляет значительные трудности для пользователей, имеющих дело с двухмерными устройствами ввода и обычными мониторами. Наиболее интересные исследования в области проектирования пользовательского интерфейса связаны с разработкой более удачных парадигм трехмерного ввода и манипулирования. Однако пока что никаких революций не произошло, и наблюдается лишь расширение двухмерных идиом на мир трехмерных объектов.

Большинство приложений трехмерной графики связаны либо с изготовлением точных чертежей (например, архитектурные приложения), либо с трехмерной анимацией. При создании трехмерных моделей анимационные программы сталкиваются с проблемами, аналогичными проблемам чертежных приложений. Однако анимация переводит нас на более высокий уровень сложности, когда дело доходит до движения и динамического изменения этих моделей. Часто аниматоры создают модели в специализированных приложениях и лишь затем загружают модели в анимационные приложения.

Идиомы трехмерного манипулирования являются настолько обширной темой, что о них можно написать отдельную главу или даже целую книгу. Поэтому мы лишь кратко затронем самые важные вопросы данной темы.

Проблемы отображения и идиомы

Возможно, самой значительной проблемой трехмерного взаимодействия на плоском экране является нехватка параллакса – возможности воспринимать глубину бинокулярно. Если не брать в расчет дорогие и экзотические периферийные устройства в виде специальных очков, у проектировщиков практически нет средств, чтобы справиться с этой трудностью. Другой важной проблемой является видимость: ближние объекты перекрывают дальние. Эти навигационные проблемы, а также некоторые вопросы ввода, обсуждаемые в следующем разделе, по-видимому, являются основной причиной того, что виртуальная реальность еще не стала пользовательским интерфейсом нового поколения.

Множественные ракурсы

Использование **множественных ракурсов** является, пожалуй, самым старым методом решения этих двух проблем, но этот метод во многих отношениях неэффективен с точки зрения взаимодействия. Тем не менее большинство приложений трехмерной графики представляют на экране несколько видов одного объекта, показывая его под разными углами. Как правило, предлагаются виды сверху, спереди и сбоку, причем они привязаны к абсолютным осям и могут масштабироваться. Часто присутствует и четвертый вид – ортографическая либо перспективная проекция сцены, – параметры которого могут уточняться пользователем. Когда эти виды представлены в отдельных окнах, каждый в своей рамке и со своими элементами управления, идиома становится весьма неповоротливой: окна то и дело перекрываются, мешают друг другу, а ценное экранное пространство расходуется понапрасну на повторяющиеся элементы управления и рамки окон. Более удачным решением является одно окно с несколькими панелями. Оно допускает одно-, двух-, трех- и четырехпанельные представления (трехпанельное включает в себя одну большую панель и две маленьких). Выбор конфигурации должен в идеале выполняться одним щелчком по панели инструментов или с помощью комбинации клавиш.

Недостаток множественных ракурсов состоит в том, что пользователю приходится одновременно отслеживать несколько областей экрана, чтобы определить расположение объекта. Когда вы, заставляя пользователя в поисках предмета рассматривать сложную сцену сверху, сбоку и спереди, предполагаете, что он сумеет мысленно собрать эти представления воедино в реальном времени, вы слишком многого от него требуете (даже если допустить, что он является экспертом в области моделирования). Тем не менее множественные ракурсы все же оказываются полезными, когда нужно выровнять объекты по каким-либо осям.

Координатные сетки, индикация глубины, тени и нормали

Координатные сетки, индикация глубины, тени и нормали – это те идиомы, которые помогают обходить проблемы, возникающие в связи с множественными ракурсами. За всеми ними стоит одна и та же идея –

дать пользователю ощущение местоположения и перемещения объектов в трехмерной сцене, представленной в ортографической или перспективной проекции.

Координатные сетки создают в сцене виртуальный пол и стены (по одной плоскости на каждую ось), которые помогают пользователю ориентироваться. Они особенно полезны, когда есть возможность свободно поворачивать камеру (а такая возможность, как правило, есть).

Индикация глубины – это средства, благодаря которым предметы, расположенные в поле зрения дальше от зрителя, выглядят более тусклыми. Обычно этот эффект непрерывен, и поверхность даже одного объекта несет на себе признаки глубины, предоставляя пользователю важную информацию о размерах, форме и протяженности объекта. Индикация глубины в приложении к сеткам позволяет четко определять ориентацию сетки в сцене.

Один из методов позиционирования объектов в некоторых трехмерных приложениях связан с **тенями** – силуэтами выделенных объектов, спроецированными на координатные сетки так, как если бы луч света падал на каждую сетку строго перпендикулярно к ней. Перемещая объект в трехмерном пространстве, пользователь может наблюдать за перемещением или изменением размера в каждом измерении благодаря теням на сетках.

Тени прекрасно справляются со своей задачей, но все эти сетки и тени перегружают картинку на экране. Альтернативой является единственная **напольная сетка** и **нормаль**. Нормали работают во взаимодействии с горизонтально ориентированной сеткой. Когда пользователь выделяет объект, появляется вертикальная линия, связывающая центр объекта и сетку. По мере перемещения объекта нормаль перемещается вместе с ним, оставаясь вертикальной. Пользователь следит за движением объекта, наблюдая за основанием нормали, передвигающимся по сетке (в осях X и Y), и за ее длиной и ориентацией по отношению к сетке (ось Z).

Направляющие и прочие обогащенные визуальные подсказки

Идиомы, описанные в предыдущих разделах, являются примерами обогащенной немодальной обратной связи, которую мы подробно обсудим в главе 25. Однако в некоторых приложениях многочисленные сетки и нормали могут создавать перегрузку. Рассмотрим в качестве примера программу SketchUp компании Google, предназначенную для создания архитектурных эскизов. Пользователи программы могут чертить собственные линии при помощи таких инструментов, как рулетка и транспортир, и по ходу работы получают подсказки в виде цветовых кодов, позволяющих ориентироваться относительно осей. Кроме того, пользователь может включить голубое градиентное «небо» и цвет «земли», что тоже помогает ориентироваться в пространстве. Поскольку приложение специализировано на архитектурных эскизах

и не предназначено для универсального решения задач трехмерного моделирования и анимации, его разработчикам удалось создать экономичный, мощный и простой интерфейс, легкий как в изучении, так и в использовании (рис. 19.13).

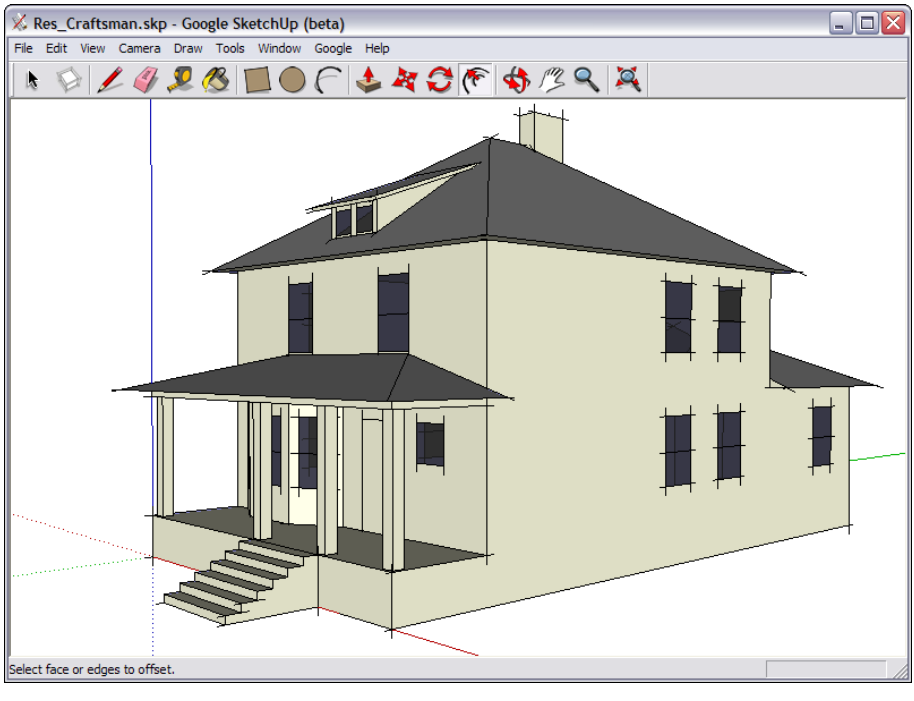


Рис. 19.13. *Программа SketchUp компании Google – блестящее приложение, объединяющее возможности создания трехмерных архитектурных эскизов с гладкой интерактивностью, обогащенной обратной связью и легко управляемым набором инструментов проектирования. Пользователи могут задать цвет «неба» и реалистичное изображение теней сообразно географическому положению, ориентации, времени суток и года. Все это не только способствует визуализации, но и помогает пользователю ориентироваться. Пользователь может выводить на экран трехмерную сетку и направляющие линии для измерений – точь-в-точь как в приложении для создания двухмерных эскизов. Функции поворота и наезда камеры удобно привязаны к колесику мыши, что не препятствует одновременному использованию других инструментов. Всплывающие текстовые подсказки облегчают рисование линий и выравнивание объектов*

Каркасы и ограничивающие боксы

Каркасы и **боксы** решают проблемы видимости объектов. Во времена слабых процессоров все объекты изображались в виде «проволочных» каркасов, потому что компьютеры были недостаточно быстры, чтобы

выполнять обсчет поверхностей в реальном времени. Сегодня приложения моделирования обычно отображают поверхности выделенных объектов, а остальные объекты представляют в виде каркасов. Прозрачность как свойство объектов тоже может послужить этой цели, но она требует больших вычислительных ресурсов. В очень сложных сценах иногда необходимо или желательно отображать вместо невыделенных объектов ограничивающие боксы, хотя, конечно, это не идеальное решение.

Проблемы ввода и идиомы

В приложениях трехмерного моделирования приживаются многие идиомы из программ для работы с плоской графикой – в частности, идиомы маркеров перетаскивания и вершинных маркеров. Однако у трехмерного ввода есть свои особенности.

Пороги смещения

Одной из фундаментальных проблем непосредственного манипулирования объектами в двухмерной проекции трехмерной сцены является проблема преобразований плоского движения курсора в осмысленное движение в трехмерном виртуальном пространстве.

В трехмерной проекции необходим особый порог смещения, позволяющий различать движение по трем, а не двум осям. Как правило, движения курсора вверх и вниз преобразуются в движение вдоль одной оси, а перемещения под углом 45 градусов относятся к оставшимся двум осям. В программе SketchUp применяются подсказки в виде пунктирных линий разного цвета, которые появляются, когда пользователь перетаскивает объект параллельно определенной оси. Кроме того, применяются всплывающие текстовые подсказки. В трехмерной среде богатая обратная связь в виде смены формы курсора и других подсказок становится жизненно необходимой.

Проблема захвата

Другая серьезная проблема трехмерного манипулирования известна как **проблема захвата**. Поскольку при сборке сцен объекты должны быть прозрачными или представленными в виде каркасов, программе трудно понять, какой из многочисленных перекрывающихся объектов пользователь хочет выделить, наведя указатель мыши. Здесь может помочь подсветка, но ее недостаточно, поскольку объект может оказаться полностью перекрытым другими объектами. Групповое выделение еще более сложное.

Многие приложения трехмерной графики прибегают к косвенным методам, таким как список или иерархическая структура объектов вне трехмерного представления, где пользователь может выбрать нужный объект. Хотя такой тип взаимодействия находит себе применение, существуют более прямолинейные подходы.

Например, наведение указателя мыши на какой-то участок сцены может открыть всплывающее меню, позволяющее пользователю выделить один или несколько перекрывающихся объектов (в подобном меню нет необходимости, если можно недвусмысленно указать на нужный объект). Если могут быть выделены отдельные грани, вершины или ребра, то каждый такой элемент должен сигнализировать о своей отзывчивости, когда по нему проходит указатель мыши.

Хотя это не связано с обсуждаемой темой напрямую, простой способ навигации по сцене может упростить решение проблемы захвата. В программе SketchUp функции наезда и **орбитального** вращения камеры привязаны к колесику мыши. Покрутите колесико – и изображение увеличится или уменьшится относительно центральной нулевой точки в трехмерном пространстве. Нажмите на колесико – и вы включите режим облета, позволяющий вращать камеру вокруг центральных осей в любом направлении. Такая плавная навигация позволяет манипулировать архитектурной моделью почти с такой же легкостью, с какой вы могли бы вертеть ее в руках.

Вращение объектов; движение, вращение, наезд камеры

Темой, специфичной для приложений трехмерной графики, является количество функций пространственного манипулирования. Объекты можно перемещать, масштабировать, видоизменять по любой из трех осей. В дополнение к этому их можно вращать вокруг трех осей. Мало того, камеру тоже можно поворачивать или перемещать по кругу относительно трех осей и фокусной точки. Наконец, камера может наезжать и отъезжать.

Сказанное означает не только важность применения в приложениях трехмерной графики клавиатурных сокращений. Есть еще одна проблема: точка зрения оператора камеры не позволяет легко различать трансформацию камеры и трансформации объектов, хотя разница может быть очень существенной. Одним из методов решения проблемы может стать размещение в углу экрана миниатюры с абсолютным видом сцены. При желании ее можно увеличить или уменьшить, и она может служить для проверки сущности трансформации и для навигации, если пользователь заблудится в сцене. (Заметьте, что миниатюра такого рода может оказаться полезной и для навигации на плоскости.)

Связывание объектов

В некоторых приложениях очень мощным инструментом может оказаться такая идиома прямого манипулирования, как **связывание**. Пользователь щелкает по объекту и перетаскивает указатель мыши к другому объекту. При этом первый объект не перетаскивается, а вместо этого от него ко второму объекту протягивается соединительная линия или стрелка.

Если вы применяете средства управления проектами или построения организационных диаграмм, вы, несомненно, знакомы с этой идиомой. Например, чтобы соединить одно задание с другим на сетевой диаграмме (называемой также PERT-схемой[1]) в программе управления проектом, вы щелкаете и протягиваете между ними стрелку. В этом случае направление стрелки имеет значение: задание, на котором была нажата кнопки мыши, является *источником*, а задание, на котором кнопка мыши отпущена, – *приемником*.

Когда пользователь связывает два объекта, процесс сопровождается визуальной обратной связью в виде **резиновой нити**: стрелка образует линию, которая протягивается от точки щелчка до текущего положения курсора мыши. Линия анимирована, один ее конец следует за указателем мыши, а второй закреплен в точке щелчка. Когда курсор проходит над кандидатами для связывания, курсорная подсказка должна сообщать о том, что объекты можно соединить. Когда пользователь отпускает кнопку мыши над корректной целью, программа проводит стрелку от одного объекта к другому. В некоторых программах объекты при этом дополнительно связываются логически. Как и в случае перетаскивания объектов с одного места на другое, крайне важно обеспечить удобную функцию отмены – скажем, привязать эту функцию к клавише <Esc> или аккордному щелчку.

Связи между объектами сами могут быть полноценными объектами с маркерами формы и редактируемыми свойствами. При такой реализации пользователь может выделять, передвигать и удалять связи. Когда связи между объектами в программе должны нести дополнительную информацию (например, если речь идет о программе для планирования проектов), имеет смысл делать их полноправными объектами.

Связывание не требует столь богатых курсорных подсказок, как другие идиомы, поскольку действие резиновой нити вполне наглядно. Однако в программе, устанавливающей также логическую связь между объектами, будет полезным показывать, какие объекты могут быть целью для стрелки. Другими словами, когда пользователь протягивает стрелку к какой-либо пиктограмме или элементу управления, как он узнает, можно ли установить связь с этим объектом? Потенциальная цель должна активно демонстрировать визуальную подсказку. Если все объекты в равной степени могут быть целями связывания, такая подсказка может быть неброской или отсутствовать вовсе. Однако целевые объекты обязательно должны подсвечиваться, когда по ним проходит указатель мыши, чтобы продемонстрировать готовность принять связь.

---

[1] PERT, Program Evaluation and Review Technique (буквально: техника оценки и анализа программ) – вид диаграмм, используемых для составления плана проекта, оценки времени его выполнения и пересмотра. – *Примеч. ред.*

20

Поведение окон

В любой книге, посвященной проектированию пользовательских интерфейсов, должно присутствовать обсуждение окон – неотъемлемой части современных графических интерфейсов. Хотя оконные системы позволяют сделать интерфейс модульным и гибким, их можно применять не только во благо. В этой главе мы сначала рассмотрим вопрос в исторической перспективе, а затем обсудим важные вопросы проектирования, связанные с использованием окон в приложениях.

PARC и Alto

Все современные графические пользовательские интерфейсы унаследовали свой внешний вид от Xerox Alto – экспериментальной компьютерной системы для настольных компьютеров, разработанной в середине 1970-х годов в исследовательском центре компании Xerox в Пало Альто (PARC – Palo Alto Research Center), ныне PARC, Inc. Компьютер Alto, созданный в PARC, был первым компьютером с графическим интерфейсом. Его создали для того, чтобы исследовать возможности применения компьютеров в качестве настольных бизнес-систем. Alto был спроектирован как сетевая офисная система, позволяющая создавать документы, редактировать и просматривать их в режиме WYSIWYG (what you see is what you get – «что видите, то и получаете»), хранить, искать, передавать между рабочими станциями в электронном виде и печатать. Мир персональных компьютеров получил в подарок от системы Alto множество важных нововведений, которые ныне являются общепринятыми: мышь, прямоугольное окно, полоса прокрутки, кнопка, метафора «рабочего стола», объектно-ориентированное программирование, раскрывающиеся меню, стандарт локальной сети Ethernet и лазерная печать.

Трудно переоценить влияние лаборатории PARC на всю компьютерную индустрию и на то, как мы работаем с компьютерами сегодня. Как глава Apple Computer Стив Джобс, так и основатель Microsoft Билл Гейтс видели систему Alto в лаборатории PARC и получили незабываемые впечатления.

Компания Xerox предприняла самостоятельную попытку превратить в коммерческий продукт сначала Alto, а затем и последовавшую за ней компьютерную систему Star. Однако обе системы оказались дорогими, сложными, мучительно медленными и потерпели поражение. У многих специалистов возникло чувство, что руководству корпорации Xerox, занимавшейся в то время главным образом копировальной техникой, не хватает дальновидности и находчивости для организации эффективной рекламы и продаж «безбумажных офисов». Когда стало понятно, что Xerox упустила эту сказочную возможность, в PARC началась утечка мозгов, изрядно обогатившая другие компьютерные компании, в частности Apple и Microsoft.

Стив Джобс с беженцами из PARC сразу же попытались создать копию Alto/Star в системе Lisa. Им это удалось во многих отношениях, включая даже неспособность Star отвечать требованиям реального мира. Lisa была замечательной, простой в использовании, восхитительной, слишком дорогой (9995 долларов в 1983 году) и ужасно медленной. Несмотря на коммерческий провал эта система разбудила воображение многих людей в маленькой, но уже процветающей отрасли микрокомпьютеров.

В то же время Билл Гейтс был сильно впечатлен не столько привлекательной «графичностью» Alto/Star, сколько объектно-ориентированной моделью представления и коммуникационной моделью. Этот стиль мышления нашел свое отражение в программном обеспечении, выпущенном компанией Microsoft в начале 1980-х, в первую очередь – в электронной таблице Multiplan (предшественнице Excel).

Стива Джобса не смутил провал компьютера Lisa. Он был убежден в своевременности идеи о полностью графическом персональном компьютере, родившейся в PARC. Укрепив свою группу беженцев из PARC квалифицированными и энергичными сотрудниками из различных подразделений Apple, он начал секретные разработки коммерчески жизнеспособного варианта Alto. Результатом этих усилий стал легендарный Macintosh – машина, оказавшая глубочайшее влияние на наши технологии, дизайн и культуру. Mac единолично продемонстрировал всей компьютерной индустрии важность дизайна и эстетики. Он не только поднял планку дружелюбности программ к пользователю, но и открыл мир компьютеров целому слою профессионалов из различных областей, преодолев зацикленность компьютерной индустрии на технологических безделушках.

Почти религиозная аура, окружавшая компьютеры Macintosh, бросала отсвет и на многие аспекты пользовательского интерфейса этих машин: раскрывающиеся меню, метафоры, диалоговые окна, прямо-

угольные перекрывающиеся окна и другие элементы интерфейса стали частью его загадочного обаяния. К сожалению, из-за того, что дизайн Macintosh достиг таких высот, его реальные недостатки остались незамеченными.

Принципы PARC

Работая над Alto, исследователи лаборатории PARC создали не только революционные технологии программного и аппаратного обеспечения, но и многие из тех концепций, которые сегодня считаются непререкаемыми в среде проектировщиков интерфейсов и разработчиков приложений.

Визуальные метафоры

Одной из идей, рожденных в PARC, была *визуальная метафора*. Глобальная визуальная метафора считалась в PARC решающей для понимания системы пользователем, а значит – критической для успеха всего продукта и его концепции в целом. Как говорилось в главе 13, применение метафоры при проектировании взаимодействия ведет к значительным ограничениям. Однако совершенно неудивительно, что первые проектировщики интерфейсов посчитали этот подход привлекательным для масс, совершенно не знакомых с компьютерами.

Отказ от режимов

Еще одним принципом современных графических пользовательских интерфейсов является мысль о том, что режимов следует избегать. **Режим** – это состояние программы, при котором результаты действий пользователя могут отличаться от обычных, то есть ответвление в поведении программы.

Так, например, старые программы требовали от пользователя переключения в особый режим для ввода данных, а затем переключения в другой режим для их печати. Такие поведенческие состояния и называются режимами, и они способны очень сильно запутывать и раздражать пользователей. Ларри Теслер (Larry Tesler), бывший исследователь в PARC и ведущий научный сотрудник в Apple, стал одним из первых сторонников отказа от режимов в программном обеспечении. На фотографии в одном влиятельном журнале он был запечатлен в футболке с дерзкой надписью «Don't mode me in» (Не режимьте меня). Надпись на номерном знаке его машины гласила «NOMODES» (БЕЗ РЕЖИМОВ). При работе с командной строкой режимы и в самом деле отвратительны. Однако в мире графических пользовательских интерфейсов, где царит порядок «объект – глагол», режимы не являются безоговорочно вредными, хотя *плохо спроектированные* режимы могут чрезвычайно раздражать. К несчастью, принцип «не режимьте меня» стал уже неотъемлемой частью языка проектировщиков.

Например, одной из программ для компьютеров Macintosh, оказавших большое влияние на индустрию, было приложение MacPaint, в интерфейсе которого режимы применялись весьма интенсивно. Эта программа давала пользователю возможность рисовать на экране пиксел за пикселом. Пользователь выбирал инструмент из палитры, содержавшей около десятка инструментов, и затем рисовал этим инструментом на экране. Выбор инструмента – это вход в режим, поскольку поведение программы определяется свойствами выбранного инструмента. В каждом режиме программа ведет себя строго определенным образом.

Исследователи из лаборатории PARC не ошибались – их просто не так поняли. По сравнению со многими современными программами преимущества пользовательского интерфейса MacPaint были довольно велики, просто их источником было не воображаемое отсутствие режимов, а скорее, легкость, с которой можно было задавать текущий режим и менять его.

Перекрывающиеся окна

Другим фундаментальным принципом компьютеров Mac, возникшим в лаборатории PARC (и пустившим метастазы в Microsoft Windows), является идея перекрывающихся прямоугольных окон. Прямоугольный мотив в современных графических пользовательских интерфейсах настолько силен и вездесущ, что его часто считают жизненно важным для успешного визуального взаимодействия.

Существуют серьезные причины отображать данные внутри прямоугольных рамок. Возможно, наименее существенная из этих причин – соответствие технологии производства дисплеев: ЭЛТ- и ЖК-дисплеям легче работать с прямоугольниками, чем с другими формами. Гораздо более важно то, что люди по большей части пользуются прямоугольным форматом для вывода данных: текст размещают на прямоугольных страницах со времен Гутенберга, и многие другие формы выражения, такие как фотография, кино или видео, тоже соответствуют прямоугольным сеткам. Прямоугольные графики и диаграммы являются самыми легкими для восприятия. Видимо, в прямоугольниках есть нечто, соответствующее человеческому разуму. Кроме того, прямоугольники позволяют довольно эффективно использовать пространство.

Перекрывающиеся окна наглядно показали, что существуют более подходящие способы переключения управления между одновременно запущенными приложениями, нежели таинственные команды командной строки. Изначально окна были призваны представлять листы бумаги на письменном столе пользователя. Хорошо, но почему именно таким образом? Ответом на этот вопрос опять-таки является глобальная метафора рабочего стола. Ваш письменный стол (если он похож на наши столы) завален различными бумагами. Когда вам хочется прочитать или отредактировать какую-либо из них, вы извлекаете соответ-

ствующий лист из этой кучи, кладете его сверху и принимаетесь за работу. Однако это комфортно в той же степени, что и работа с бумагами на настоящем столе, которая не особенно удобна, когда стол завален бумагами и имеет диагональ всего 21 дюйм.

Сама *концепция* перекрывающихся окон хороша, но ее исполнение в реальном мире непрактично. Метафора перекрывающихся листов бумаги перестает работать, как только на экране появляются три или более программ и документов, – она попросту не масштабируется. Есть у этой идиомы и другие проблемы. Если пользователь промахнется мышью на пару пикселов, нужная ему программа может исчезнуть, уступив место другой. Пользовательское тестирование, проведенное компанией Microsoft, показало, что типичный пользователь может несколько раз запустить один и тот же текстовой редактор, ошибочно полагая, что он каким-то образом умудрился «потерять» программу и должен начать все сначала. Именно проблемы такого рода побудили Microsoft ввести в систему панель задач, а компанию Apple – придумать инструмент Expose, ставший привлекательным и удобным средством для слежения за открытыми окнами (хотя отсутствие его интеграции с приложениями, свернутыми в Dock, представляет собой реальную проблему).

Еще один аспект путаницы с перекрывающимися окнами связан с тем, что в реализациях некоторых других идиом также применяются перекрывающиеся окна. Первая такая идиома – знакомые диалоговые окна, а кроме них есть еще меню и плавающие панели инструментов. Подобное наложение окон внутри одного приложения – естественная и удобная идиома; более того, она обладает определенным метафорическим смыслом: кто-то как будто бы вручает вам важное сообщение.

Microsoft и окна плиткой

Следуя традиции обращать внимание на самые заметные аспекты нового графического пользовательского интерфейса лаборатории PARC, Билл Гейтс дал своему неряшливому, собранному впопыхах ответу на успех системы Macintosh название «Windows» (Окна).

Первая версия Microsoft Windows несколько отклонилась в сторону от находок Xerox и Apple. Вместо перекрывающихся прямоугольных окон, представляющих накладывающиеся друг на друга листы бумаги на письменном столе, первая версия Windows располагала окна так называемой **плиткой**, что давало пользователям иметь несколько открытых приложений на экране одновременно. Расположение окон плиткой означало деление доступного экранного пространства между запущенными приложениями в виде равномерной прямоугольной мозаики. Плитка появилась как идеалистический способ решения проблем с ориентацией и навигацией, порождаемых перекрывающимися окнами. Навигация между окнами, уложенными плиткой, гораздо проще

навигации между перекрывающимися окнами, но за эту легкость приходиться платить чрезвычайно дорогими пикселами. Кроме того, как только пользователь начинает двигать аккуратно расположенные плиткой окна, он сразу же сталкивается с интерфейсным налогом перекрывающихся окон (см. главу 11). Плитка как основная идиома не получила широкого распространения, но ее все еще можно встретить в самых неожиданных местах: попробуйте, например, щелкнуть правой кнопкой мыши на панели задач в современной версии Windows.

Полноэкранные приложения

Перекрывающиеся окна не могут упростить навигацию между несколькими открытыми приложениями, так что производители продолжают поиск новых путей. **Виртуальные рабочие столы** менеджера сеансов на некоторых UNIX-платформах расширяют рабочий стол до шестикратного размера видимого экрана. (Apple недавно представила похожую возможность в Mac OS X.) В углу экрана появляется миниатюрное изображение всех шести рабочих столов, каждый из которых может содержать по нескольку открытых окон, соответствующих различным активным приложениям. Для переключения между такими виртуальными рабочими столами можно щелкать по их миниатюрам. В некоторых реализациях можно даже перетаскивать миниатюры окон с одного рабочего стола на другой.

Microsoft стойко выдержала двойной судебный процесс против Apple – нарушение контракта и нарушение патентных соглашений, – чтобы иметь возможность добавить технологию перекрывающихся окон в Windows 2.0. Похоже, что в пылу сражений была забыта главная проблема: как облегчить навигацию пользователя между приложениями? Множество окон, разделяющих маленький экран – перекрывающихся либо уложенных плиткой, – не очень удачное решение в общем случае (хотя и бывает полезным в определенных ситуациях). Мы быстро движемся в мир полноэкранных программ. Каждое приложение занимает весь экран, когда оно активно. Инструменты вроде панели задач отнимают минимум пикселов у активных приложений, чтобы обеспечить наглядный способ переключения между задачами. (Забавно, но похожая концепция использовалась на заре существования Macintosh в специальной программе Switcher, которая применялась для переключения между несколькими полноэкранными приложениями.) Такое решение гораздо лучше сберегает пикселы, не так сильно путает пользователей и лучше подходит для тех случаев, когда приложение используется продолжительное время. В операционных системах Mac OS X, Windows XP или Vista у пользователей есть выбор: разворачивать приложения на весь экран или работать в режиме перекрывающихся окон.

Проектирование современных программных продуктов часто начинается с предположения, что пользовательский интерфейс не содержит

режимов и состоит из ряда перекрывающихся окон, связанных общей глобальной метафорой. Наследие PARC все еще очень сильно. Большая часть наших представлений о современном проектировании графических пользовательских интерфейсов – как правильных, так и ошибочных – уходит корнями к этому наследию. Однако хороший проектировщик отрекается от мифов и смотрит на вопросы проектирования программ свежим взглядом, пользуясь историей как путеводителем, а не как догмой.

Многопанельные приложения

Оказывается, все же существует идиома, привносящая в монопольные полноэкранные приложения лучшее из плиточного расположения окон, – это идиома **многопанельных окон**. Многопанельные окна состоят из независимых представлений, или **панелей**, соседствующих внутри одного окна. **Смежные панели** разделяются с помощью закрепленных или подвижных **разделителей**. (Мы обсудим разделители более подробно в главе 21.) Классическим примером многопанельного приложения является Microsoft Outlook, в котором для показа на одном экране списка почтовых ящиков, содержимого выбранного ящика и выбранного письма используются различные панели (рис. 20.1).

Преимуществом многопанельных окон является то, что внутри одного окна монопольного приложения можно отобразить несколько независимых, но связанных фрагментов информации, сводя тем самым к минимуму проблемы навигации и управления окнами. Для монопольного приложения любой сложности наличие смежных панелей практически неизбежно. Говоря более точно, интерфейсные решения, предоставляющие средства навигации и/или строительные блоки в одной панели и позволяющие просматривать либо конструировать данные в смежной панели, представляют собой эффективный шаблон, достойный подражания.

Смежные панели нашли применение и в Интернете в форме так называемых **фреймов**, однако вследствие неудачной изначальной реализации и войны стандартов между гигантами Netscape и Microsoft фреймы так и остались сложными и неуклюжими. Хочется верить, что с развитием веб-технологий и распространением высокоинтерактивных веб-приложений концепция, стоящая за фреймами, снова найдет применение в броузерах (сами-то броузеры *уже* используют панели). Существующие сегодня в Интернете клиентские технологии (AJAX и Flash) уже позволяют в определенной степени реализовать поведение с применением панелей.

Другая форма множественных панелей – это **стопка панелей**, или **вкладки**. Хотя чаще всего вкладки встречаются в диалоговых окнах (см. главу 24), иногда они оказываются полезными в окнах монопольных приложений. Хорошим примером такого использования является

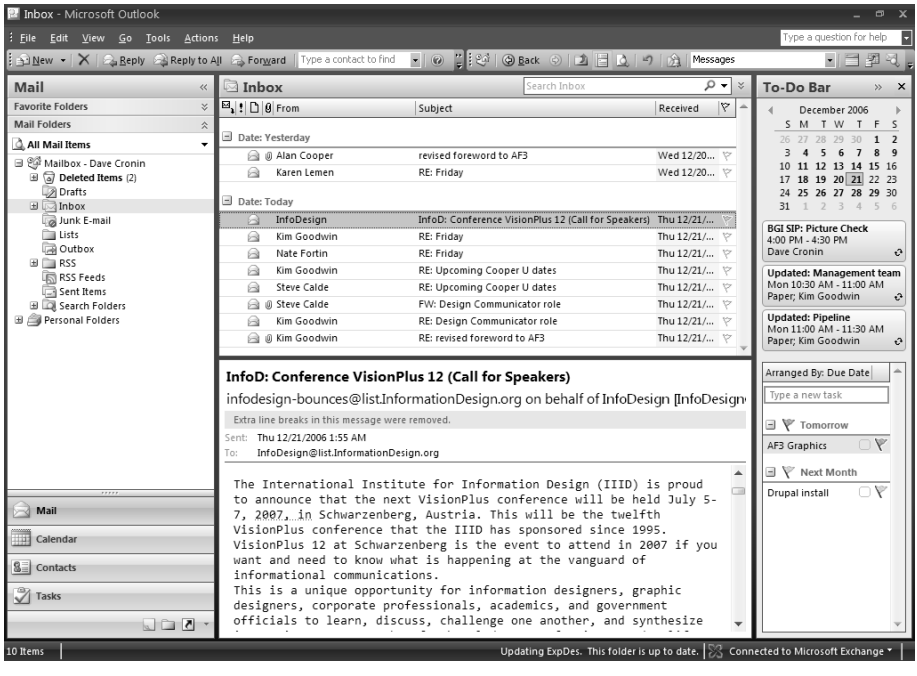


Рис. 20.1. *Microsoft Outlook 2007 – классический пример многопанельного приложения. Крайняя левая панель содержит перечень почтовых ящиков, а также позволяет переключаться между почтой и календарем; в верхней центральной панели перечислены все сообщения выбранного почтового ящика, а панель под ней отображает содержимое выбранного сообщения. Панель справа показывает три ближайших встречи и текущие задачи*

Microsoft Excel, который использует стопки панелей для организации своих «листов», позволяя обращаться к нескольким связанным между собой таблицам посредством перевернутых вкладок в нижней части экрана.

Проектирование окон

Строительным материалом для наших программ служат окна двух видов – главные и подчиненные (например, документы и диалоговые окна). Выбор способа применения окон в приложении – важный аспект общей инфраструктуры пользовательского интерфейса (см. главу 7).

Лишние комнаты

Если мы представим наше приложение в виде дома, то каждое окно – это отдельная комната. Сам дом будет представлен главным окном программы, а каждая комната – это окно документа, диалоговое окно или панель. Мы не пристраиваем к нашему дому комнату, если у нее

нет уникальной функции, которую не способны выполнять другие комнаты. Точно так же мы не должны добавлять окно в программу, если у него нет предназначения, которое не могут или не должны выполнять существующие окна.

Важно подходить к вопросам предназначения с позиции будущих целей и пользовательских ментальных моделей. Думая о том, каково **предназначение** комнаты, мы подразумеваем определенную цель – не обязательно конкретную задачу или функцию.

Диалоговое окно – это еще одна комната; не ходите в комнату без веской причины.

К примеру, чтобы изменить яркость и контраст фотографии в Adobe Photoshop, нужно открыть меню Image (Изображение), выбрать подменю Adjustments (Коррекция), а затем выполнить команду Brightness/Contrast (Яркость/Контрастность). Откроется диалоговое окно, где можно выполнить коррекцию (рис. 20.2). Подобная последовательность дей-

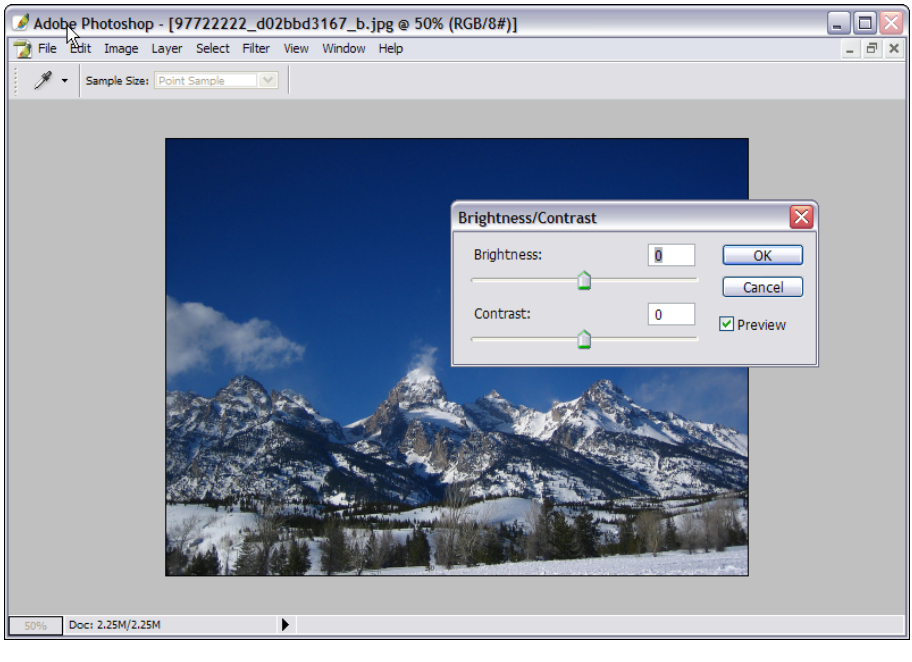


Рис. 20.2. *Одна из многих «комнат» Adobe Photoshop: Brightness/Contrast (Яркость/Контрастность). Мы настолько привыкли к необходимости вызывать диалоговое окно, чтобы выполнить простую функцию, что уже не замечаем этого. Однако такой подход заставляет пользователя делать лишнюю работу; к тому же диалоговое окно перекрывает собой самый важный объект на экране – собственно изображение*

ствий настолько распространена, что мы даже не замечаем ее, – и все же это несомненно плохое решение.

Коррекция изображения – основная задача программы редактирования изображений. Фотография находится в главном окне – и здесь же должны находиться инструменты, воздействующие на фотографию. С точки зрения назначения приложения изменение яркости и контраста – задача не вспомогательная, а самая что ни на есть ключевая.

Размещение функций в диалоговом окне подчеркивает их обособленность от главной задачи. Коррекция яркости и контраста в диалоговом окне работает замечательно, но создает затруднения при взаимодействии. С точки зрения программиста коррекция яркости и контраста – всего лишь одна функция, не связанная с множеством других функций, поэтому кажется естественным отвести ей собственный контейнер. Но с точки зрения пользователя эта функция – неотъемлемая часть той работы, которую ему необходимо сделать, и потому должна быть явно представлена в основном окне.

Ситуация значительно улучшилась в программе Adobe Lightroom. Приложение разделено на представления, или «комнаты». Каждая «комната» имеет специфическое назначение: Library (Библиотека), Develop (Проявка), Slideshow (Слайд-шоу), Print (Печать) и Web (Веб). В комнате Develop управление яркостью и контрастом представлено в панели справа от главного окна наряду со всеми прочими инструментами коррекции изображений, какие только можно себе вообразить (рис. 20.3).

ПРИНЦИП проектирования

Встраивайте функции в то окно, где они применяются.

Это один из наиболее часто нарушаемых принципов в проектировании пользовательских интерфейсов. Программисты выполняют свою работу, разбивая приложение на дискретные функции, при этом пользовательский интерфейс часто разрабатывается в тесном соседстве. В сочетании с изумительной легкостью, с которой программисты могут создавать диалоговые окна, это дает закономерный результат: по диалоговому окну на каждую функцию. Современные инструменты создания графических пользовательских интерфейсов всемерно облегчают создание диалоговых окон, в то же время добавление элементов управления на поверхность главного окна или создание идиом непосредственного манипулирования обычно ими не поддерживается. Разработчику, желающему создать улучшенный пользовательский интерфейс, зачастую приходится создавать собственный инструментарий, не рассчитывая на поддержку поставщиков инструментального программного обеспечения.

Рис. 20.3. *Adobe Lightroom – существенный шаг вперед в сравнении с Photoshop. Принципиально важные инструменты сгруппированы по назначению и представлены прямо в главном окне, рядом с редактируемым изображением*

Жизненно важные комнаты

Когда вы хотите поплавать, было бы странно предложить вам в качестве раздевалки гостиную, наполненную людьми. Воспитанность и благопристойность в данном случае являются веской причиной для того, чтобы вам захотелось иметь отдельную комнату для переодевания. Наличие отдельной комнаты является абсолютно уместным, если имеется явная потребность в ней.

Когда пользователи совершают действия, выходящие за пределы обычного хода событий, программа должна отвести специальное место для их выполнения. Например, оптимизация базы данных – это не повседневная задача. Она включает в себя использование ряда приспособлений и средств, которые не используются во время обычного рабочего процесса при работе в приложении с интегрированной базой данных. Более прозаические части приложения помогают решать повседневные задачи, такие как ввод или просмотр записей, но массовое стирание записей – не повседневная задача, поэтому операция полномасштабной оптимизации базы данных должна располагаться в отдельном диалоговом окне. Для программы совершенно естественно пе-

ревести пользователя для выполнения этой функции в отдельную «комнату» – либо окно, либо диалоговое окно.

Руководствуясь принципами целеориентированного мышления, мы можем определить полезность каждой функции. Если человек рисует в графическом редакторе, его цель – создать привлекательное и эффектное изображение. Все инструменты рисования непосредственно связаны с этой целью, но самые нужные инструменты – это карандаши, кисти и ластик. Эти инструменты следует глубоко интегрировать в рабочее пространство, ведь так поступают художники в реальном мире, располагая свои инструменты у холста, под рукой. Инструменты готовы к моментальному применению, за ними не нужно даже тянуться, не говоря уже о том, чтобы ходить за ними в другую комнату. Инструменты программы рисования следует располагать у границы холста и делать доступными в один щелчок мышью. Пользователя не следует отправлять за ними в меню или диалоговое окно.

Например, Corel Painter размещает инструменты художника на специальной панели и позволяет перемещать их так, чтобы самые нужные находились в максимальной доступности. Панели и палитры можно скрывать, но по умолчанию они включены и являются частью главного окна рисования. Их можно располагать где угодно в пределах окна. Если вы создадите тонкую угольную кисть конкретного красного оттенка, которую вам нужно будет многократно использовать, просто «оторвите» ее от палитры и поместите где угодно в рабочей области – прямо как на настоящем мольберте. Этот способ выбора инструментов имитирует действия художника в реальном мире.

С другой стороны, если вам потребовалось импортировать готовый фрагмент изображения, то, хотя эта функция тоже относится к конечной цели – созданию хорошей иллюстрации, используемые при этом инструменты не связаны напрямую с рисованием. Очевидно, что каталог бесплатных изображений имеет опосредованное отношение к основным целям художника – это всего лишь средство для достижения цели. Обычный художник, скорее всего, не будет держать книгу с каталогом готовых изображений прямо на мольберте; однако весьма вероятно, что такая книга будет лежать неподалеку – скажем, на книжной полке, откуда он сможет достать ее, не вставая с места. Подобно этому в компьютерной программе для рисования средства для работы с готовыми рисунками должны быть легкодоступны, но поскольку речь идет о целом ряде инструментов, не являющихся необходимыми при обычном рисовании, их следует разместить отдельно, например в диалоговом окне.

Закончив работу над своим произведением искусства, художник достиг первоначальной цели – создал эффектное изображение. В этот момент его цели меняются: новой целью становится сохранение рисунка, его защита, а также донесение своих идей до зрителей посредством рисунка. Потребность в кистях и карандашах отпадает, потребность

в готовых фрагментах отпадает, и теперь художник может легко отказаться от соответствующих инструментов. В реальном мире художник в этот момент открепил бы рисунок от мольберта, вынес бы его в зал и обработал фиксатором. Затем рисунок был бы свернут и вложен в тубус. Заметьте, что художник не случайно убирает свои инструменты для рисования: с одной стороны, он не хочет испортить их фиксатором, а с другой – стремится избежать нечаянного повреждения законченного рисунка углем или краской. Тубусы для почтовой пересылки используются нечасто и к тому же не особенно тесно связаны с самим процессом рисования, поэтому их уместно хранить в чулане. Компьютерный эквивалент этого процесса выглядит так: вы закрываете программу рисования, убираете инструменты, находите подходящее место на жестком диске для хранения изображения, а затем отсылаете его кому-то по электронной почте. Эти функции четко отделены целями и мотивами от процесса рисования.

Изучая цели пользователей, мы естественным образом приходим к уместной форме приложения. Вместо того чтобы отвести по диалоговому окну на каждую функцию, мы делаем вывод, что некоторым функциям вообще не место в диалоговых окнах, другие следует помещать в диалоги, тесно связанные с основным интерфейсом, а есть и такие функции, которые нужно попросту удалить из программы.

Замусоривание окнами

Некоторые проектировщики считают, что каждое диалоговое окно должно вмещать единственную функцию. Результатом такого подхода становится замусоривание **интерфейса окнами**.

Решение многих пользовательских задач требует выполнения последовательности действий. Если для каждого действия открывается отдельное диалоговое окно, то экран вскоре визуально перегружается, а навигация усложняется. Программа CompuServe Navigator (рис. 20.4) служит иллюстрацией именно такой ситуации.

Смазав машинным маслом педальный узел своего велосипеда, вы обнаружите, что крутить педали стало легче; однако это вовсе не означает, что, облив велосипед маслом, вы сделаете его самодвижущимся. Проектировщик программы Navigator всячески стремился увеличить количество окон в нашей жизни (возможно, искренне – но ошибочно – полагая, что окна хороши по самой своей природе).

Полезность любой идиомы взаимодействия зависит от контекста.

Возможно также, что программа Navigator стала результатом труда большой команды программистов, работавших без общей спецификации дизайна. В результате каждый функциональный модуль был вы-

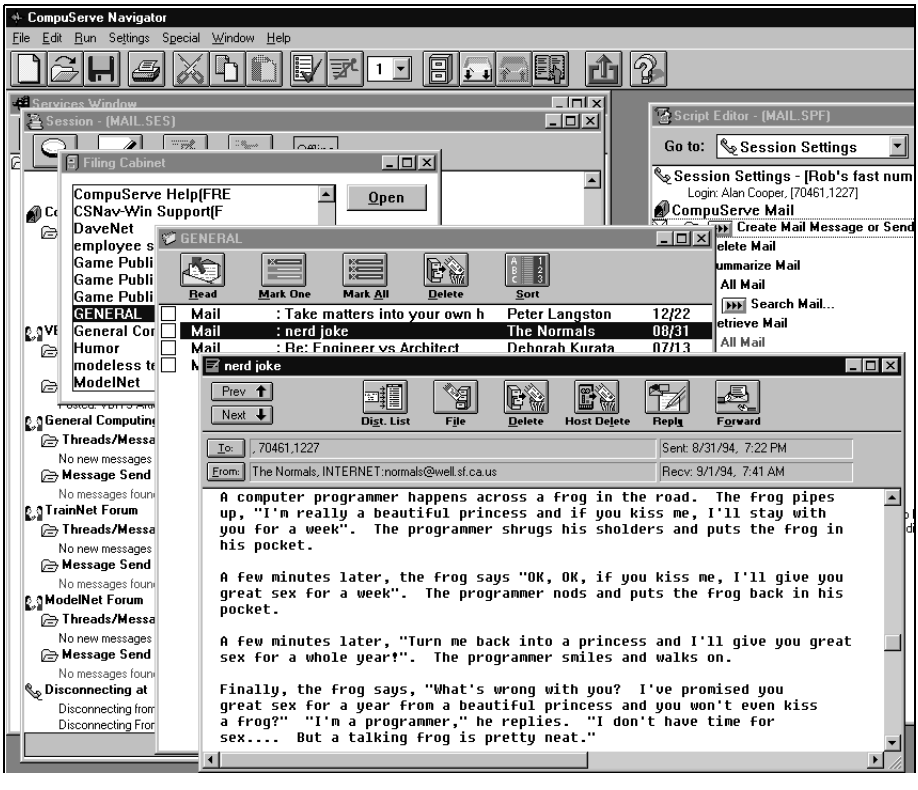


Рис. 20.4. *Версия 1.0 программы CompuServe Navigator серьезно страдала от мусорных окон. Обычная загрузка электронной почты требовала открытия трех окон. Для прочтения сохраненного письма нужно было открыть еще три. Просмотр почты является целостной деятельностью – и вся эта деятельность должна происходить в одном интегрированном окне. Однако худшее еще впереди: пользователям приходилось закрывать каждое из этих окон по отдельности – в порядке, противоположном тому, в котором они были открыты*

полнен в форме самостоятельного окна либо отдельного диалогового окна, поскольку это проще, нежели пытаться аккуратно согласовать их по факту создания. Перед нами во всей красе предстает классическая модель реализации. Этот пример присутствовал уже в первом издании нашей книги, вышедшей в 1995 году; нам очень хотелось бы сказать, что положение дел с тех пор заметно улучшилось. Однако достаточно лишь бросить взгляд на современный интерфейс America Online, чтобы понять: с тех пор мало что изменилось. Компания AOL, несмотря на весь ущерб, который наносят ее пользователям путаница и интерфейсные налоги, продолжает оставаться одним из главных поставщиков мусорных окон на планете.

Не существует способа отобразить связи между множеством окон, поэтому, пожалуйста, не создавайте множество окон. Эта проблема встает особенно остро при работе с Visual Basic, где так легко создавать **формы**. Формы – это независимые окна верхнего уровня. По своему поведению они являются немодальными диалоговыми окнами. Реализация приложения в виде коллекции немодальных диалоговых окон – сомнительный подход, который никогда не был особенно популярен, пока Visual Basic не сделал их создание настолько простым. Однако простота реализации решения еще не делает его удачным. Каждое новое окно взваливает на плечи пользователя дополнительный интерфейсный налог, связанный с управлением окнами. При ежедневном использовании программы совокупный размер этих налогов может разрастись до невероятных размеров.

Один программист на Visual Basic как-то жаловался, что спроектировать его программу было особенно трудно, поскольку она включала в себя 57 форм. Ни одна программа с 57 формами не может использоваться эффективно. Каждая из этих форм может быть прекрасна сама по себе, но все вместе – это, мягко говоря, перебор. Ситуация чем-то похожа на дегустирование 57 марочных «бордо» за один раз или на поездку на 57 спортивных автомобилях за одну субботу.

Состояния окон

Программисты обычно называют главное окно приложения **окном верхнего уровня**. Каждое окно верхнего уровня может находиться в одном из трех состояний (в Windows, Mac OS X и некоторых графических оболочках UNIX-систем) в зависимости от того, как они запрограммированы. Странно, но только два из этих трех состояний получили собственные названия от Microsoft: **свернутое** и **развернутое**.

В графических оболочках UNIX, таких как Motif, и в старых версиях Microsoft Windows (до Windows 95) **свернутые** окна обычно были представлены квадратными пиктограммами (размер которых, как правило, немного превышал размер стандартных пиктограмм рабочего стола), размещенными на рабочем столе. Начиная с Windows 95 сворачивание окна уменьшает его до кнопки на панели задач. В системе Mac OS X сворачивание окна убирает его в Dock.

Развернутые окна занимают весь экран, скрывая все прочие объекты на рабочем столе. Apple называет кнопку перехода в это состояние кнопкой Масштабировать (Zoom), но, похоже, не имеет отдельного термина для обозначения стандартного состояния. И продуктам Microsoft, и продуктам Apple каким-то образом удается избегать прямых отсылок к третьему состоянию, и лишь в системном меню Microsoft есть намек на его название (щелкните в левом верхнем углу заголовка окна, чтобы открыть это меню) – там способ перехода в это состояние описывается глаголом Восстановить (Restore). Эта функция переводит

развернутое окно верхнего уровня в то самое *третье* состояние. В целях сохранения рассудка мы будем называть это третье состояние **свободным**, хотя его называют также **восстановленным**.

Свободное состояние является промежуточным между пиктографическим представлением окна и развернутым состоянием, когда окно занимает весь экран. Окно, находящееся в свободном состоянии, делит экран с пиктограммами и другими свободными окнами. Свободные окна могут быть перекрывающимися или уложенными плиткой.

Во времена Windows 1.0 свернутые и развернутые окна назывались **миниатюрой** (iconized) и **полномасштабным окном** (zoomed) – это были более наглядные и выразительные варианты. Корпорация IBM, пребывавшая тогда в дружеских отношениях с Microsoft, потребовала перейти на корпоративный жаргон, ошибочно предполагая, что это должно упростить жизнь американским руководящим работникам. Так прижились более слабые варианты.

Развернутое состояние окна является наиболее естественным для монопольного приложения. Нет особых причин держать такую программу в свободном режиме – разве что для поддержки переключения между программами или перетаскивания данных между ними или документами (последнюю задачу можно вполне успешно решать с помощью элемента управления на панели инструментов). В то же время некоторые временные приложения, вроде Проводника (Windows Explorer), Калькулятора и iTunes, вполне естественно работают в свободном режиме.

MDI против SDI

Около 20 лет назад компания Microsoft начала активно пропагандировать новый метод организации функций в Windows-приложениях. Компания назвала его **многодокументным интерфейсом**, или **MDI (multiple document interface)**. Такой интерфейс отвечал потребностям определенного рода приложений, а именно тех, которые работали одновременно с несколькими однотипными документами. Известные примеры – это Word и Excel.

Microsoft подкрепила новый стандарт кодом, встроенным прямо в операционную систему, так что становление MDI как стандарта стало неизбежным. На протяжении определенного периода времени в конце 80-х и начале 90-х некоторые представители Microsoft считали MDI своего рода универсальным патентованным лекарством от всех болезней пользовательских интерфейсов. Его свободно выписывали для лечения всех видов заболеваний.

Сейчас Microsoft отказалась от MDI и выбрала **интерфейс одного документа**, или **SDI (single document interface)**. Похоже, целительная сила MDI была преувеличена.

Если вы хотите скопировать ячейку из одной электронной таблицы в другую, открывать и закрывать по очереди обе таблицы – слишком утомительное занятие. Было бы гораздо удобнее держать обе таблицы открытыми одновременно. Есть два способа достичь этого: запустить одну программу с несколькими таблицами внутри нее или открыть несколько программ, каждую со своим экземпляром таблицы. Второй вариант технически совершеннее, но более требователен к компьютерным ресурсам.

На заре существования Windows компания Microsoft выбрала первый вариант по очень простой практической причине – ради экономии ресурсов компьютера. Одна программа с несколькими таблицами потребляла меньше байтов и циклов процессора, чем несколько экземпляров той же программы, а производительность компьютеров в то время была серьезной проблемой.

К несчастью, модель «один экземпляр программы, несколько документов» нарушала изначальное фундаментальное правило проектирования, установленное в Windows: в любой момент времени активным может быть только одно окно. Задача же требовала того, чтобы в некоторый момент времени было активно одно окно программы и одновременно внутри него – окно с документом. Идея MDI и была трюком, который решал эту задачу.

С момента превращения MDI в стандарт появились два новых обстоятельства. Во-первых, сбитые с толку программисты стали злоупотреблять этим подходом – хотя и с благими намерениями. Во-вторых, наши компьютеры стали гораздо мощнее – они достигли уровня, при котором несколько копий программы, каждая со своим документом, перестали вызывать проблемы производительности. Поэтому Microsoft ясно дала понять, что модель MDI если и не отжила свое, то по крайней мере больше не является хорошим тоном.

Если не принимать во внимание очередной ветер перемен, гуляющий по Microsoft, следует признать, что у модели MDI есть свои достоинства, при условии, что ею не злоупотребляют. В частности, она полезна, когда пользователю требуется работать с несколькими связанными представлениями информации в единой среде. И хотя нет ничего ужасного в переключении между экземплярами Word при работе с различными документами, вряд ли вам понравится, если агент по продажам будет вынужден переключаться между различными экземплярами корпоративной системы, чтобы взглянуть на счет и историю платежей поставщика. Эти вещи используются вместе и служат одной цели, а потому и представлены должны быть совместно.

Разумеется, здесь также возможны злоупотребления. Программа CompuServe Navigator предлагает не менее десятка различных типов окон документов, крайне затрудняя понимание происходящего (AOL грешит этим и по сей день). Чтобы достичь своей цели в ERP-системе SAP R/3, пользователю иногда приходится открывать десять окон. По

мере того как функции теряют четкие границы и навигация становится все более и более тяжеловесной, усиливается путаница. При выборе различных типов документов приложению приходится соответствующим образом менять меню. Пользователи же полагаются на то, что статичность меню поможет им ориентироваться на экране. Изменение содержания меню лишает пользователей этой опоры. Более того, все, что говорилось выше про сворачивание, разворачивание и свободные режимы окон, вдвойне актуально для окон документов в MDI-приложении. Вынуждать пользователя управлять маленькими окнами внутри большого окна – образец гнуснейших интерфейсных налогов. Гораздо лучше явно переходить из одного окна в другое. Переход из одной развернутой таблицы в другую развернутую таблицу – это мощный и эффективный прием.

Сегодня практически нет никакой разницы между MDI и SDI в реализации Microsoft. В большинстве приложений Microsoft для переключения между таблицами можно пользоваться как оконным меню, так и панелью задач, с помощью той же панели задач переключаясь с программы Excel на Word.

21

Элементы управления

Элементы управления – это доступные для манипулирования самодостаточные экранные объекты, посредством которых люди взаимодействуют с цифровыми продуктами. Элементы управления (controls, известные также под названиями widgets[1], gadgets[2] и gizmos[3]) – это базовые строительные блоки графического пользовательского интерфейса.

Рассматривая элементы управления в свете целей пользователя, их можно разбить на четыре основные категории: **командные элементы управления**, применяемые для выполнения функций, **элементы выбора**, позволяющие выбирать данные или настройки, **элементы ввода**, применяемые для ввода данных, и **элементы отображения**, используемые для наглядного непосредственного манипулирования. Некоторые элементы управления сочетают в себе свойства более чем одной категории.

Большинство знакомых нам элементов управления входят в состав Windows, Mac OS и других распространенных оконных интерфейсов. Этот набор готовых к употреблению элементов управления всегда имел ограниченные возможности и пределы применения.

Нет окнам, перегруженным элементами управления!

В большинстве оконных систем создать диалоговое окно проще простого. Среди стандартных возможностей диалогового окна – автоматическое размещение элементов управления. К сожалению, это дает разра-

---

[1] Сокращенное windows gadgets – оконные приспособления. – *Примеч. перев.*

[2] Устройства, приспособления. – *Примеч. перев.*

[3] Буквально – штуковины. – *Примеч. перев.*

ботчикам возможность легко создавать пользовательские интерфейсы, основанные преимущественно на перегруженных управляющими элементами диалоговых окнах. Гораздо сложнее создавать наглядные интерфейсы с применением идиом непосредственного манипулирования, соответствующих ментальным моделям пользователей и их рабочим процессам. Как следствие, существующая литература рассказывает в основном о «патентованных» элементах управления, игнорируя прочие подходы. Но мы-то знаем, что успешное проектирование пользовательского интерфейса состоит отнюдь *не* в насыщении диалоговых окон элементами управления. (Более подробно о силе пользовательских интерфейсов, основанных на непосредственном манипулировании, читайте в главе 19. Диалоговые окна более подробно обсуждаются в главе 24.)

Большое количество диалоговых окон, перегруженных элементами управления, не гарантирует качество пользовательского интерфейса.

Поймите нас правильно: мы вовсе не предлагаем избегать стандартных элементов управления. Однако, хотя стандартные элементы управления могут гарантировать простоту реализации, они абсолютно не гарантируют удобства в использовании. Как любые другие составляющие качественного пользовательского интерфейса, элементы управления должны применяться благоразумно и сообразно ситуации.

Теперь более подробно рассмотрим каждый из четырех типов элементов управления: командные элементы, элементы выбора, элементы ввода и элементы отображения.

Командные элементы управления

Имеется язык взаимодействия человека и компьютера, который состоит из существительных (иногда называемых объектами), прилагательных, глаголов и наречий. Отдавая команду, мы определяем глагол – действие, которое содержится в предложении. Описывая предмет, на который влияет действие, мы определяем имя существительное. Иногда мы выбираем имя существительное из имеющегося списка, а иногда вводим новое. Мы можем уточнять существительное и глагол прилагательными и наречиями соответственно.

Элемент управления, соответствующий глаголу, называется **командным**, поскольку связан с немедленным действием. Командные элементы управления выполняют действия, причем делают это немедленно. Элементы меню (которые мы обсудим в главе 22) также являются командными идиомами. В мире элементов управления квинтэссенцией командной идиомы является кнопка. Она же, по сути дела, – единственная идиома, хотя и обладает множеством вариантов отображения. Щелкните по кнопке – и связанное с ней действие-глагол будет немедленно выполнено.

Кнопки

Кнопки чаще всего легко опознаются благодаря их псевдотрехмерности (рис. 21.1). Если этот элемент управления прямоугольный (а иногда овальный) и кажется выпуклым (благодаря тени на правой и нижней гранях и подсветке верхней и левой граней), то его ожидаемое назначение указывает на командный характер элемента управления. Он выполнит действие, как только пользователь щелкнет по нему курсором мыши и отпустит кнопку. В диалоговых окнах зачастую особым образом выделяется **кнопка по умолчанию**, соответствующая наиболее разумному в данной ситуации действию.

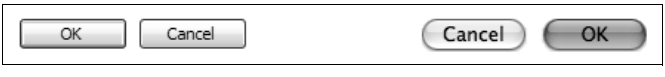


Рис. 21.1. *Стандартные кнопки в Microsoft Windows (слева) и Mac OS X (справа). Применение теней и подсветки позволяет создать эффект объема, который наделяет кнопки свойством «нажимаемости»*

Кнопка – вероятно, самый привлекательный с визуальной точки зрения элемент управления в арсенале проектировщика. Неудивительно, что в пользовательском интерфейсе встречается такое разнообразие кнопок. Именно ожидаемое назначение современных псевдотрехмерных кнопок сделало их столь популярными. Это же здорово – так почему бы *не использовать* их повсеместно?

Облик кнопки и ее отзывчивость подсказывают, что на нее можно нажать, и тем самым характеризуют ее ожидаемое назначение. Когда пользователь наводит курсор на кнопку и нажимает клавишу мыши, кнопка на экране изменяет свой внешний вид, превращаясь из выпуклой в утопленную и демонстрируя тем самым, что она активизирована. Это пример динамической визуальной подсказки, о которой мы говорили в главе 19. Плохо спроектированные программы и большинство веб-сайтов предоставляют пользователю кнопки, которые изображены на экране как кнопки, но при щелчке не меняются. Это облегчает жизнь разработчикам (особенно в среде Всемирной паутины), но сильно дезориентирует пользователя, который мысленно задается вопросом: «А было ли действие выполнено фактически?» Пользователь ожидает увидеть реакцию кнопки – та должна стать утопленной, и ваша задача – удовлетворить его ожидания.

Кнопки-значки (butcons)

С выходом Windows 3.0 была представлена идиома **панели инструментов** (о которой мы подробно поговорим в главе 23), ставшая стандартом де-факто, столь же хорошо знакомым, как и строка меню. Кнопки, традиционно расположенные в диалоговых окнах, были адаптированы для размещения на панели инструментов. В процессе развития

функции панели инструментов, ее роль в приложении и богатство визуального представления значительно расширились.

В диалоговых окнах кнопка имеет прямоугольную форму (на компьютерах Mac – со скругленными углами) и повсеместно сопровождается текстовой надписью. При переселении на панель инструментов кнопка стала квадратной, потеряла текстовую надпись и обзавелась пиктограммой – пояснением в виде графического значка. Таким образом родилась **кнопка-значок**, то есть кнопка и значок одновременно (рис. 21.2). В Windows 98 кнопки-значки (они же – кнопки панели инструментов) продолжили развиваться и утратили рельефность (она стала возникать лишь тогда, когда пользователь взаимодействует с кнопкой) в целях уменьшения визуального шума, создаваемого загроможденными инструментальными панелями. К сожалению, это осложнило восприятие идиомы для новичков. Начиная с Windows 2000 кнопка-значок демонстрирует ожидаемое назначение кнопки, только когда указатель мыши находится над ней.

Рис. 21.2. *Кнопки-значки в Microsoft Office 2003. Слева – примеры из системы Windows, справа – те же примеры из пакета Office для системы OS X. Обратите внимание, что ни один элемент не отображается в виде кнопки, пока курсор мыши не окажется над ним*

В теории кнопки-значки очень удобны: они постоянно на виду и для взаимодействия с ними не требуется так много времени или ловкости, как для взаимодействия с раскрывающимся меню. Поскольку они постоянно видны, то легко запоминаются, особенно в монопольных приложениях. Преимущества кнопки-значка трудно отделить от преимуществ инструментальной панели – они неразрывно связаны между собой. Проблема, преследующая кнопки-значки, связана не с их кнопочной сущностью, а со значками. У большинства пользователей не возникает вопросов относительно ожидаемого назначения кнопки. Проблема в том, что редко бывает понятным изображение на кнопке.

Пиктограммы вообще трудно поддаются однозначной расшифровке с первого взгляда, но в этом помогают всплывающие подсказки. Тем не менее для того, чтобы создать узнаваемую и запоминающуюся пиктограмму, не заставляющую пользователей *каждый раз* при взгляде на кнопку обращаться к всплывающей подсказке, требуются усилия опытного и талантливого дизайнера. Хорошую пиктограмму пользователь изучит и запомнит, если часто пользуется соответствующей функцией. Именно такое поведение обычно демонстрируют середняки и подготовленные пользователи.

Но даже лучший в мире автор пиктограмм не сможет без усилий создать систему значков, понятную новичкам без всяких подписей. Раз-

умеется, всплывающие подсказки придут им на помощь, однако чрезвычайно неудобно останавливать курсор над каждой пиктограммой и ждать, когда появится всплывающая подсказка для кнопки-значка. В подобных случаях четкие названия пунктов меню предлагают гораздо более удобный командный вектор. Более подробно о меню мы поговорим в главе 22. Дополнительные материалы по кнопкам-значкам, панелям инструментов и всплывающим подсказкам содержатся в главе 23.

Гиперссылки

Гиперссылки, или просто **ссылки**, – принятый в среде Всемирной паутины способ навигации, который перекочевал уже во множество самых различных приложений. Ссылка обычно выглядит как подчеркнутый текст (хотя, разумеется, изображение тоже может быть ссылкой) и является командным элементом управления для навигационных целей. Это простая, прямолинейная, полезная идиома взаимодействия. Если пользователя интересует подчеркнутое слово, он может щелкнуть по этому слову и попасть на новую страницу, содержащую дополнительную информацию.

К сожалению, успех и полезность этой идиомы не совсем правильно толкуются некоторыми проектировщиками, которым кажется, что замена многих распространенных командных элементов управления, таких как кнопки и кнопки-значки, подчеркнутыми словами автоматически приводит к созданию более удобных и качественных пользовательских интерфейсов. Редко когда это так. Поскольку большинство пользователей знает, что ссылки – идиома навигации, они путаются и сбиваются с толку, если щелчок по ссылке приводит к выполнению действия. В общем случае следует применять ссылки для навигации по информации, а кнопки или кнопки-значки – для выполнения других действий или функций.

Используйте ссылки для навигации, а кнопки и кнопки-значки – для выполнения действий.

Элементы управления выбором

Поскольку командный элемент является глаголом, ему требуется имя существительное, обозначающее объект, над которым выполняется действие. Элементы выбора и ввода данных – это два типа элементов управления, которые используются для определения существительных (в дополнение к идиомам непосредственного манипулирования). **Элемент выбора** позволяет пользователю из группы допустимых объектов выбрать тот объект, с которым будет совершено действие. Элементы выбора применяются также для действий по настройке – в случае идиомы непосредственного манипулирования существительное

может быть уже определено, а элемент выбора позволяет определить прилагательное или наречие. Примеры распространенных элементов выбора: флажки, списки, раскрывающиеся списки.

Раньше использование элементов управления выбором не приводило к немедленному выполнению действий – требовалась еще и активация командного элемента. Но сегодня это не всегда так. В одних случаях (например, когда раскрывающийся список применяется для навигации на веб-странице) такое поведение элемента может дезориентировать пользователей. В других случаях (когда раскрывающийся список используется для изменения размера шрифта в текстовом редакторе) это может быть вполне естественным.

Как это часто бывает в проектировании взаимодействия, оба подхода имеют свои плюсы и минусы. Если желательно дать пользователю возможность несколько раз осуществить выбор перед выполнением действия, следует создать явный командный элемент управления (то есть кнопку). Если же пользователю полезно сразу видеть результат своих действий и эти действия легко отменить, совершенно разумно сделать так, чтобы элемент выбора играл также и роль командного элемента.

Флажки

Флажок – одна из самых первых визуальных идиом управления, по сей день остающаяся наиболее популярным элементом представления выбора из двух взаимоисключающих вариантов (рис. 21.3). Флажок имеет ярко выраженное ожидаемое назначение, побуждающее щелкать по нему. Флажок побуждает к выполнению щелчка благодаря подсветке при наведении указателя мыши и трехмерному визуальному «углублению». Щелкнув по флажку, пользователь видит, что появилась галочка: можно считать, что он узнал все, что должен знать для того, чтобы работать с этим элементом: «Щелкните, чтобы поставить галочку, щелкните еще раз, чтобы убрать галочку». Флажок прост, нагляден и изящен.

Однако флажок – элемент управления, основанный прежде всего на тексте. Это знакомая и весьма эффективная идиома, имеющая те же достоинства и недостатки, что и меню. Качественный текст может исключить возможность неоднозначного толкования флажка. Однако

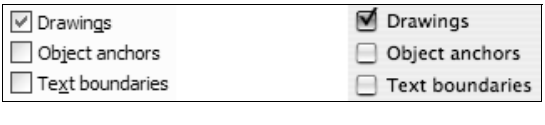


Рис. 21.3. *Вот стандартные флажки из Microsoft Windows (слева) и Mac OS X (справа). И снова применение теней и подсветки создает эффект объема и наделяет флажок отличительным свойством, побуждающим к нажатию. Обратите внимание, что флажки Windows имеют более типичный вогнутый вид, тогда как флажки OS X, наоборот, выпуклые*

этот же поясняющий текст вынуждает пользователя замедляться для прочтения, а также занимает значительное экранное пространство.

Традиционно флажки имеют квадратную форму. Пользователи распознают визуальные объекты по форме, и квадратность флажков – важный стандарт. В квадратной форме самой по себе нет ничего плохого или хорошего, просто изначально была выбрана именно форма квадрата, и большое число пользователей уже научились узнавать ее. Нет разумных причин отступать от этой формы. Не делайте флажки ромбовидными или круглыми, независимо от того, что говорят специалисты по маркетингу или дизайнеры.

Существует, однако, возможность сделать функцию флажка более наглядной, применив в качестве основы кнопки-значки. Кнопка превратилась в кнопку-значок путем замены текста пиктограммой и размещения ее на панели инструментов. На этом метаморфозы кнопки не закончились – и ее функциональность была расширена возможностью фиксироваться в нажатом состоянии после щелчка по ней и возвращаться в исходное состояние после повторного нажатия. Так появилась **бинарная кнопка-значок**, или **выключатель** (рис. 21.4). Состояние выключателя меняется не на мгновение, а фиксируется до следующего щелчка. Характер элемента управления изменился настолько, что он перешел в другую категорию – превратился из командного в элемент управления выбором.

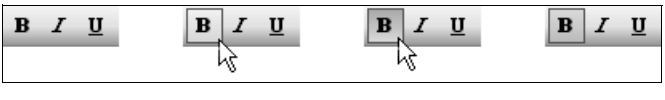


Рис. 21.4. *Перед вами бинарные кнопки-значки в различных состояниях: состояние по умолчанию, состояние при наведении курсора, фиксированное состояние (Microsoft Office 2003)*

Выключатель широко используется вместо флажка как идиома бинарного выбора. Он особенно полезен в немодальных взаимодействиях, которые не требуют переключения внимания пользователя для принятия решения.

Бинарные кнопки-значки более экономно расходуют экранное пространство, нежели флажки: они занимают меньше места, потому что их назначение описывается не посредством текста, а с помощью визуальных средств. Разумеется, это означает, что им присущ тот же недостаток, что и обычным кнопкам-значкам, – неоднозначность пиктограмм. Здесь на помощь опять приходят всплывающие подсказки. Эти крошечные всплывающие окна дают нам достаточный объем информации, который устраняет неоднозначность, не занимая слишком много пикселов на постоянной основе.

Триггеры: опасная идиома выбора

Кнопки-триггеры – это разновидность элемента управления, призванного экономить экранное пространство, к сожалению, достигающего этой экономии ценой значительной дезориентации пользователя. С глаголом, относящимся к кнопке-триггеру, связано несколько состояний, соответствующих состоянию кнопки. Классический пример – размещение на одной кнопке функций воспроизведения и паузы для музыкального проигрывателя: на кнопке нарисован универсальный треугольник, обозначающий воспроизведение, а если по ней щелкнуть, треугольник сменяется универсальной пиктограммой паузы – двумя вертикальными полосками.

Этот элемент управления сообщает, что по нему можно щелкнуть, и, отображая пиктограмму воспроизведения, утверждает, что после щелчка начнется воспроизведение музыки. После щелчка кнопка изменяется – теперь она отображает пиктограмму паузы, указывая, что после щелчка воспроизведение будет прервано. Подводным камнем такого подхода является то, что элемент управления можно ошибочно посчитать индикатором состояния проигрывателя («на паузе» или «идет воспроизведение»). Это означает, что существуют две разумные, но противоречивые интерпретации пиктограмм на кнопке. Элемент управления может служить либо индикатором состояния, либо кнопкой переключения состояний, но не тем и другим одновременно (рис. 21.5).

Рис. 21.5. *Кнопки-триггеры очень эффективны. Они занимают мало места, объединяя два взаимоисключающих состояния. Проблема триггерных элементов управления в том, что они не в состоянии выполнять вторую обязанность любого элемента управления, а именно – информировать пользователя о своем текущем состоянии. Если кнопка говорит «Включено», когда находится в состоянии «выключено», то непонятно, в каком же состоянии находится кнопка. Если кнопка говорит «Выключено», когда находится в состоянии «выключено», тогда возникает вопрос, где искать кнопку «Включить»? Не используйте эти элементы управления*

Решение этой проблемы состоит в том, чтобы или обстоятельно объяснить назначение элемента управления глаголом либо фразой на кнопке (Воспроизведение или Пауза), или же – что лучше – использовать совершенно иной подход, например заменить элемент управления двумя кнопками. Минусом последнего решения является дополнительный расход экранного пространства.

Радиокнопки

Радиокнопка внешне похожа на флажок (рис. 21.6). Ее название говорит само за себя. Когда впервые в автомобилях появились радиоприемники, мы обнаружили, что настраивать радиоприемник, вращая ручку настройки во время вождения, далеко не безопасно. Поэтому автомобильные радиоприемники стали снабжаться новомодной панелью с полудюжиной хромированных кнопок, каждая из которых настраивала приемник на конкретную радиовещательную станцию. Так появилась возможность переключать радиоприемник на любимую волну простым нажатием кнопки, не отрывая взгляда от дороги. Это очень мощная идиома, и для нее по сей день находится масса применений в проектировании взаимодействия.

Рис. 21.6. *Слева изображены радиокнопки из Microsoft Windows XP. Справа – радиокнопки из Mac OS X*

Радиокнопки являются **взаимоисключающими**, то есть выбор одного из вариантов автоматически аннулирует предыдущий выбор. В каждый момент времени может быть выбрана только одна кнопка.

Как следствие такой архитектуры, радиокнопки всегда объединяются в группы из двух или более радиокнопок, причем в каждой группе одна радиокнопка всегда выбрана. Поведение единственной радиокнопки в группе не определено – она должна действовать как обычный флажок. (В такой ситуации следует использовать флажок или подобные ему невзаимоисключающие элементы управления.)

В смысле затрат экранного пространства радиокнопки могут быть еще менее эффективными, чем флажки. Сама по себе радиокнопка занимает столько же места, сколько флажок, но поскольку радиокнопки имеют смысл только в группе, то занимаемая ими площадь всегда больше. В некоторых случаях такой расход экранного пространства вполне оправдан, особенно там, где важно всегда показывать пользователю полный набор доступных вариантов. Радиокнопки хорошо подходят для целей обучения, то есть их присутствие совершенно оправданно в нечасто используемых диалоговых окнах, однако для основного интерфейса монопольного приложения, с которым пользователи работают повседневно, как правило, больше подходят раскрывающиеся списки.

Радиокнопки всегда круглые по той же причине, по которой флажки – всегда квадратные: именно такими они были изначально (за исключением среды Motif, где радиокнопки имели форму ромбов, но это решение, похоже, не прижилось).

Как можно догадаться, кнопка-значок преобразовала радиокнопки так же, как флажки, заменив их в основном интерфейсе приложения. Если две или более бинарные кнопки объединены схемой взаимного исключения – так, чтобы в каждый момент могла быть включена лишь одна из них, – они ведут себя точно так же, как радиокнопки. Так образуются **радиокнопки со значками**.

Они действуют аналогично радиокнопкам: одна кнопка всегда нажата, и всякий раз, когда нажимается другая кнопка, первая возвращается в ненажатое состояние. Элементы управления выравниванием текста в Word – прекрасный пример радиокнопок со значками (рис. 21.7).

Рис. 21.7. *Элементы управления выравниванием текста в Word представляют собой группу радиокнопок со значками и действуют на манер обычных радиокнопок. Одна из кнопок всегда находится в нажатом состоянии, а когда нажимается другая кнопка, первая возвращается в обычное, ненажатое состояние. Эта идиома позволяет значительно экономить экранное пространство и прекрасно подходит для переключения между часто используемыми режимами*

Как и все идиомы с кнопками-значками, радиокнопки со значками очень эффективно используют экранное пространство, позволяя опытным пользователям полагаться на распознавание образов при их идентификации и напоминая неопытным пользователям о своем предназначении посредством всплывающих подсказок. Новые пользователи либо будут достаточно умны, чтобы обучиться на всплывающих подсказках, либо будут учиться более медленно, но не менее надежно на основе других параллельных обучающих командных векторов.

Комбо-кнопки

Разновидность радиокнопки со значком – **комбо-кнопка**, названная так из-за сходства с элементом управления типа «комбо-список» (combo box) (рис. 21.8). Обычно она выглядит как кнопка-значок с небольшой стрелкой справа, направленной вниз (в Windows), но если нажать

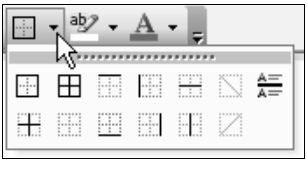


Рис. 21.8. *Эта комбо-кнопка из Microsoft Office 2003 представляет группу кнопок-значков с фиксируемым состоянием, которая ведет себя как поле с раскрывающимся списком*

на стрелку и удерживать ее в нажатом состоянии, разворачивается меню из нескольких кнопок-значков, среди которых пользователь может выбрать один вариант. Теперь выбранная кнопка-значок появляется на рабочем столе рядом со стрелкой. Щелчок по собственно кнопке-значку активирует команду, обозначенную текущим состоянием комбо-кнопки. Как и команды меню, кнопки-значки должны раскрыться, если пользователь щелкнет по стрелке, а затем при нажатой кнопке мыши наведет курсор на нужную кнопку и отпустит кнопку мыши.

Существует разновидность комбо-кнопки, для которой роль отдельно вынесенной стрелки (используемой на инструментальных панелях Microsoft) играет расположенное в нижнем правом углу самой кнопки изображение маленького треугольника, указывающего вниз и вправо. В продуктах от Adobe эта разновидность кнопок используется в палитрах: чтобы получить выпадающее меню (которое в продуктах от Adobe разворачивается не ниже, а правее кнопки, как показано на рис. 21.9), необходимо нажать на саму кнопку и удерживать ее в нажатом состоянии. Вы можете разнообразить эту идиому в достаточно больших пределах, и проектировщики, творчески подходящие к вопросу создания программ, именно этим и занимаются в нескончаемых попытках уместить как можно больше функций на извечно недостаточно больших экранах.

Еще одну вариацию можно обнаружить в Microsoft Word, где кнопки-значки для выбора цвета фона и текста больше напоминают неболь-

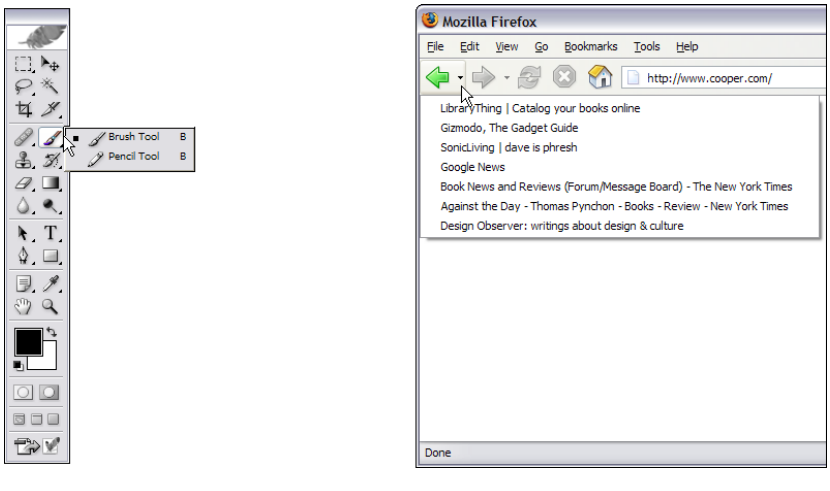


Рис. 21.9. *Эти комбо-кнопки из Adobe Photoshop (слева) и Mozilla Firefox (справа) демонстрируют разнообразные варианты применения идиомы. В Photoshop щелчок по комбо-кнопке позволяет выбирать между различными модальными курсорными инструментами, а в Firefox – выбирать уже посещенную ранее страницу, к которой необходимо вернуться. В первом случае выполняется настройка пользовательского интерфейса, а во втором – выполняется конкретное действие*

шие палитры, нежели наборы кнопок. Как можно видеть из рис. 21.9, эти меню способны представлять значительный объем информации и богатый набор функций в весьма компактном виде. Такая возможность определенно предназначена для середняков и опытных пользователей (особенно для тех, кто хорошо владеет мышью) – но никак не для новичков. Однако, если пользователь хотя бы немного знаком с имеющимися инструментами, то при самостоятельном обнаружении он быстро осваивает эту идиому. Это превосходная идиома управления для монопольных программ, с которыми пользователи взаимодействуют подолгу. Чтобы работать с меню, содержащим мелкие элементы управления, требуется достаточно высокая ловкость рук, но такая работа выполняется намного быстрее, чем вызов раздела главного меню, выбор пункта меню, ожидание появления диалогового окна, выбор цвета в диалоговом окне и подтверждающий щелчок по кнопке OK.

Списки

Элементы управления типа «список» позволяют осуществлять выбор из конечного множества текстовых строк, каждая из которых представляет команду, объект или признак. Эти элементы управления иногда называют **списками выбора**, так как они содержат списки элементов, из которых пользователь может производить выбор. Они также известны просто как **окно списка** или **представление в виде списка**, в зависимости от того, о какой платформе и какой разновидности списков идет речь. Подобно радиокнопкам, списки – мощный инструмент, упрощающий взаимодействие за счет устранения возможности неправильного выбора.

Списки – это небольшие текстовые области с вертикальной полосой прокрутки по правому краю (см. рис. 21.10). Приложение отображает объекты как отдельные строки текста в области списка, а полоса прокрутки перемещает их вверх или вниз. Пользователь может выбрать единственную строку текста, щелкнув по ней мышью. Некоторые списки позволяют выбирать сразу несколько строк; обычно для этого приходится щелкать по элементам списка, удерживая клавишу <Shift> или <Ctrl>.

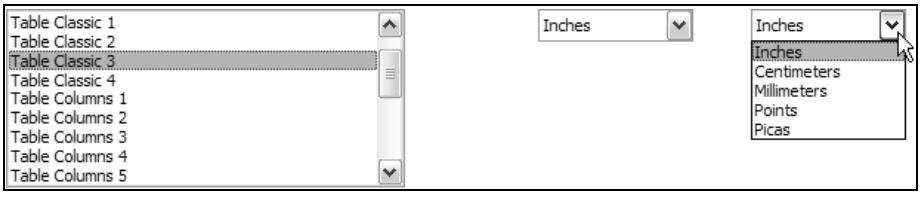
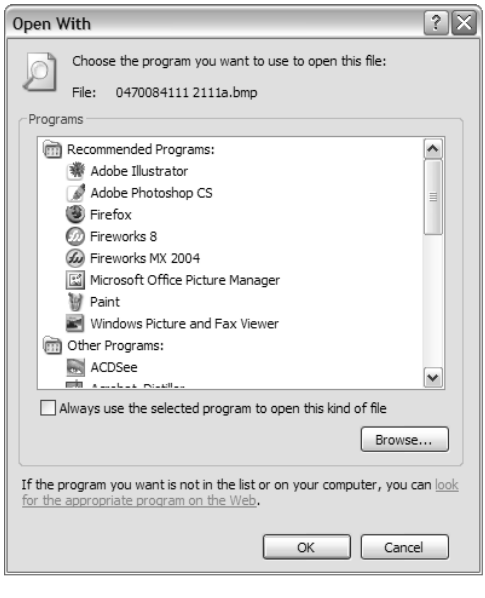


Рис. 21.10. *Слева – стандартный список из Windows. Изображения справа показывают различные состояния раскрывающегося списка*

Раскрывающийся список – повсеместно встречающийся вариант обычного списка. Он показывает лишь выбранный элемент в одну строку, но если нажать на стрелку, открываются другие варианты выбора (рис. 21.10)

Ранние списки могли отображать только текст – и это до сих пор сказывается на характере их поведения. Список, состоящий лишь из строк в чистом виде, без поддержки графических символов, – все равно что безводная пустыня перед путником. Однако с выходом Windows 95 появилась возможность сопровождать каждую строку текста пиктограммой – и для этого даже не нужно писать дополнительный код. Этой цели послужил элемент управления **представление в виде списка** (listview). Такая возможность весьма полезна – существует множество ситуаций, когда можно облегчить пользователю жизнь, располагая графические идентификаторы рядом со строками важных вариантов выбора (рис. 21.11). В соответствии с новыми веяниями элементы списка все чаще используются в качестве средства предварительного просмотра. Этот подход обычно применяется в тех случаях, когда элемент управления отвечает одновременно за выбор и выполнение команды – как, к примеру, элемент выбора стиля в Microsoft Word (рис. 21.11).



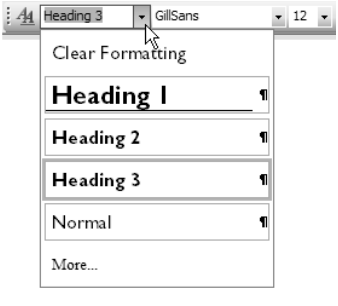


Рис. 21.11. *Слева – элемент управления типа «список» с пиктограммами в Windows XP, позволяющий пользователям визуально находить нужное приложение. Справа – раскрывающийся список стилей из Office 2007. Здесь элементы списка обеспечивают предварительный просмотр результата выбора*

Выделяйте наиболее важные элементы списков с помощью пиктограмм.

Представление в виде списка, соответствуя своему названию, хорошо подходит для отображения списков элементов и позволяет пользователю выбрать один или несколько элементов. Кроме того, это хорошая идиома для создания источника перетаскиваемых элементов (за исключением случая раскрывающегося списка, когда это просто неудобно). Если строки можно перетаскивать внутри представления в виде списка, мы получаем отличное средство упорядочивания строк (см. раздел «Упорядоченные списки» далее в этой же главе).

Отметки

Вообще говоря, пользователи выбирают элементы из списков в качестве входных аргументов некоторой функции, например осуществляют выбор названия шрифта из перечня доступных. Поведение элемента выбора в списке вполне стандартно, допускается применение клавиатурных сокращений; сам выбранный элемент помечается выделяющей рамкой и цветовой подсветкой.

Однако иногда списки используются для выбора сразу нескольких элементов, что может послужить источником затруднений. Идиома выбора из списка прекрасна для выбора единственной строки, но для множественного выбора подходит плохо. Вообще говоря, множественный выбор отдельных объектов адекватно работает тогда, когда на экране виден весь перечень, как, например, в случае с пиктограммами на рабочем столе. Когда одновременно выбраны две или более пиктограммы, вы ясно видите это, потому что все пиктограммы находятся в поле зрения.

Но если перечень доступных дискретных элементов слишком велик, чтобы уместиться в одном окне, и для доступа к части из них необходимо осуществить прокрутку, идиома выбора немедленно становится громоздкой. Для списков данная ситуация является нормой. Обычно они функционируют в режиме выбора единственного элемента, когда выбор одного пункта аннулирует выбор, сделанный перед этим. В случае, когда необходимо реализовать множественный выбор, это приводит к тому, что пользователи, прокручивая список, перемещают выбранный элемент за пределы окна и, выбирая второй элемент, легко упускают из виду, что теперь выбор предыдущего элемента *аннулирован*, поскольку больше не могут видеть его.

Альтернатива оказывается ничем не лучше: элемент управления «список», запрограммированный без взаимоисключающего выбора, разрешает пользователям выделить такое число элементов списка, какое им требуется. Все работает вроде бы прекрасно: пользователь выделяет один элемент списка за другим – и каждый из них остается выделенным. Ложкой дегтя в данной ситуации становится отсутствие какой-

либо визуальной индикации того, что поведение элемента выбора отличается от обычного. Весьма вероятно, что пользователь выберет элемент, прокрутит его за пределы окна, затем обнаружит, что предпочтителен другой элемент, и выберет его, *ожидая, что выбор первого, невидимого элемента будет автоматически аннулирован* в соответствии со стандартом взаимоисключающего выбора. И вот вы оказываетесь перед нелегким выбором: какую половину ваших пользователей огорчить – первую или вторую? Ситуация тупиковая.

Если объект может быть прокручен за пределы окна, множественный выбор требует более качественной, более выраженной идиомы. Правильное решение состоит в использовании идиомы, отличной от идиомы простого выбора, то есть идиомы, которая имеет визуальные отличия. Но что это за идиома?

Так случилось, что у нас уже имеется другая известная идиома для выбора чего-либо – флажок. Флажки свидетельствуют о своем назначении весьма недвусмысленно и, как все хорошие идиомы, чрезвычайно просты в освоении. Флажки очень четко дистанцируются от любого намека на возможность взаимного исключения при выборе. Если добавить флажок к каждому элементу в нашем списке, пользователь не только четко увидит, какие позиции выбраны, а какие нет, но поймет также, что элементы списка не связаны друг с другом правилом взаимного исключения. Так мы убиваем сразу двух зайцев. Подобное использование флажка, в противовес множественному выбору, называют **отметкой** (см. пример на рис. 21.12).

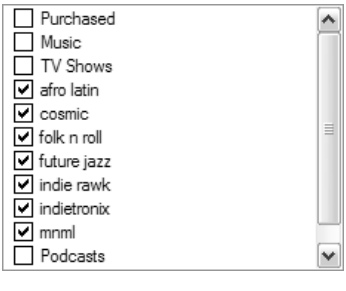


Рис. 21.12. *Выбор обычно является действием в рамках правил взаимного исключения. Когда возникает необходимость избавиться от этих правил и дать возможность множественного выбора, ситуацию может испортить прокручивание строк списка за пределы видимости. Решением проблемы является отметка. Флажок у каждого элемента списка как средство выделения позволяет пользователю уверенно обозначать свой выбор. Флажки – идиома, не связанная правилами взаимного исключения и очень хорошо знакомая пользователям. Они моментально схватывают ее смысл*

Перетаскивание содержимого списка

Элементы управления списками могут выступать в роли палитр объектов для использования в идиомах непосредственного манипулирования. Если бы список был, скажем, частью программы для генерации отчетов, то можно было бы, щелкнув по элементу списка, перетащить его в область отчета, чтобы добавить в отчет указанное поле. Эта операция не является выбором в обычном смысле, поскольку она полностью самодостаточна. Без сомнения, многие программы стали бы удобнее, если бы использовали списки, поддерживающие перетаскивание.

Перетаскиваемые элементы помогают пользователям формировать наборы объектов. Два смежных списка, один из которых перечисляет все доступные элементы, а другой – выбранные, стали обычной идиомой графического пользовательского интерфейса. Между такими списками помещается одна кнопка или пара разнонаправленных кнопок, с помощью которых производится перенос элементов между списками, как показано на рис. 21.13. Гораздо более приятно, когда есть идиома,

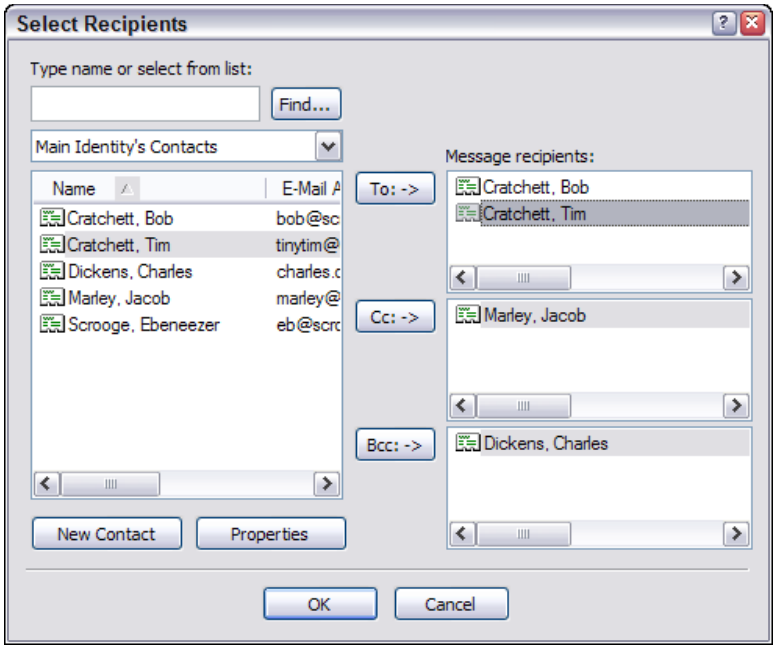


Рис. 21.13. *Это диалоговое окно Microsoft Outlook Express определенно стало бы более удобным, если бы позволяло перетаскивать контакт из списка слева в списки To (Кому), Cc (Копия) и Bcc (Скрытая копия) справа. Попутно обратите внимание на неудачное использование горизонтальных полос прокрутки во всех списках. Строка с полной информацией могла бы выводиться в виде всплывающей подсказки (особенно это касается левого списка). (Как вариант, диалоговое окно можно было бы расширить – ведь нет никаких разумных причин ограничивать его имеющимся размером.)*

позволяющая перетаскивать желаемый объект из одного списка в другой без необходимости что-то выбирать и вызывать какие-то функции.

Упорядоченные списки

Иногда возникает необходимость перенести элемент списка на другую позицию внутри того же списка. (Вообще говоря, эта потребность возникает намного чаще, чем предполагает большинство проектировщиков взаимодействия.) Многие программы предлагают средства автоматической сортировки для важных списков. Проводник, например, позволяет сортировать файлы по имени, типу, дате модификации и размеру. Это прекрасно, но не было бы еще лучше иметь возможность упорядочивать файлы по их важности? На практике программа вполне могла бы упорядочить их в соответствии с частотой обращения пользователя, но такой подход не всегда будет давать верные результаты. Учет даты и времени последнего обращения к файлам даст более близкие к истине результаты, но все еще недостаточно точные. (Microsoft применяет этот подход в средствах выбора шрифтов в некоторых приложениях, и в этих случаях алгоритм работает вполне прилично.) Почему бы не позволить пользователю переместить наиболее важные для него объекты в начало списка и сортировать их отдельно (в алфавитном или каком-либо другом порядке) в дополнение к сортировке, расположенной ниже оставшейся части списка? Например, вы могли бы пожелать переупорядочить список сотрудников вашего отдела в порядке убывания – по тому, где они сидят. Не существует автоматической функции, которая сделала бы это, – придется самостоятельно сортировать записи о сотрудниках до тех пор, пока не будет получен требуемый порядок следования. Но ведь именно такого рода настройка может потребоваться опытному пользователю после многих часов изучения приложения. Чтобы упорядочить каталог подобным образом, потребуются серьезные усилия, и программа *обязана* запоминать точный порядок следования от сеанса к сеансу – в противном случае ее способность изменять порядок следования объектов ничего не стоит.

Возможность перемещать элементы внутри списка из одного места в другое крайне полезна, но она требует реализации автоматической прокрутки (см. главу 19). Если мы выбрали элемент в списке, а место, в которое его требуется перенести, находится за пределами видимости, должна быть возможность прокрутить содержимое списка, не отпуская перемещаемый объект.

Горизонтальная прокрутка

Обычно в списках присутствует вертикальная полоса прокрутки, позволяющая перемещаться по списку вверх и вниз. Кроме того, списки могут предусматривать и горизонтальную прокрутку, что позволяет программистам втискивать очень длинные строки текста в списки с минимальными усилиями. Однако пользователям эта возможность не приносит ничего, кроме головной боли.

Горизонтальная прокрутка текста – это нечто ужасное, и она не должна применяться *нигде и никогда* – за исключением больших таблиц (например, электронных), где фиксированные заголовки колонок и строк способны порождать локальный контекст. При горизонтальной прокрутке текста из виду скрываются один или более первых символов каждой строки отображаемого текста. Это делает нечитаемыми *все строки*, что приводит к потере целостности восприятия текста.

Ни в коем случае не используйте горизонтальную прокрутку текста.

Если вы столкнетесь с ситуацией, которая, на ваш взгляд, просто-таки требует горизонтальной прокрутки, попробуйте поискать альтернативные решения. Прежде всего попытайтесь ответить, почему текст в списке такой длинный. Можно ли сократить его? Можно ли перенести текст на следующую строку, чтобы сократить его длину? Можно ли позволить пользователю заменить длинные строки более короткими вариантами? Можно ли вместо текста использовать небольшие графические изображения? Можно ли использовать всплывающие подсказки? Рассмотрите также возможность увеличения ширины списка. Может быть, имеется способ как-то иначе перераспределить элементы управления в окне, с тем чтобы увеличить горизонтальный размер списка?

Если нет возможности увеличить ширину элемента управления, наилучший выход – перенос текста на следующую строку. Используйте отступ, чтобы перенесенная строка отличалась от других. Это означает, что теперь вам придется иметь дело со списком, в котором элементы имеют переменную высоту, но это все равно лучше, чем горизонтальная прокрутка.

Помните: мы говорим только о *тексте*. Для графических изображений и больших таблиц нет ничего плохого в горизонтальных полосах прокрутки, как нет ничего плохого в самих по себе окнах с горизонтальной прокруткой. Но предложить пользователю горизонтальную прокрутку в текстовом списке – практически то же самое, что предложить компьютер с педальным приводом... чрезвычайно удобно, не так ли?

Прямой ввод данных в список

Исторически сложилось так, что программисты не особенно утруждали себя попытками обеспечить пользователям возможность ввода текста напрямую внутри списков. Конечно, потребность вводить текст прямо по месту нахождения текстового элемента возникает достаточно часто, и неуклюжие диалоговые окна во многом являются следствием желания программистов увернуться от написания кода для редактирования по месту.

Однако современные элементы управления, предназначенные для просмотра списков и деревьев в операционной системе Windows и на

других платформах, предлагают средства редактирования по месту. Оба этих элемента управления используются в Проводнике (Windows Explorer); можно увидеть, как они работают, если попытаться переименовать файл или каталог. Чтобы переименовать файл в Mac OS или в Windows 95, вы дважды щелкаете мышью на имени объекта (но не слишком быстро, чтобы два щелчка не были восприняты как один двойной щелчок, по которому открывается рассматриваемый объект). После этого вы вносите необходимые изменения. (Это поведение претерпело некоторые изменения в Windows XP, так что в некоторых представлениях, чтобы войти в режим переименования, приходится щелкать правой кнопкой мыши и выбирать из контекстного меню пункт Переименовать; можно ли это назвать прогрессом?) Объекты, которые можно отредактировать вне списков, должны быть доступными для редактирования и внутри списков.

Граничный случай, превращающий редактирование по месту в настоящую проблему, – добавление в список нового элемента. Большинство проектировщиков для добавления новых элементов в список задействует другие идиомы – кнопку или соответствующий пункт меню, – добавляющие в список новый, пустой элемент, после чего пользователь может редактировать его название по месту. Наверное, более разумным было бы предусмотреть возможность создания пустого элемента двойным щелчком мыши по пустому пространству между существующими элементами или, по крайней мере, иметь свободное пространство в начале или в конце списка с надписью «Добавить новый элемент», чтобы пользователь легко мог обнаружить эту возможность. Другое решение этой проблемы связано с применением комбо-списка, о котором мы расскажем ниже.

Комбо-списки

В Windows 3.0 появился новый элемент управления – комбинированный список, или **комбо-список** (combobox). Как следует из названия, этот элемент представляет собой сочетание списка и поля редактирования (рис. 21.14). Он обеспечивает недвусмысленный метод ввода данных в список. Вариант с раскрывающимся списком, к тому же, значительно экономит экранное пространство.

В комбо-списках четко разграничиваются собственно список и поле ввода, что сводит к минимуму непонимание со стороны пользователя. Комбо-список превосходно подходит для тех случаев, когда необходимо организовать выбор единственного объекта. Поле редактирования может использоваться для ввода новых элементов списка, и, кроме того, в нем отображается текущий, выбранный из списка объект. Когда текущий объект отображается в поле редактирования, пользователь может изменять его. Это своего рода урезанный вариант редактирования по месту.

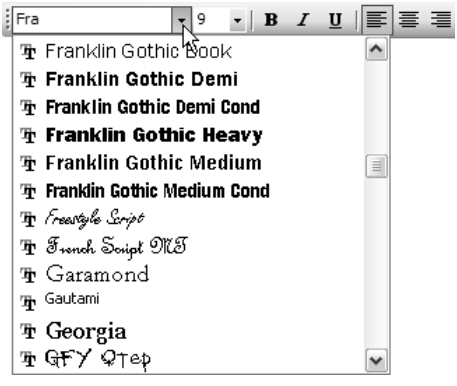


Рис. 21.14. *Раскрывающееся поле со списком в редакторе Word позволяет пользователям выбирать шрифты из списка или просто набирать название нужного шрифта в текстовом поле*

Поскольку поле редактирования комбо-списка отображает текущий выбор, этот элемент неизбежно предназначен для выбора единственного значения. Не существует полей со списком, предлагающих возможность множественного выбора. Единственность выбора подразумевает взаимное исключение, и это одна из причин того, почему поле со списком легко может использоваться для замены групп радиокнопок. (В операционной системе Mac OS заменой крупных групп радиокнопок служили всплывающие меню, которые появились раньше, чем комбо-списки в Windows, однако у этого варианта не было возможности редактирования данных, которая есть у комбо-списков.) Другой причиной является высокая степень экономии пространства и возможность динамического добавления элементов – то, что радиокнопки делать не позволяют.

Когда комбо-список реализован как раскрывающийся список, он показывает текущий выбор, не занимая места для отображения всех вариантов выбора. По существу, собственно список доступен по первому требованию и этим напоминает меню, отображающее список команд по желанию пользователя. На практике обычно используются именно раскрывающиеся комбо-списки.

Бережливое отношение комбо-списка к экранному пространству позволяет такому сложному элементу вытворять удивительные вещи, например находиться внутри главного окна программы. Он может достаточно удобно расположиться даже на панели инструментов. Это очень эффективный элемент управления для всех монопольных приложений. Применение полей со списками на панелях инструментов более эффективно, нежели размещение эквивалентных функций в меню, поскольку пользователь может видеть состояние комбо-списка непосредственно, не совершая дополнительных действий, – нет необходимости открывать меню.

Раз перетаскивание реализовано для списков, оно должно быть реализовано и для комбо-списков. Возможность раскрыть комбо-список, прокрутить его до нужного элемента, а затем перетащить этот элемент внутрь документа порождает очень мощную идиому. Возможность перетаскивания следует делать естественной частью комбо-списков.

Элементы управления деревьями

В Mac OS 7 и Windows 95 появились универсальные элементы управления деревьями, которые уже какое-то время использовались в мире UNIX. Элементы управления деревьями – это варианты списков, способные представлять иерархические данные. Они отображают дерево с последовательно развернутыми ветвями, где каждая ветвь снабжена пиктограммой. Ветви можно разворачивать и сворачивать – как элементы в системах редактирования структурированных текстов. Программистам нравится такое представление данных. Оно часто используется для навигации по файловой системе и является очень эффективным инструментом представления иерархически структурированной информации.

К сожалению, иерархические деревья – это инструмент, которым злоупотребляют чаще всего. Они могут создавать значительные сложности для пользователей; многим людям тяжело думать в терминах иерархических структур данных. Мы видели бесчисленное множество интерфейсов, в которых программисты запихивали в древовидную структуру неиерархические данные, мотивируя это тем, что деревья «интуитивно понятны». Деревья, безусловно, интуитивно понятны программистам (а прочие люди, безусловно, способны привыкнуть к деревьям), однако их огромный минус в том, что они не дают пользователям задействовать другие, более интересные связи между объектами, не представленные жесткой иерархией.

Единственная ситуация, в которой применение древовидной структуры оправдано (неважно, насколько велик соблазн), – это ситуация, когда данные изначально фигурируют в мыслительных процессах как иерархии (например, генеалогическое древо). А представление через древовидную структуру произвольных объектов, организованных в произвольном порядке, вызванное к жизни прихотью программиста, ставит под большое сомнение такие факторы, как удобство и простота использования.

Элементы ввода

Элементы ввода дают пользователю возможность не только выбирать существующие сведения, но и вводить новую информацию.

Самый простой элемент ввода – поле редактирования текста (поле ввода). Как и элементы выбора, элементы ввода представляют собой имена существительные пользовательского интерфейса программы. По-

скольку комбо-списки содержат в себе поле ввода, доступное для редактирования, некоторые разновидности комбо-списков могут считаться также и элементами управления вводом. Кроме того, любой элемент управления, который позволяет пользователю вводить числовые значения, тоже является элементом ввода. В эту категорию попадают такие элементы управления, как счетчики (spinners), линейки (gauges), ползунки (sliders) и рукоятки (knobs).

Ограничивающие и неограничивающие элементы ввода

Любой элемент управления, ограничивающий набор значений, доступных для ввода пользователем, является **ограничивающим элементом ввода**. Так, например, ползунок со шкалой значений от 0 до 100 является ограничивающим элементом ввода. Независимо от действий пользователя не может быть введено никакое число, выходящее за диапазон определенных программой значений. Тем самым пользователь просто не в состоянии ввести недопустимое значение, когда для ввода используется элемент управления с ограничениями.

И наоборот, простое текстовое поле может принять любые алфавитно-цифровые данные, введенные пользователем. Эта идиома открытого ввода – пример **неограничивающего элемента ввода**. Неограничивающие элементы ввода позволяют пользователю без труда вводить недопустимые значения. Разумеется, впоследствии программа может отклонить введенные данные, но за пользователем по-прежнему сохраняется возможность ввода таких данных.

Проще говоря, ограничивающие элементы ввода должны использоваться везде, где необходимо ограничить множество допустимых значений. Если программа требует ввести число между 7 и 35 и при этом дает возможность вводить числа в диапазоне от –1 000 000 до +1 000 000, она не сделает пользователя счастливее.

Люди предпочтут элемент управления, который имеет нижнюю допустимую границу, равную 7, и верхнюю границу, равную 35 (полезно указать на эти ограничения понятным пользователю способом). Пользователи умны, они сразу же оценят такое ограничение и будут за него благодарны.

Важно понимать, что мы говорим о качественном свойстве элемента ввода, а не о свойстве самих данных. Ограничивающий элемент ввода должен четко информировать пользователя (преимущественно визуально) о допустимых границах. Текстовое поле, которое отвергает ввод пользователя *после того, как* он выполнил ввод, *не может* считаться ограничивающим элементом управления. Это просто *невежливый* элемент управления.

Применяйте ограничивающие элементы управления для получения ограниченных данных.

Данные, которые нужны программам, обычно имеют допустимые пределы – и все же многие программы позволяют вводить числовые данные с помощью неограничивающих элементов ввода. Когда пользователь непреднамеренно вводит значение, которое программа не может принять, она выводит окно с сообщением об ошибке. Тем самым она вызывает у пользователя раздражение, предлагая якобы существующие возможности. «Что бы вы хотели на десерт? У нас есть все!» – говорит программа. «Мороженое», – отвечаете вы. «Жаль, но у нас его нет», – говорит она. «Как насчет пирога?» – невинно спрашиваете вы. «Не-а», – отвечает программа. «Пирожные?» – «Не-а!» – «Леденцы?» – «Не-а!» – «Шоколад?» – «Не-а!» – «Что тогда у вас есть?!» – в гневе и расстройстве кричите вы. «Не сердитесь! – отвечает программа возмущенно. – У нас широкий ассортимент фруктовых компотов». Это наглядная демонстрация того, что чувствует пользователь, когда мы даем ему неограничивающий элемент ввода для ввода данных с фиксированным диапазоном возможных значений. Пользователь вводит **«17»**, а мы вознаграждаем его за это сообщением об ошибке, которое гласит: «Вы можете ввести значения из диапазона от 4 до 8». Это пример ужасно спроектированного пользовательского интерфейса. Намного лучше сделать так, чтобы элемент ввода автоматически ограничивал ввод числами 4, 5, 6, 7 и 8. Если множество ограниченных значений состоит из фрагментов текста, а не из чисел, по-прежнему можно использовать какой-то ползунок, или комбо-список, или простой список. На рис. 21.15 представлен пример ограничивающего ползунка, используемого Microsoft в диалоговом окне настройки параметров экрана Windows. Этот элемент работает как ползунок или полоса прокрутки, но имеет четыре дискретных позиции, которые представляют различные разрешения дисплея. А ведь здесь Microsoft с легкостью могла бы использовать нередактируемый комбо-список. Во многих случаях ползунок – хороший выбор, потому что он недвусмысленно сообщает о диапазоне допустимых для ввода значений. Поле со списком по размерам немногим меньше, но оно не раскрывает свои карты,

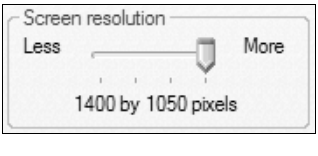


Рис. 21.15. *Ограничивающие элементы ввода дают пользователю возможность вводить только допустимые значения. Они не позволят ввести недопустимое значение лишь для того, чтобы потом отклонить его. На этом рисунке показан ограничивающий элемент ввода – ползунок из диалогового окна настройки параметров экрана в Windows XP. Этот небольшой ползунок имеет четыре дискретных положения. При перемещении ползунка слева направо описание под ним изменяется с «800 на 600 точек» на «1024 на 768 точек», далее на «1280 на 1024 точек» и на «1400 на 1050 точек»*

пока пользователь не щелкнет по нему, – такой вариант поведения менее дружелюбен.

Если пользователь должен выразить выбор числовым значением в определенных границах, дайте ему элемент управления, сообщающий об этих границах и предотвращающий ввод недопустимых значений. Такую возможность дают ползунки. Хотя у ползунков имеются существенные недостатки, в одной области они являют собой пример для подражания: ввод количественной информации аналоговым способом. Ползунки позволяют пользователю определять числовые значения в относительных терминах, а не в результате непосредственного ввода с клавиатуры. Иначе говоря, пользователь перемещает бегунок, определяя его относительную позицию и тем самым – пропорциональное значение для использования внутри программы. Ползунки менее удобны для ввода точных значений, хотя во множестве программ они используются именно с этой целью. Для ввода точных значений лучше подходят счетчики (spinners).

Счетчики

Счетчики – это общепринятая форма элементов ввода, которые позволяют вводить числовые данные как с клавиатуры, так и с помощью мыши. Счетчик состоит из небольшого поля ввода и двух прикрепленных к нему кнопок половинной высоты (рис. 21.16). Благодаря счетчикам грань между ограничивающими и неограничивающими элементами ввода данных становится размытой.

Маленькие кнопки со стрелками позволяют пользователю изменять значение в поле редактирования небольшими шагами. Эти шаги могут выполняться до определенного предела: значение не может превысить максимум, установленный программой, или стать меньше установленного минимума. Если пользователь пожелает внести значительное изменение за одно действие или ввести определенное число, он может сделать это за счет прямого ввода числа в поле редактирования – точно так же, как в любое другое поле ввода. К сожалению, поле редактирования данного элемента управления не накладывает ограничений, предоставляя пользователю возможность вводить любые значения, в том числе далеко выходящие за рамки ограничений. В диалоговом окне «Параметры страницы», показанном на рисунке, при вводе пользователем недопустимого значения программа будет вести себя так же, как и большинство других грубых программ: вывесит сообщение об ошибке, которое в редких случаях сопровождается индикацией допустимых значений, и потребует, чтобы пользователь щелкнул по кнопке OK для продолжения работы.

В общем и целом счетчики – прекрасная идиома, которая вполне может использоваться вместо простых полей ввода в качестве ограничивающего элемента ввода. В главе 25 мы обсудим способы улучшения обработки ошибок в рамках элементов управления.

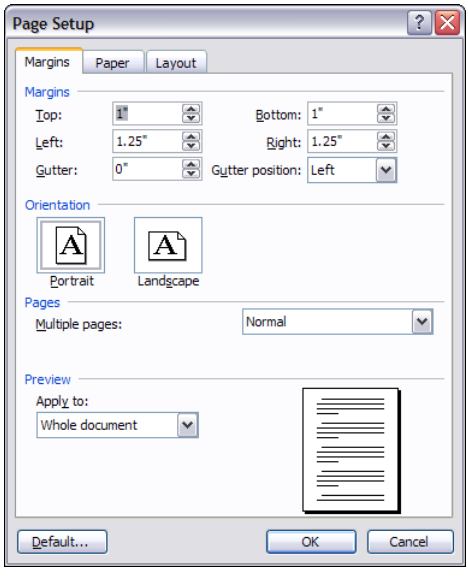


Рис. 21.16. *Счетчики широко используются в диалоговом окне «Параметры страницы» в Microsoft Word. В верхней части окна вы можете наблюдать пять счетчиков. Щелчком по одной из маленьких кнопок со стрелками пользователь может увеличивать или уменьшать значение поля небольшими шагами. Если пользователь желает кардинально изменить значение счетчика за одно действие или ввести точное значение, он может отредактировать значение прямо в поле ввода. Кнопки со стрелками реализуют ограничение элемента ввода, тогда как поле редактирования этого не делает. Является ли счетчик при этом ограничивающим элементом управления?*

Рукоятки и ползунки

Рукоятки и **ползунки** – это идиомы, основанные на метафорах механического века – вращающихся ручках и линейно движущихся рычагах. Рукоятки очень эффективно расходуют экранное пространство, и оба этих элемента управления замечательно справляются с задачей обеспечения визуальной обратной связи по настройкам (рис. 21.17).

Если рукоятки реализованы некорректно, то ими может быть очень трудно манипулировать. Иногда программисты ошибочно заставляют пользователей водить курсором по дуге, что весьма непросто. Правильная реализация рукояток должна позволять линейный ввод в двух измерениях: щелчок по рукоятке и перемещение курсора вверх или вправо должны увеличивать значение, а щелчок с перемещением влево или вниз – уменьшать его. Разумеется, этой идиоме пользователи должны будут научиться (иначе у них может возникнуть желание водить курсор по дуге), так что рукоятки лучше всего применять в специализированных приложениях, где у пользователей есть время на ос-

Рис. 21.17. *Модульный программный синтезатор Reaktor (компании Native Instruments) активно использует рукоятки и ползунки. Это удобные элементы интерфейса – и не просто потому, что музыканты и продюсеры знакомы с ними по аппаратным устройствам. Гораздо важнее то, что они дают пользователям наглядную и простую для понимания обратную связь, которая гораздо доступнее демонстрирует текущие настройки, нежели длинные списки чисел, являющиеся для композитора удручающим зрелищем*

воение идиом. Ползунки часто являются предпочтительным выбором, поскольку визуально подчеркивают тот факт, что движение происходит лишь вдоль одной оси. Благодаря своему компактному размеру и визуальным качествам (не говоря уже об истории), ползунки популярны в приложениях для работы со звуком.

Ползунки и рукоятки применяются в основном в качестве ограничивающих элементов управления ввода, однако иногда используются (не факт, что корректно) в качестве элементов, управляющих отображением данных. В большинстве ситуаций для *перемещения* информации внутри экрана полосы прокрутки предпочтительнее, поскольку они способны наглядно показывать объемы прокручиваемой области; у ползунков же с этим имеются некоторые проблемы. Однако ползунки – превосходное средство для действий, связанных с масштабированием, например для изменения масштаба карты или размера миниатюры фотографии.

Круговые манипуляторы

Круговой манипулятор – это разновидность рукоятки, но гораздо более простая в использовании. Экранные круговые манипуляторы похожи на колесо прокрутки мыши и ведут себя во многом так же. Они популярны в некоторых 3D-приложениях, поскольку компактны и не ограничивают ввод, что идеально подходит для определенных видов

панорамирования и масштабирования. В отличие от полосы прокрутки, круговой манипулятор не должен информировать пользователя о пропорциях, поскольку диапазон работы этого элемента управления – бесконечность. Имеет смысл связывать такие элементы управления с неограниченным движением в определенном направлении (например, с масштабированием) или с перемещением по закольцованным данным.

Прочие ограничивающие элементы управления вводом

Отказываясь от наследия традиционных элементов управления графического пользовательского интерфейса и багажа механических аналогов, новое поколение экспериментальных пользовательских интерфейсов порождает больше визуальных и жестовых идиом – от простых двухмерных панелей, где щелчок в любой точке определяет значения сразу для двух механизмов ввода (вертикальная и горизонтальная координаты определяют значения двух различных параметров), до более сложных интерфейсов непосредственного взаимодействия (см. примеры на рис. 21.18). Такие элементы управления обычно являются ограничивающими, поскольку их реализация требует внимательного изучения отношения между жестом мышью и функцией. Они часто содержат механизмы визуальной обратной связи. Кроме того, эти элементы управления лучше всего подходят для ситуаций, когда пользователю требуется управлять целым набором переменных и он готов потратить время и силы на изучение сложной идиомы.

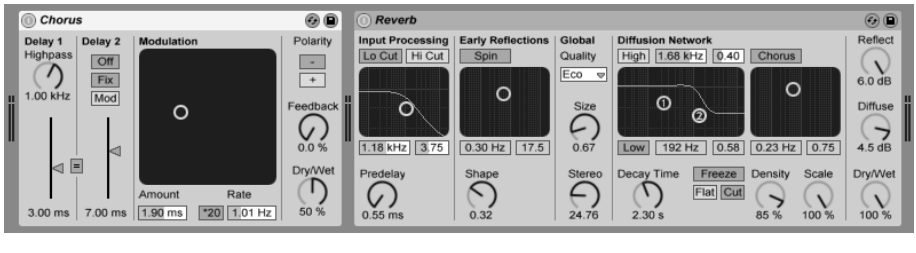


Рис. 21.18. *Ableton Live – компьютерный инструмент для создания и исполнения музыки, задействующий целый ряд двухмерных элементов управления вводом. Они предоставляют качественную визуальную обратную связь, позволяют пользователям регулировать сразу несколько параметров и поддерживают выразительные жестовые взаимодействия. Их ограничивающая сущность сообщает пользователю о текущих настройках в контексте границ диапазонов и исключает возможность, что пользователь сделает недопустимый выбор (композитору вряд ли понравится, если диалоговое окно вдруг прервет его работу)*

Неограничивающий ввод: поля ввода текста

Первый из неограничивающих элементов ввода – **поле ввода текста.** Этот простейший элемент управления позволяет пользователям наби-

рать любые алфавитно-цифровые строки. Как правило, поля ввода – это небольшие области, внутри которых можно набрать одно-два слова, но они могут быть реализованы и в виде довольно сложных текстовых редакторов. Пользователь может редактировать текст внутри полей, используя стандартные средства непрерывного выделения (см. главу 19), как с помощью мыши, так и с помощью клавиатуры.

Поля ввода текста часто применяются для ввода значений в приложениях, основанных на базах данных (включая веб-сайты, в основе которых лежит база данных), и имеют вид либо поля ввода текста в диалоговых окнах, либо комбо-списка. Независимо от роли поля редактирования текста часто используются для организации ввода значений с ограничениями. Однако если множество допустимых значений ограничено, для них не следует использовать поля ввода текста. Если приемлемые значения являются числовыми, используйте ограничивающие числовые элементы ввода, такие как ползунки. Если в качестве приемлемых значений используется список текстовых строк, используйте списочные элементы управления, чтобы пользователю не пришлось набирать текст вручную.

Иногда набор приемлемых значений является конечным, но слишком большим, чтобы для его представления можно было использовать список. Например, программа может требовать ввода строки, состоящей из 30 алфавитных символов, исключая пробелы, символы табуляции и знаки препинания. В этом случае применение элемента текстового ввода, хотя и ограничивающего, вероятно, неизбежно. Но если это единственное ограничение, элемент ввода текста можно спроектировать так, чтобы отклонять ввод неалфавитных символов и не позволять вводить строки длиннее 30 символов. Однако это порождает вопросы, связанные с проверкой допустимости введенных данных.

Проверка допустимости

Когда пользователю предложено неограничивающее текстовое поле ввода, которое при этом принимает лишь строки определенного формата, вероятно, имеется необходимость помочь пользователям конструировать «допустимые» строки. Как правило, приложение оценивает ввод пользователя, когда он закончил набор, и выбрасывает сообщение об ошибке, если ввод неправильный. Очевидно, это может раздражать пользователей и, в конечном счете, снижать их производительность.

Как мы уже упоминали, лучшее решение этой проблемы – использовать ограничивающие элементы ввода и сделать ввод недопустимых строк невозможным. (Распространенный пример: раскрывающийся список месяцев, освобождающий пользователя от необходимости без ошибок набирать слово «февраль».)

В других случаях это не всегда практично (ввод регистрационного ключа на экране регистрации приложения). Программисты справляются с дилеммой, создавая **элементы ввода с проверкой данных**, которые

являются разновидностью неограничивающих полей текстового ввода со встроенной проверкой допустимости ввода и обратной связью. Имеется множество стандартных форматов вводимых данных – даты, телефонные номера, почтовые индексы, номера социального страхования. На рынке есть коммерческие специализированные элементы ввода текста; вы можете приобрести варианты полей текстового ввода, которые, к примеру, позволяют осуществлять ввод только чисел или телефонных номеров либо не дают вводить символы пробела и табуляции.

Хотя проверка допустимости при вводе – весьма распространенная идиома, большинство таких элементов управления оставляют желать лучшего. Ключ к успешному проектированию элемента ввода с проверкой данных – в хорошо развитой обратной связи с пользователем. Элемент ввода данных, просто отвергающий введенные данные, тем самым грубит пользователю и гарантированно злит и обижает его.

Одно из фундаментальных улучшений основано на следующем принципе проектирования: *элементы, различающиеся поведением, должны различаться и визуально* (см. главу 14). Следует визуально отделить поля ввода, осуществляющие проверку допустимости данных, от обычных элементов ввода, используя шрифт, цвет границы или фон поля.

Однако главное направление совершенствования – обеспечение обогащенной обратной связи с пользователем. К сожалению, существующие сегодня элементы текстового ввода практически не имеют встроенных средств для организации какого-либо рода обратной связи. Проектировщики должны подробно описывать такие механизмы, а программистам, вероятно, придется реализовывать их в специально разработанных элементах ввода.

Активная и пассивная проверка допустимости

Некоторые элементы ввода отклоняют символы по мере их ввода с клавиатуры. Элемент управления, активно отклоняющий символы в процессе нажатия клавиш, – это пример **активной проверки допустимости**. Так, например, элемент ввода текстовой информации может принимать только алфавитные символы и отвергать цифры. Некоторые элементы ввода отвергают ввод любых символов, кроме цифровых. Другие в реальном времени отвергают ввод пробелов, символов табуляции, дефисов и прочих знаков препинания. Некоторые разновидности могут быть довольно интеллектуальными и отклонять ввод определенных цифр, основываясь на оперативных вычислениях, например на вычислении контрольной суммы.

Когда активная проверка допустимости отклоняет нажатие клавиши, следует сообщить пользователю о том, что произошло и почему. Если такое объяснение будет предоставлено, пользователь не будет склонен считать, что ввод отвергнут беспричинно (или из-за неисправности клавиатуры). Кроме того, подобная обратная связь сообщает пользователю, что именно требуется программе.

Иногда диапазон допустимых данных таков, что программа не в состоянии определить их правильность в процессе набора, – требуется, чтобы пользователь закончил ввод. В этом случае процесс проверки начинается только после того, как элемент ввода потерял фокус, то есть когда пользователь закончил работу с этим полем ввода и переходит к другому полю. Процесс проверки должен выполняться также при закрытии диалогового окна, а если элемент ввода находится не в диалоговом окне – тогда, когда пользователь вызывает другую функцию (например, когда он нажимает на кнопку Отправить на веб-странице). Если элемент управления ожидает, когда пользователь закончит вводить данные, и только потом приступает к проверке правильности – это **пассивная проверка допустимости**.

Элемент ввода может, например, ожидать, пока адрес не будет введен полностью, после чего проверить существование такого адреса в базе данных. Каждый отдельный символ является вполне допустимым, но весь адрес целиком может и не пройти проверку. Программа могла бы попытаться проверять адрес по мере ввода каждого отдельного символа, но это, скорее всего, привело бы к перегрузке сети и сервера. Кроме того, хотя программа в принципе может знать в любой конкретный момент, допустим ли набранный адрес, пользователь может покинуть поле ввода до того, как полностью наберет его.

Чтобы организовать проверку правильности ввода, можно попробовать предусмотреть счетчик времени, который сбрасывается нажатием на очередную клавишу. Если счетчик достиг нуля, проверьте допустимость значения. Устанавливайте счетчик на время, приблизительно равное половине секунды. Пока пользователь нажимает клавиши чаще, чем раз в полсекунды, система чрезвычайно отзывчива. Если пользователь приостанавливается больше чем на 500 миллисекунд, программа разумно полагает, что пользователь сделал паузу для того, чтобы подумать (такой срок может означать «месяцы» жизни с точки зрения центрального процессора), и производит промежуточный анализ введенной информации.

В целях обеспечения обогащенной визуальной обратной связи поле ввода может изменять свой цвет, отражая оценку корректности введенных данных. Поле ввода может отображаться в розовых оттенках, пока программа не убедилась в допустимости данных, после чего цвет поля ввода может измениться на белый или зеленый.

Оперативные подсказки

Еще одно неплохое решение проблемы проверки допустимости – **оперативные подсказки**. Это небольшие всплывающие окна, которые выглядят и ведут себя точно так же, как обычные всплывающие подсказки (хотя цветовое решение может быть иным, чтобы не возникало путаницы). Их назначение – указать диапазон допустимых данных для элемента ввода, поддерживающего проверку допустимости, – как активную, так и пассивную. Обычная всплывающая подсказка появляется,

когда курсор мыши на мгновение задерживается над элементом управления, тогда как оперативная подсказка с описанием допустимого диапазона данных должна появляться в тот момент, когда элемент управления обнаружит запрещенный символ (впрочем, такого рода подсказку можно выводить точно так же, как и обычную всплывающую подсказку, если курсор мыши задерживается над элементом ввода на срок более одной секунды или около того). Если, например, пользователь вводит нечисловой символ в поле ввода числа, программа должна вывести панель с подсказкой около поля ввода, при этом не загораживая его. Такая подсказка, например, может содержать текст: «0 – 9» – краткий, лаконичный, но очень эффективный. Да, ввод пользователя отвергнут – но он не игнорируется. Оперативные подсказки можно использовать и для элементов ввода с пассивной проверкой допустимости, как показано на рис. 21.19.

Рис. 21.19. *Идиома всплывающей подсказки настолько эффективна, что она с легкостью может применяться в самых различных ситуациях. Вместо желтых всплывающих подсказок, которые заменяют текстовые надписи на кнопках-значках, мы можем предусмотреть розовые подсказки, описывающие поведение неограничивающих элементов ввода. Такого рода оперативные подсказки способны также помочь в искоренении окон с сообщениями об ошибках. В данном примере, если пользователь вводит значение меньше допустимого, программа может заменить введенное значение минимально допустимым и отобразить подсказку, которая немодально объясняет причину такой замены. Теперь пользователь может сразу набрать новое значение либо принять предложенное, не прерываясь на диалоговое окно с сообщением об ошибке*

Обработка недопустимых данных

Обычно поле ввода используется для получения от пользователя числового значения, необходимого программе, например размера шрифта. Пользователь может ввести любое число, по своему желанию, от 5 до 500 – и поле примет его и передаст введенное значение программе. Если пользователь вводит некорректное значение, элемент управления должен принять какое-либо решение. Например, в Microsoft Word, если вы введете в поле размера шрифта строку **«asdf»**, программа выдаст окно с сообщением об ошибке: «Указано неверное число». После этого она вернет содержимое элемента ввода в его первоначальное состояние. Диалоговое окно с сообщением об ошибке – обычная глупость, а вот отклонение моего бессмысленного ввода совершенно оправданно. Но что произойдет, если вы введете в числовое поле стро-

ку **«девять»**? Программа отклонит этот ввод с тем же самым коротким сообщением об ошибке. Если бы вместо этого элемент ввода создавался как поле для ввода чисел, то, возможно, его поведение улучшилось бы. Меня мало беспокоит, как программа преобразует слово **«девять»** в цифру **9**, но, конечно же, неправильно, когда программа сообщает, что **«девять»** – это «неверное число». Без сомнения, это правильное число, а программа просто «села в лужу».

Единицы измерения и размеры

Хорошо, когда элементы ввода текстовой информации достаточно интеллектуальны, чтобы распознать соответствующие единицы измерения. Например, если программа запрашивает размер чего-либо и пользователь в ответ вводит **«5с»**, или **«5см»**, или **«5 сантиметров»**, элемент ввода должен не только передать программе число «пять», но и сообщить о том, что это именно пять *сантиметров*. Если пользователь вводит **«5мм»**, элемент ввода должен сообщить о том, что введено именно пять *миллиметров*. Программа SketchUp – изящное приложение для эскизирования – поддерживает такого рода обратную связь. Точно так же, если речь идет о качественно спроектированном приложении для финансового анализа, оно должно понимать, что «5мм» означает пять миллионов.

Предположим, что поле ввода запрашивает ширину столбца. Пользователь может ввести либо число, либо число и единицу измерения, как описано выше. Можно было бы также позволить пользователю ввести слово «по умолчанию» («default») как предписание программе установить ширину столбца в значение по умолчанию. Пользователю можно было бы позволить ввести и слово «подогнать» (best fit) как предписание программе самостоятельно измерить ширины всех элементов в столбце и выбрать оптимальную ширину столбца. В таком подходе есть одна неувязка: слова *по умолчанию* и *подогнать* должны уже быть в голове пользователя, а не где-нибудь в программе. Впрочем, эта проблема легко решается. Достаточно предоставить соответствующий вариант выбора в комбо-списке. Пользователь может раскрыть список и обнаружить несколько стандартных размеров, среди которых будут находиться слова *по умолчанию* и *подогнать*. Microsoft использовала эту идею в Word (рис. 21.20).

Раскрыв список, пользователь увидит такие элементы, как «по ширине страницы» или «страница целиком», и может выбрать наиболее подходящий. Благодаря этой идиоме информация переместилась из головы пользователя в программу, где она видима и доступна для выбора.

Режимы ввода: вставка и замена

В большинстве редакторов текста существует возможность переключения между режимом вставки, когда текст, расположенный правее курсора, сохраняется, сдвигаясь в сторону по мере ввода нового текста, и режимом замены, когда текст, расположенный правее курсора,

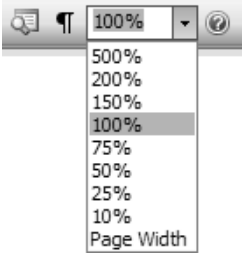


Рис. 21.20. *Это раскрывающееся поле со списком – превосходный инструмент для реализации ограничивающих полей ввода, потому что может содержать не только числовые значения. Пользователь не должен запоминать такие словосочетания, как «по ширине страницы» или «страница целиком», поскольку их можно выбрать из раскрывающегося списка. Программа самостоятельно переводит данные словосочетания в необходимые числа – и оба участника диалога остаются довольны*

замещается вводимым текстом. Эти два режима повсеместно используются в мире текстовых процессоров и, подобно ФОРТРАНу, похоже, отмирать не собираются. Режимы вставки и замены – это режимы, которые вызывают существенное изменение в поведении интерфейса без видимой индикации режима (пока пользователь не начнет набирать текст), а очевидный способ переключения между этими режимами отсутствует (по крайней мере, в Windows) – есть лишь малоочевидное сочетание клавиш.

Сегодня уже трудно себе представить человека, который при наличии современных текстовых процессоров с графическим интерфейсом использовал бы режим замены, – но такие люди, несомненно, есть. В то же время давать возможность ввода с заменой в простых текстовых элементах ввода было бы глупо – неприятности в этом случае могут быстро перевесить преимущества. Разумеется, дело обстоит совершенно иначе, если вы проектируете текстовой процессор.

Элементы управления текстовым вводом как средство вывода – плохая идея

Текстовое поле с его знакомым системным шрифтом и наглядным белым полем однозначно ассоциируется с вводом данных. И все же разработчики программного обеспечения часто используют элементы ввода в режиме отображения информации, недоступной для редактирования. Разумеется, элементы ввода могут исполнять роль элементов вывода, но использовать их только для вывода означает обмануть ожидания пользователя, что едва ли его обрадует. Если требуется выводить текстовые данные, используйте для этого элементы вывода текстовой информации, а не *элементы ввода*. Например, если вы желаете показать объем свободного дискового пространства, не используйте текстовый элемент ввода, потому что пользователь, скорее все-

го, подумает, что он сможет получить больше свободного пространства, введя большее число. По крайней мере, именно об этом говорит ему элемент ввода на своем «языке тела».

Если вы собираетесь выводить редактируемую информацию, тогда можете смело использовать элементы редактирования, потому что в этой ситуации они работают именно так, как обещает их внешний вид. В противном случае старайтесь использовать элементы управления отображением, которые будут описаны в следующем разделе.

Чтобы отобразить информацию, недоступную для редактирования, используйте элементы, предназначенные только для вывода.

Элементы управления отображением

Элементы управления отображением используются для отображения и управления *визуальным представлением информации* на экране. Типичными примерами элементов отображения являются разделители (splitters) и полосы прокрутки (scrollbars). Элементы, которые управляют способом визуального отображения объектов на экране, так же относятся к этой категории, как и те, что отображают статическую информацию, доступную только для чтения. Сюда входят разделители страниц, линейки, направляющие, сетки, рамки, а также трехмерные вдавленные и выпуклые линии. Мы не будем обсуждать все эти элементы подробно, а сосредоточимся лишь на тех из них, которые вызывают наибольшие затруднения.

Текстовые элементы

Вероятно, самый простой элемент управления отображением – **элемент вывода текстовой информации**, который отображает текстовое сообщение в некоторой позиции на экране. Задача, которая возложена на этот элемент управления, довольно прозаическая – предоставить текстовые метки для других элементов управления и вывести данные, которые не могут или не должны быть изменены пользователем.

Единственная серьезная проблема этого элемента состоит в том, что он зачастую используется там, где должны присутствовать элементы ввода (и наоборот). В массе своей информация, сохраняемая в компьютере, может быть изменена пользователем. Так почему бы не позволять пользователю изменять ее в том же самом месте, где программа ее отображает? Почему механизм ввода значения должен отличаться от механизма вывода? Во многих случаях не имеет смысла разделять эти две взаимосвязанные функции. В большинстве ситуаций там, где программа отображает значение, которое может быть изменено, она обязана сделать его доступным для редактирования – с помощью элемента ввода. Благодаря этому пользователь сможет щелкнуть по значению

и изменить его. Специальные режимы редактирования практически всегда оказываются интерфейсным налогом.

В течение многих лет Adobe Photoshop настаивала на открытии диалогового окна при создании форматированного текста. Пользователь не мог точно увидеть, как будет выглядеть текст, наложенный на изображение, что вынуждало его повторять процедуру ввода снова и снова, методом проб и ошибок добиваясь приемлемых результатов. В конце концов Adobe устранила эту проблему, позволив пользователю вводить форматированный текст непосредственно в изображение, в полном соответствии с принципом WYSIWYG (what you see is what you get – «что видите, то и получаете»), как это и должно быть.

Полосы прокрутки

Полосы прокрутки в современном графическом пользовательском интерфейсе служат крайне важной цели – они позволяют осмысленным образом помещать большие объемы информации внутри крохотных рамок окон и панелей. К сожалению, они также расходуют экранное пространство, ими сложно манипулировать, и они обычно вызывают недовольство пользователей. Несомненно, полосы прокрутки часто используются сверх меры и, кроме того, плохо изучены. Применение же полос прокрутки как средства навигации по содержимому окна и документа – то есть как элемента управления отображением – полностью оправданно.

Замечательное преимущество полосы прокрутки, кроме ее почти универсальной доступности, состоит в создании контекста текущего положения в окне. **Бегунок** полосы прокрутки – это маленький перемещаемый прямоугольник, который указывает текущее положение и нередко масштаб «территории», доступной для прокрутки.

Многие полосы прокрутки скупы на информацию. Лучшие полосы прокрутки задействуют бегунки, размер которых изменяется пропорционально размеру просматриваемого документа, а кроме того сообщают:

- сколько всего страниц содержится в документе;
- номер текущей страницы (записи, картинки) по мере перемещения бегунка;
- первое предложение (или элемент) на каждой странице по мере перемещения бегунка.

В то же время типичная полоса прокрутки имеет достаточно ограниченные функциональные возможности. Чтобы помочь нам в навигации внутри документов, она должна предоставить нам мощные инструменты для мгновенных перемещений. Например:

- предусмотреть кнопки для перехода к нужной странице/главе/разделу/ключевому слову;
- предусмотреть кнопки для перехода в начало или в конец документа;

- дать возможность устанавливать закладки, к которым можно будет быстро вернуться.

Полосы прокрутки последних версий Microsoft Word содержат большинство из этих возможностей.

Если отвлечься от недостатков контекстуального характера, одна из самых серьезных проблем полосы прокрутки состоит в том, что она требует прецизионных движений мышью. Прокрутка документа *в одну сторону* (вниз или вверх) – обычно гораздо более простая операция, чем прокрутка документа *туда-сюда*. Вам приходится позиционировать курсор более внимательно, неизбежно отвлекаясь от содержимого листаемого документа. Некоторые полосы прокрутки дублируют обе кнопки, управляющие прокруткой вверх и вниз, на обоих концах полосы. Для полноэкранных окон это может быть удобным, но для маленьких окон такое дублирование элементов управления, вероятно, излишне и просто загромождает экранное пространство (более подробное обсуждение этой идиомы содержится в главе 19 и на рис. 19.1).

Повсеместное распространение полос прокрутки, к сожалению, привело к некоторому злоупотреблению ими. Самое распространенное злоупотребление связано с навигацией во времени. Не вдаваясь в философию и теологию, мы, пожалуй, можем сойтись на том, что время не имеет осмысленного начала или конца (по крайней мере для человеческого мозга). Какое же в этом случае значение имеет перемещение бегунка в один из концов полосы прокрутки календаря (рис. 21.21)?

Существуют жизнеспособные альтернативы полосам прокрутки. Одна из лучших альтернатив – **навигатор документа** – небольшая миниатюра полного документа, обеспечивающая возможность прямой навигации в различные точки этого документа (рис. 21.22). Во многих приложениях для работы с изображениями (к примеру, в Photoshop) данный элемент управления применяется для навигации по документу, выведенному в крупном масштабе. Навигаторы могут быть очень полезны и при работе с временны́ми шкалами аудио- и видеопотоков. Успех подобных идиом в первую очередь кроется в их способности выразительно представлять общую картину документа. По этой причине навигатор не всегда годится для представления длинных текстовых документов. В таких случаях полезной альтернативой полосам прокрутки может стать структура документа. Простейший пример этого подхода – представление структуры в Microsoft Word (задумка хорошая, но результат имеет ограниченную пригодность, поскольку разработчики решили, что отображения заслуживают только заголовки первого и второго уровня).

Разделители

Разделители – удобный инструмент для разделения главного окна приложения на несколько связанных между собой панелей, в каждой из которых можно просматривать, изменять или переносить ту или

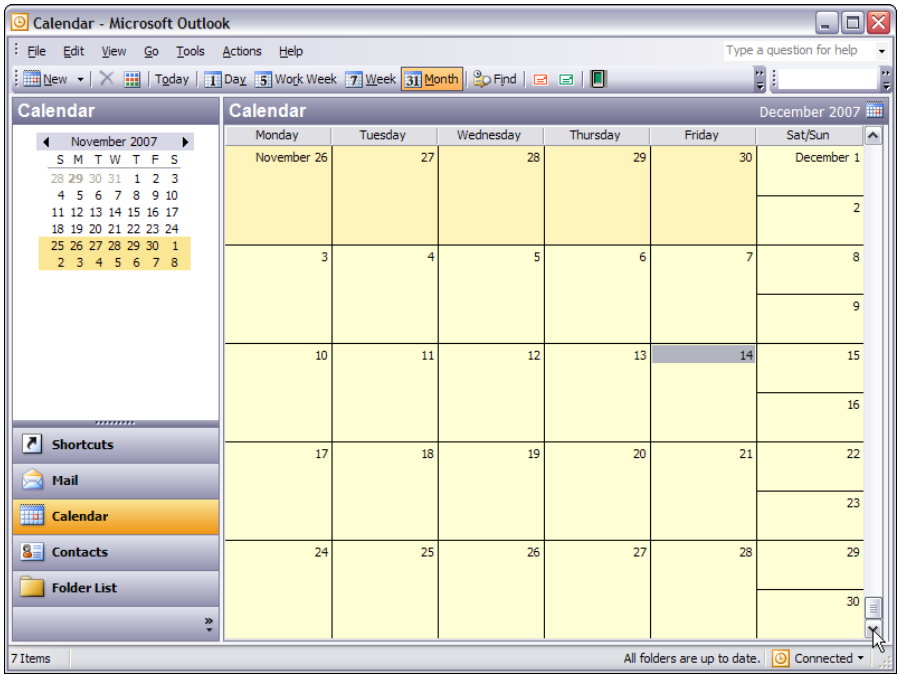


Рис. 21.21. *Вот демонстрация недостатков использования полосы прокрутки в целях навигации в бесконечном потоке времени. Перетаскивание бегунка в самый конец полосы прокрутки переносит пользователя на один год в будущее. Такое решение кажется взятым с потолка и накладывает ненужные ограничения*

иную информацию. Подвижные разделители всегда должны сообщать о своей подвижности посредством изменения формы курсора. Существует соблазн сделать все разделители подвижными, и это делается достаточно просто. Однако следует проявлять осторожность, выбирая, какие именно разделители должны стать подвижными. В общем случае разделитель не должен перемещаться таким образом, чтобы содержимое панели становилось непригодным к использованию. Когда некоторые области окна необходимо сворачивать, выдвижная панель может оказаться более подходящей идиомой.

Выдвижные панели и рычажки

Выдвижные панели – это панели монопольного приложения, которые можно открывать и закрывать в одно действие. Они могут использоваться совместно с разделителями, когда площадь выдвижной панели может изменяться пользователем. Обычно выдвижная панель открывается нажатием на расположенный рядом элемент управления. Этот элемент управления должен быть постоянно на виду и может быть

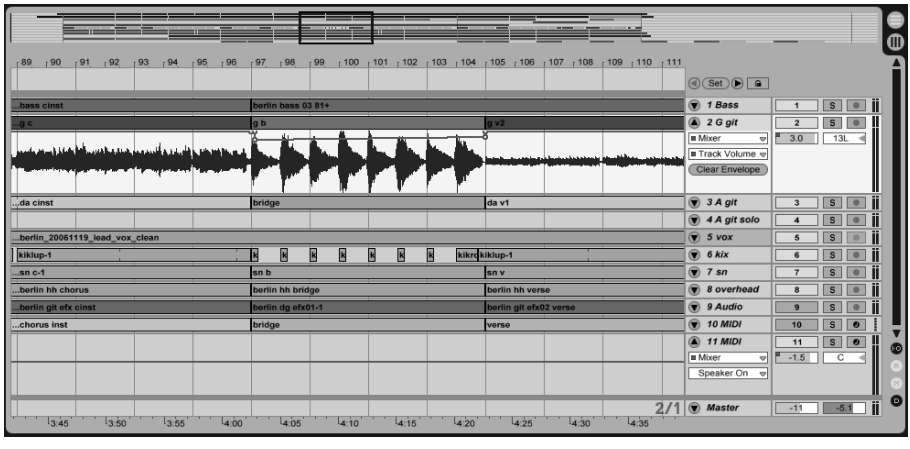


Рис. 21.22. *В Ableton Live навигатор документа располагается вверху экрана аранжировки и дает обзор всей композиции. Черная рамка обозначает ту часть композиции, которая в данный момент видна в рабочей области. Навигатор снабжает пользователя контекстной информацией, не позволяя ему потерять место в партитуре, а также предлагает идиому прямой навигации – пользователю достаточно просто переместить рамку в навигаторе, чтобы перейти к другому фрагменту композиции*

представлен или бинарной кнопкой (либо кнопкой-значком), или же **рычажком**, который ведет себя подобным же образом, но при этом поворачивается, чтобы показать, что выдвижная панель открыта или, соответственно, закрыта.

Выдвижные панели – замечательное место для элементов управления и функций, которые используются совместно с основной рабочей областью приложения, но не столь часто. Выдвижные панели более удобны, чем диалоговые окна, так как не закрывают основное окно. Описания свойств, списки объектов или компонентов со встроенными поисковыми фильтрами, хронологические списки – вот первые кандидаты на размещение на выдвижных панелях.

Хотя общие принципы, обсуждавшиеся в этой книге, способны дать значительное преимущество при создании продуктов, которые будут нравиться пользователям и приносить им удовлетворение, важно помнить, что все дело в мелочах. Неудобные элементы управления создают постоянный фон раздраженности, даже если общая концепция продукта просто превосходна. Убедитесь, что вы расставили все точки над i, и обеспечьте корректное поведение всех элементов управления.

22

Меню

Меню – вероятно, самая древняя идиома пантеона графических пользовательских интерфейсов. Она окружена почетом, суевериями и научными знаниями. Большинство проектировщиков, программистов и пользователей безоговорочно верят в правильность традиционных меню – ведь она доказана существованием огромного количества программ. Но эта вера походит на попытки отгонять тигра, щелкая пальцами. Вы говорите – никаких тигров здесь нет? Ну вот, щелчок пальцами работает! В определенных случаях хорошо спроектированное меню может быть крайне полезным способом обеспечения доступа к функциям приложения. Мы начнем главу с краткой истории вопроса, а затем обсудим проблемы, связанные с меню, и рассмотрим правильные способы их использования.

Немного истории

Сегодняшние графические пользовательские интерфейсы и диалоговые окна активно используются лишь с 1984 года, когда они были представлены на платформе Macintosh. Однако они получили такое широкое распространение, что их легко принять за данность. Прежде чем перейти к подробному рассмотрению современных идиом меню, будет полезно оглянуться назад и окинуть взглядом путь, по которому мы пришли к этим идиомам. Это позволит нам лучше понять преимущества и возможные недостатки меню.

Интерфейс командной строки

В семидесятые годы прошлого века, чтобы пообщаться с ЭВМ производства IBM, вы должны были вручную набить на перфораторе колоду перфокарт, воспользовавшись запутанным языком JCL (job control language – язык управления заданиями), и вставить эту самую колоду

перфокарт в шумное механическое устройство считывания. Каждая строка программы на языке JCL должна была быть пробита на отдельной перфокарте. Даже первые микрокомпьютеры, маленькие, медленные и глупые, работавшие под управлением примитивной операционной системы CP/M, были гораздо приятнее в общении, чем эти неповоротливые динозавры в охлаждаемых стеклянных пещерах. С микрокомпьютерами под управлением CP/M можно было общаться, набирая команды со стандартной клавиатуры. Это же чудо! Программа выводила на экран приглашение к вводу, которое выглядело следующим образом:

```
A>
```

В ответ на приглашение можно было вводить имена программ, хранившихся в виде файлов, а также обычные команды – и CP/M исполняла их. Этот интерфейс мы называли **интерфейсом командной строки** и считали его огромным шагом вперед в организации общения между машиной и человеком.

Единственный недостаток состоял в том, что вы должны были заранее знать, что вводить. Для опытных пользователей, которые в те времена главным образом были представлены программистами, командная строка была чрезвычайно мощным и эффективным средством, поскольку предоставляла самый быстрый и самый эффективный способ исполнения требуемых задач. Положив руки на клавиатуру и пользуясь преимуществами слепого набора, пользователь мог быстро задать команду `copy a:*.* b:`, чтобы скопировать диск. Даже сегодня, обладая необходимыми знаниями о наборе команд, пользователи могут быстрее (и часто эффективнее) решать задачи из командной строки, чем с помощью многочисленных операций, выполняемых мышью.

Интерфейс командной строки действительно делил пользователей на мужчин (и женщин) с одной стороны и компьютерных ботаников с другой. Но с ростом возможностей и сложности программного обеспечения интерфейс командной строки стал создавать слишком большую нагрузку на память – и настало время уступить дорогу чему-то более совершенному.

Последовательные иерархические меню

Наконец где-то в конце 70-х годов некоторым очень смышленым программистам пришла в голову идея предложить пользователю список возможных вариантов на выбор. Такой список можно было бы читать и выбирать из него пункты, как блюда из ресторанного *меню*. Название прижилось – и так началась эра **последовательных иерархических меню**.

Последовательные иерархические меню позволяли пользователям забыть большую часть команд и параметров, требуемых интерфейсом командной строки. Вместо того чтобы постоянно удерживать в голове

всю необходимую информацию, пользователь мог получать ее с экрана. Еще одно чудо! В районе 1979 года удобство вашей программы оценивалось по наличию в ней меню. Производителям программного обеспечения, застрявшим в мире командной строки, пришлось подвинуться на обочину, уступая дорогу более современной парадигме.

Хотя в то время парадигму и называли *меню*, мы называем меню того времени *последовательными* и *иерархическими*, чтобы отличать их от современных меню. До появления графического пользовательского интерфейса меню имели очень глубокую иерархию: выбрав пункт одного меню, пользователь получал другое, затем еще одно, и еще, опускаясь все ниже и ниже по иерархическому дереву команд.

Учитывая, что на экране в каждый конкретный момент времени могла быть выведена только одна страница меню, а программное обеспечение в то время все еще в значительной степени следовало традициям пакетного стиля вычислений больших ЭВМ, парадигма иерархических меню предполагала последовательное их отображение. Сначала перед пользователем выводилось меню верхнего уровня, из которого он мог выбрать одну из основных функций, например:

```
1. Ввести транзакцию
2. Закрыть данные за месяц
3. Напечатать отчет о доходах
4. Напечатать балансовую ведомость
5. Выйти
```

Когда пользователь выбирал одну из функций, например `Ввести транзакцию`, открывалось другое меню, которое учитывало его предыдущий выбор, например:

```
1. Внести счета
2. Внести оплату
3. Внести поправки к счетам
4. Внести поправки к оплате
5. Выйти
```

Выбрав из этого списка, пользователь обычно получал еще пару подобных меню, прежде чем появлялась возможность приступить непосредственно к выполнению работы. Пункт `Выйти` перемещал его на один уровень иерархии вверх. В результате навигация по меню превращалась в самую настоящую каторгу.

Сделав выбор, пользователь не имел возможности от него отказаться. Разумеется, люди постоянно совершали ошибки, так что наиболее прогрессивные разработчики того времени добавили меню с подтверждениями. Программа принимала выбор пользователя, как и раньше, но затем выводила еще одно меню: «Чтобы изменить ваш выбор, нажмите клавишу Escape, чтобы продолжить, нажмите клавишу Enter». Это стало настоящей головной болью, потому что независимо от того, сделали вы ошибку или нет, вы были обязаны дать ответ на этот неук-

люжий и обескураживающий вопрос, что само по себе могло привести к появлению той самой ошибки, которой вы надеялись избежать.

По сегодняшним стандартам такие меню были бы оценены как кошмарные. Их главный недостаток состоял в том, что даже при ограниченном наборе команд иерархии становились довольно глубокими. Им категорически недоставало гибкости и прозрачности в общении с человеком. И все же такие меню были лучше командной строки, где приходилось запоминать не всегда простой синтаксис команд, а также точное написание каждого операнда. Последовательные иерархические меню снизили нагрузку на память пользователей, но вынудили их плутать по лабиринтам бестолковых вариантов выбора и параметров. Эти меню должны были уступить место чему-то более совершенному. (Кстати, чтобы обнаружить современные устройства и информационные киоски, построенные на этой древней идиоме, далеко ходить не надо – возьмите хоть банкоматы.)

Интерфейс Lotus 1–2–3

Следующее серьезное усовершенствование в технологии пользовательского интерфейса было сделано корпорацией Lotus в 1979 году в ее оригинальной электронной таблице 1-2-3. Эта программа по-прежнему управлялась многоуровневыми иерархическими меню, но Lotus добавила кое-что свое – **видимые иерархические меню.** Именно эти меню сделали продукт самым успешным приложением того времени.

В 1979 году на экране компьютера могло уместиться ровно 2000 символов (рис. 22.1): 25 строк по 80 символов в каждой. Меню в 1-2-3 располагалось горизонтально в верхней части экрана и занимало всего две строки из 25 имеющихся. Это означало, что меню могло сосуществовать на экране вместе с самой электронной таблицей. В отличие от прежних программ, имевших иерархическое меню, здесь пользователь не был вынужден покидать основной экран, чтобы увидеть меню. Он мог выполнять команды меню, не прерывая работу в программе.

Рис. 22.1. *В электронной таблице Lotus 1-2-3, увидевшей свет в 1979 году, появилась новая примечательная структура меню, позволявшая в рамках одного экрана сосуществовать меню с рабочей областью. Другие программы того времени, работавшие на основе меню, вынуждали пользователя оставлять рабочий экран, чтобы сделать выбор из меню*

Lotus применяла новую идиому крайне энергично и создала структуру иерархических меню невероятных размеров. Дерево меню содержало десятки разделов и сотни пунктов. Пользователь мог найти любой пункт меню, взглянув на верхние строки экрана и перемещаясь вверх или вниз к требуемому пункту. Программа различала операции ввода данных для электронной таблицы и ввода команды меню по нажатию клавиши обратной косой черты (\). Если пользователь набирал этот символ, все последующие нажатия клавиш воспринимались как ввод команды меню, а не как данные. Чтобы выбрать пункт меню, достаточно было прочитать его и ввести первый символ названия этого пункта, предварив его символом обратной косой черты. После этого подменю замещало главное меню в верхней строке.

Опытные пользователи быстро обнаружили, что такие шаблоны легко запоминаются и необходимость в чтении меню отпадает. Они могли просто набрать `\-s`, чтобы сохранить данные на диск, или `\-c-g-x`, чтобы сложить числа в столбце. По сути, они вообще могли обходиться без меню. Они становились опытными пользователями, запоминая командные последовательности символов и радуясь своим знаниям о малопонятных функциях.

Сегодня такой подход может показаться наивным, но он иллюстрирует очень важный момент: хороший пользовательский интерфейс дает пользователям возможность постепенно наращивать свой опыт, превращаясь из новичков в экспертов. Опытный пользователь 1-2-3 мог до тонкостей знать пару десятков функций и вместе с тем ничего не знать о множестве других. Запомнив определенную командную последовательность, он мог далее постоянно использовать ее в своей работе. Реже используемые функции, командные последовательности для которых не отложились в памяти, он мог найти, читая текст меню. Далее в этой главе мы более подробно обсудим важность меню как средства обнаружения и изучения функций приложения.

Однако иерархические меню в 1-2-3 были ужасно сложными. Команд было просто-напросто слишком много, и каждую из них требовалось запихать в единую иерархическую структуру меню. Разработчики программы лезли из кожи вон, пытаясь выдумать логические связи между функциями в попытках оправдать распределение команд в иерархии. В горячке небывалого успеха и господства на рынке на такие вещи мало кто обращал внимание.

Как и следовало ожидать, вследствие небывалого успеха программы 1-2-3 имитации ее интерфейса получили в середине 80-х большое распространение. Постоянно видимые иерархические меню нашли свое место во множестве приложений, но в действительности эта идиома была последним вздохом символьного интерфейса – точно так же, как в конце 40-х годов большие, отлично сконструированные паровые локомотивы стали последним и самым совершенным монументом обреченной технологии. Подобно тому, как дизельные локомотивы полно-

стью вытеснили паровую тягу в течение десятилетия, графический пользовательский интерфейс всего за несколько лет вытеснил иерархические меню в стиле 1-2-3.

Раскрывающиеся и контекстные меню

Чтобы сделать возможным графический пользовательский интерфейс, понадобилось свести воедино множество концепций и технологий: мышь, быстрые видеокарты, мощные процессоры и всплывающие окна. Всплывающее окно – это прямоугольник на экране, который появляется, перекрывая основную часть экрана до тех пор, пока не выполнит свою функцию, после чего исчезает, оставляя нетронутым первоначальное изображение. Механизм всплывающих окон используется для реализации **раскрывающихся меню** (которые иногда называют выпадающими меню) и диалоговых окон.

В современных графических пользовательских интерфейсах меню располагаются у верхней границы экрана или окна в виде **строки меню**. Пользователь наводит курсор мыши на нужный ему пункт меню и щелкает по нему. В небольшом окне прямо под выбранным меню появляется перечень его пунктов. Разновидность раскрывающегося меню – меню, которое «всплывает», когда вы щелкаете (чаще всего правой кнопкой мыши) по выбранному объекту, даже при том, что этот объект не имеет заголовка меню, – называется **контекстным меню**.

После раскрытия меню пользователь может выбрать один из пунктов, щелкнув мышью или отпустив кнопку мыши в нужной точке после перетаскивания. В этом нет ничего примечательного – разве что обстоятельство, что глубина меню часто не превышает одного уровня вложенности. Выбор пользователя из такого меню приводит либо к немедленному выполнению какого-либо действия, либо к вызову диалогового окна. Если у меню только один уровень вложенности, иерархия становится «плоской». Во многих случаях – и особенно при оптимизации взаимодействия для новичков – плоская организация вариантов выбора (будь то команды или объекты) может значительно улучшить восприятие пользовательского интерфейса.

Возможно, наиболее значительным усовершенствованием меню в графических пользовательских интерфейсах стал этот переход от иерархической формы к моноклинальной группировке. Диалоговое окно – еще один пример всплывающего окна – послужило средством упрощения меню: оно дало разработчикам возможность вынести все возможные подварианты выбора любого пункта меню в самостоятельный интерактивный контейнер. Благодаря диалоговым окнам иерархия меню может быть значительно упрощена за счет перемещения всех мелких деталей из нижних уровней вложенности меню в отдельное диалоговое окно. Иерархические меню с глубокой вложенностью ушли в прошлое.

Благодаря высокому разрешению графического дисплея стало возможным поделить функции программы приблизительно на полдюжины осмысленных групп и представить каждую группу одним словом в строке меню. Меню для каждой группы также стало достаточно просторным, чтобы вместить в себя все связанные с группой функции. Потребность в дополнительных уровнях вложенности в меню практически отпала.

(Конечно, среди нас всегда находятся злодеи и негодяи – именно они придумали методы превращения раскрывающихся меню обратно в иерархические. Такие меню называют **каскадными меню**; хотя иногда они оказываются полезными, чаще каскадные меню просто заставляют более слабых духом разработчиков засорять свои меню с целью получения незначительного выигрыша. Более подробно мы обсудим эти меню чуть позже.)

Современные меню: средство обучения

С развитием современного графического пользовательского интерфейса роль меню в интерфейсе значительно изменилась благодаря двум идиомам – непосредственному манипулированию и панели инструментов. Развитие идиом непосредственного манипулирования было неспешным, но неудержимым – и началось с момента рождения графических пользовательских интерфейсов. Панель инструментов, напротив, стала новшеством, потрясшим отрасль в 1989 году. Буквально за пару лет практически в каждой программе для Windows появилась инструментальная панель, заполненная кнопками-значками. А всего несколькими годами раньше инструментальную панель никто и в глаза не видел.

Человек, не знакомый с городом, может долго плутать в поисках нужного адреса, в то время как уроженец этого города всегда будет идти по самому короткому маршруту. Так и опытные пользователи программы будут, как правило, вызывать ту или иную функцию самым простым и коротким способом – а не тем, который требует выполнения промежуточных шагов. Как следствие этого наиболее востребованные команды программы часто вызываются кнопками-значками на панели инструментов.

В конечном итоге меню все реже применяются для размещения функций, используемых ежедневно, и постепенно превращаются в способ освоения не столь часто используемых инструментов. Кнопки-значки и другие элементы управления на панели инструментов обычно дублируют команды меню. Кнопки-значки на виду, тогда как команды меню скрыты и доступ к ним осуществляется не так быстро. Однако у команд меню есть большое преимущество: они словесно описывают функции. Это ставит их в ряд наиболее полезных техник взаимодействия и делает хорошим **средством обучения** пользователей возможностям продукта.

Чтобы люди быстро учились пользоваться интерактивным продуктом, они должны иметь возможность исследовать и экспериментировать без опасения внести нежелательные изменения или нанести непоправимый ущерб. С обеспечением такой возможности прекрасно справляются функция отмены и кнопка Отмена в любом диалоговом окне. Вопреки парадигмам пользовательского интерфейса 20-летней давности, меню и диалоговые окна не следует делать основными средствами решения повседневных задач для обычных пользователей. Многие программисты и проектировщики до сих пор не осознали этого и продолжают путать основное назначение меню с его вспомогательной ролью. Основная роль меню – обучать новичков и помогать забывчивым пользователям, предоставляя возможность отыскать нечасто используемые функции.

Используйте меню для обучения пользователей.

Когда пользователь впервые сталкивается с программой, ему зачастую бывает сложно оценить ее возможности. Превосходный способ составить представление о возможностях и предназначении приложения – бегло просмотреть перечень функций, доступных в меню и диалоговых окнах. Точно так же мы просматриваем меню ресторана у входа, определяя, какие блюда в нем подаются, что представляет собой ресторан и какие в нем цены.

Понимание того, что лежит в круге решаемых программой задач, а что выходит за его пределы, – один из фундаментальных аспектов создания благоприятной атмосферы, способствующей обучению. Многие относительно удобные в работе программы отталкивают пользователя одним лишь тем, что не предлагают простого и безопасного способа изучить имеющиеся возможности.

Панель инструментов и идиомы непосредственного манипулирования могут оказаться для новичка слишком сложными, а текстовая природа меню позволяет пояснять функции словами. Надпись «Стили и форматирование» более понятна новичку, чем приведенная на рис. 22.2 кнопка-значок (хотя всплывающая подсказка «Панель форматирования», конечно, способна помочь).

Для пользователей, знакомых с приложением, но пользующихся им редко, основное назначение раскрывающихся меню и диалоговых

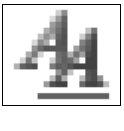


Рис. 22.2. *Пункт меню «Стили и форматирование», вероятно, будет более понятен новичкам, чем кнопка-значок вроде этой. Но после перехода пользователя в разряд середняков положение вещей в корне меняется*

окон – исполнять роль своего рода предметного указателя к инструментальным средствам: пользователь может заглянуть сюда, когда точно знает о существовании некоторой функции, но не может вспомнить, где она и как называется. В данном случае меню работает точно так же, как меню ресторана, которое позволяет человеку вновь вспомнить о существовании того восхитительного блюда из рыбы под соусом, которое он заказывал год назад, но теперь уже забыл его точное название. Раскрывающееся меню позволяет ему припоминать функции, названия которых он время от времени забывает. Он не должен запоминать такие мелочи, а может целиком и полностью положиться на меню, через которое можно получать доступ к нужным функциям по мере необходимости.

Если бы основной целью меню было исполнение команд, краткость была бы достоинством. Но основное назначение меню заключается в том, чтобы рассказать нам о доступных возможностях, о том, как ими воспользоваться и какие быстрые комбинации клавиш им соответствуют, так что краткость – противоположность того, что нам требуется на самом деле. Наши меню должны не только определять, где находится функция, но и объяснять, что она делает. Поэтому при создании меню нам выгодно быть более многословными. Так, вместо пункта «Открыть...» лучше использовать пункт «Открыть отчет...», а вместо «Упорядочить автоматически» – «Упорядочить значки автоматически». Следует воздерживаться от жаргонных выражений, поскольку пользователи нашего меню могут быть просто не знакомы с ними.

Во многих приложениях строка состояния, проходящая по нижней границе главного окна, используется также для отображения еще более длинной строки пояснительного текста, связанного с выбранным пунктом меню. Эта идиома способна повысить эффективность обучения, если пользователь будет знать о ней. Однако ее местоположение гарантирует, что информация, скорее всего, останется незамеченной.

Выступая в роли средства обучения, меню должны быть полными и предлагать исчерпывающий перечень действий и средств, доступных в программе. Просмотр меню должен давать ясную картину о назначении программы, глубине и охвате ее функциональных возможностей.

Еще одним обучающим моментом является наличие внутри меню подсказок, указывающих на альтернативные способы доступа к функциям программы. Повторение значков кнопок рядом с командами меню и подсказки, описывающие клавиатурные эквиваленты, приучают пользователей работать с более быстрыми средствами выполнения команд (более подробно мы поговорим об этом далее). Внедряя такую информацию непосредственно в меню, мы помогаем пользователю регистрировать ее подсознательно. Она не помешает его сознательному процессу, но, когда он будет готов пользоваться более эффективными средствами, информация окажется уже под рукой и будет ему знакома.

Стандартные меню для приложений рабочего стола

Большинство современных графических пользовательских интерфейсов включают в себя, как минимум, меню Файл и Правка, расположенные в крайней левой позиции, а также меню Справка, расположенное в крайней правой (в Mac OS X еще есть меню, название которого соответствует названию активного приложения; в строке меню оно предваряет пункты Файл и Правка). Руководства по стилям Windows, Macintosh и даже Motif утверждают, что меню Файл, Правка и Справка являются стандартными. Может показаться, что такая фактическая кроссплатформенность стандарта – достаточно сильное доказательство правильности идиомы. Неверно! Это лишь явный признак того, что сообщество разработчиков готово беспечно довольствоваться посредственным интерфейсным решением, отходя от него только тогда, когда конкуренты не оставляют выбора. Название меню Файл есть результат мышления, сосредоточенного на внутреннем устройстве операционных систем, меню Правка есть следствие крайней ограниченности буфера обмена, а меню Справка зачастую не особенно полезно и содержит лишь один элемент, который непосредственно связан с помощью пользователям.

Данные соглашения по именованию меню могут заманить нас в ловушку проектирования беспомощных пользовательских интерфейсов. Меню в большинстве программ могут быть знакомы пользователям, но насколько хорошо они систематизируют функции? Отдельные разделы, такие как Вид, Вставка, Формат, Сервис и Настройка, походят скорее на названия инструментальных средств и функций, чем на названия целей. Почему бы не организовывать функции по признаку целей?

Мы так и слышим возгласы программистов: «Как можно изменить то, что уже стало стандартом? Люди *привыкли* видеть меню „Файл“!». Ответ прост: люди способны привыкнуть и к боли, и к страданиям, но это не повод их продлевать. Пользователи быстро привыкнут, если мы изменим меню Файл таким образом, чтобы оно *лучше и точнее* отражало основное свое предназначение. Ключом к созданию разумной структуры меню является понимание ментальных моделей пользователей. Как они представляют себе выполняемые задачи? Какой терминологией пользуются? Если вы проектируете офисное бизнес-приложение и ваши пользователи близко знакомы с компьютером, возможно, имеет смысл использовать узнаваемые стандарты – по крайней мере, на верхнем уровне (но, возможно, и нет). Если же вы проектируете специализированное приложение для узкого круга пользователей, структура меню, скорее всего, должна быть совершенно другой.

Даже с учетом сказанного в жизни, определенно, есть место и для стандартных структур меню. Во многих случаях, когда реструктуризация меню приложения неуместна, можно добиться многого, просто разумно применяя стандарты.

Файл (или Документ)

Большинство пользователей задумывается о файлах лишь тогда, когда мы принудительно сталкиваем их с моделью реализации. Гораздо более уместным в рамках целеориентированного подхода будет название раздела меню «Документ». Если приложение работает с одним видом документов или объектов, полезно задуматься, стоит ли использовать такое название. Допустим, в музыкальном секвенсоре название «Композиция» будет гораздо более приятным и понятным для пользователя, чем «Файл». Смеем заверить: нет ничего хуже, чем думать о файловой системе во время сочинения музыки.

В главе 17 мы описали усовершенствованное меню Файл (или Документ). Мы не упомянули об одной полезной функции, которую стоит в него включить, – списке недавно открывавшихся документов, реализованном в приложениях Microsoft (рис. 22.3).

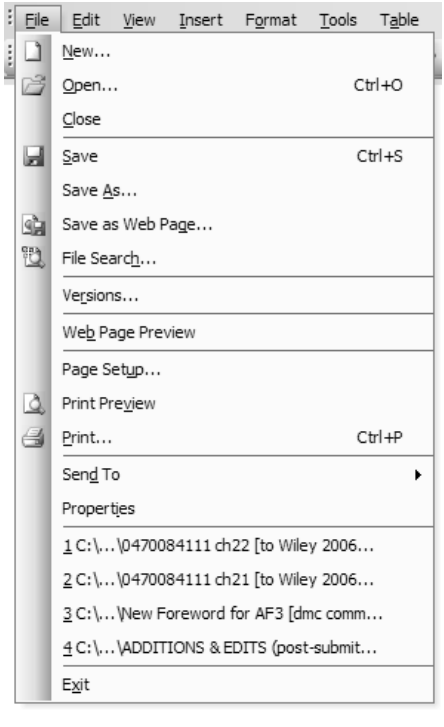


Рис. 22.3. *Меню Файл в Microsoft Word демонстрирует реализацию прекрасной идеи списка наиболее часто используемых документов. В главе 17 вы уже видели, как можно перестроить первые шесть пунктов меню, чтобы точнее отразить ментальную модель пользователя вместо бездумного следования устоявшейся модели реализации, которая показана здесь*

Правка

Меню Правка содержит в себе инструменты для выделения, вырезания, вставки и изменения выделенных объектов (если подобных функций много, их следует вынести в отдельное меню Изменить или Формат). Не используйте меню Правка для размещения функций, которые, как вам кажется, не входят в другие группы.

Окно

Меню Окно служит для упорядочивания открытых программой окон, их просмотра и переключения между ними. Здесь можно предложить также средства для одновременного просмотра нескольких открытых документов. Никакие другие функции в этом меню фигурировать не должны. Следует заметить, что данное меню редко необходимо в приложениях, в которых не реализован многодокументный интерфейс (MDI).

Справка

Современные меню Справка являются примерами дурно спроектированных и реализованных справочных систем. Мы будем много говорить о справке в главе 25, а сейчас хотим лишь заметить, что это меню должно содержать самые разнообразные инструменты, помогающие людям освоить ваше приложение. В частности, в большинстве меню Справка явно не хватает пункта Клавиатурные сокращения, объясняющего, как ликвидировать зависимость от меню. Здесь же можно сослаться на более мощные идиомы, такие как кнопки быстрого доступа, кнопки панели инструментов и идиомы непосредственного манипулирования.

Необязательные меню

Следующие меню используются достаточно часто, но рассматриваются большинством стилевых руководств как необязательные. В умеренно сложных приложениях, вероятно, потребуются хотя бы некоторые из этих меню.

Вид

Меню Вид должно содержать настройки, влияющие на способ представления данных пользователю. Кроме того, доступ ко всем дополнительным интерфейсным элементам, таким как линейки, панели инструментов, сетки, выдвижные панели, боковые панели, палитры, также должен осуществляться через это меню.

Вставка

Меню Вставка позволяет добавлять новые объекты в документ. В текстовом редакторе речь может идти, например, о вставке таблиц, гра-

фиков и символов. В музыкальном секвенсоре – о новых инструментах, эффектах, смене тональности.

Настройка

Если в вашем приложении имеется меню Настройка, вы пообещали пользователю, что любые настройки программы он сможет изменить через это меню. Не распределяйте отдельные настройки и диалоговые окна настройки по другим меню, если имеется меню Настройка. Это относится и к настройке параметров печати, которую часто по ошибке помещают в меню Файл.

Формат

Меню Формат – одно из самых бесформенных среди дополнительных меню, потому что оно почти всегда имеет отношение исключительно к свойствам визуальных объектов, а не к функциональным возможностям. В современном объектно-ориентированном мире свойствами визуальных объектов принято управлять с помощью идиом прямого манипулирования, а не функций. Меню играет обучающую роль, но имеет смысл рассмотреть возможность полного отказа от него, если при этом реализована более объектно-ориентированная схема управления свойствами.

Пункт Параметры страницы, который обычно находится в меню Файл, следует помещать в это меню. (Обратите внимание: «параметры страницы» – это далеко не то же самое, что «параметры печати».)

Сервис

Меню Сервис, которое иногда носит менее понятное название Параметры, является местом, откуда вызываются большие и мощные функции. Такие функции, как проверка правописания и средства расширенного поиска, считаются сервисными средствами. Кроме того, меню Сервис – это место, где должны размещаться пункты, отвечающие за выполнение **потенциально опасных действий**.

Потенциально опасные действия – это функции, которые должны вызываться только опытными пользователями. К ним относятся различные расширенные настройки. Например, допустим, что клиент-серверное приложение, работающее с базой данных, предоставляет удобные в работе идиомы непосредственного манипулирования для построения запросов к базе данных, за кулисами составляя соответствующие предложения на языке SQL. Возможность прямого редактирования SQL-запроса определенно является потенциально опасной функцией! Функции такого рода могут приводить к непредсказуемым последствиям или к нарушениям в работе программы, и по этой причине они визуально должны быть отделены от более мирных функций. Другой возможный подход – разместить эти функции в меню Эксперт, справа

от мирного меню Сервис. Так сделала Apple в iPhoto, хотя о корректности размещения конкретных функций в этом приложении можно поспорить.

Идиомы меню

За годы существования простые меню обросли новыми и более сложными поведенческими идиомами. Некоторые из них находят практическое применение, другие просто мешают. В этом разделе обсуждаются подобные идиомы меню и уместные применения этих идиом.

Каскадные меню

Разновидность раскрывающихся меню, которая позволяет рядом с основным активным меню выводить вторичное меню. Этот механизм называется **каскадные меню**, или **подменю** (рис. 22.4), и является крайне неудобным.

Если стандартные раскрывающиеся меню дают ясный и простой способ навигации по моноклинальным группам, каскадные меню переносят нас в опасную область уровней и иерархий. Многоуровневые подменю не только затрудняют запоминание расположения пунктов, но также требуют более высокой точности перемещения указателя мыши. (Если проследить путь, необходимый для выбора пункта в много-

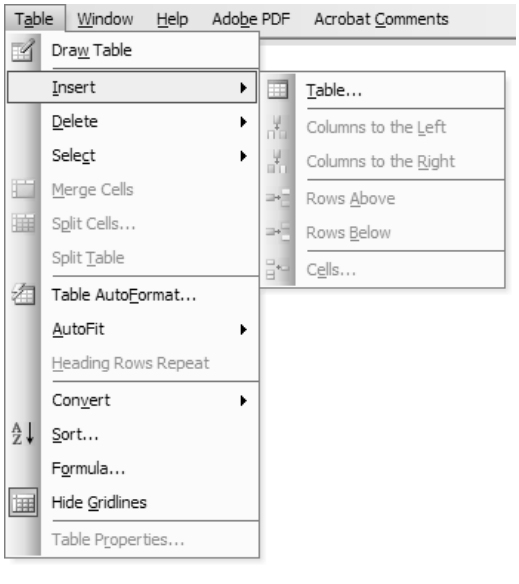


Рис. 22.4. *Этот пример каскадного меню взят из Microsoft Word 2003. Каскадные меню затрудняют поиск и просмотр команд, однако позволяют держать гораздо больше команд внутри групп*

уровневом каскадном меню, таком как меню Пуск системы Windows, вы заметите, что это буквально путь в лабиринте.)

Но и каскадные меню могут приносить пользу. Они позволяют умещать в меню гораздо большее количество команд и структурировать их иерархическим образом. Хотя на первый взгляд это полезная возможность, пожалуйста, прислушайтесь к пользователям, прежде чем реализовать эту идиому на практике.

Необходимо подчеркнуть, что каскадные меню следует применять лишь в сложных монопольных приложениях для редко используемых функций или в качестве вспомогательного средства выполнения команд, дополняющего основные методы, явно представленные в интерфейсе. Кроме того, создавая каскадные меню, позаботьтесь о допусках в движении мыши, чтобы подменю не исчезали неожиданно.

Адаптивные меню

В Microsoft Office 2000 появились **адаптивные меню**, которые отображают только пункты, наиболее часто используемые конкретным пользователем (рис. 22.5). Эта идиома по умолчанию была включена в пакетах Office 2000 и 2003, однако Office 2007 отступил от этой идеи (как и от меню вообще), введя так называемую ленту (ribbon bar), которую мы обсудим далее и в главе 23.

Адаптивные меню были попыткой Microsoft сделать продукты более простыми с виду, скрыв функции, которые пользователю никогда не

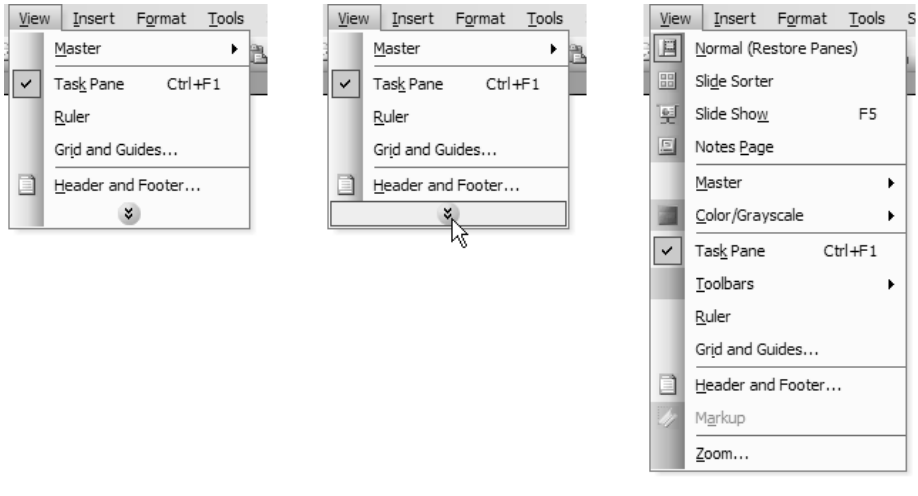


Рис. 22.5. *Эти изображения демонстрируют работу адаптивных меню PowerPoint 2003. Слева – меню в своем обычном состоянии, оно содержит подмножество наиболее часто выполняемых команд. По центру – меню с курсором, наведенным на кнопку раскрытия. Справа – результат нажатия на кнопку*

требуются. Чтобы увидеть скрытые пункты меню, пользователь должен щелкнуть по пиктограмме внизу меню или оставить над ней курсор на некоторое время. Скрытые элементы, отображаясь, перемешиваются с теми, которые были видимыми и ранее.

Адаптивные меню хороши по замыслу, и мы приветствуем попытки подстраивать интерфейс исходя из поведения пользователей. К сожалению, идиома получилась крайне неудобной и значительно снижает производительность труда человека. Адаптивные меню ощутимо увеличивают объем работы, выполняемой пользователями, поскольку эта идиома противоречит двум функциям системы меню: демонстрировать пользователям охват и глубину функциональности приложения и обеспечивать доступ к менее востребованным функциям.

Следует отметить, что юзабилити-исследования поддерживают нашу оценку. В исследовании 1989 года респонденты тратили значительно больше времени на решение задач посредством адаптивных меню, чем посредством статических, и 81% участников заявили, что предпочитают именно статические меню (Mitchell and Shniederman, 1989). Хотим сразу же предостеречь читателей: из результатов исследования не следует делать вывод, что пользователям никогда не будут по душе адаптивные интерфейсы. Скорее, разумно будет предположить, что в данном конкретном случае адаптивность просто противоречила назначению меню.

Лента

В пакете Office 2007 компания Microsoft представила новый командный элемент управления – **ленту**. Это расположенная у верхней границы окна панель с вкладками, сочетающая в единой структуре меню и панели инструментов (рис. 22.6). Идея ленты заключается в сочетании визуально выразительных свойств кнопок инструментальных панелей с подробным, более понятным и полным отображением функциональности, присущим системе меню.

В отличие от адаптивных меню лента, похоже, проектировалась с учетом сильных сторон как меню, так и панелей инструментов. Как и в случае меню, просмотр вкладок ленты обеспечивает хороший обзор воз-

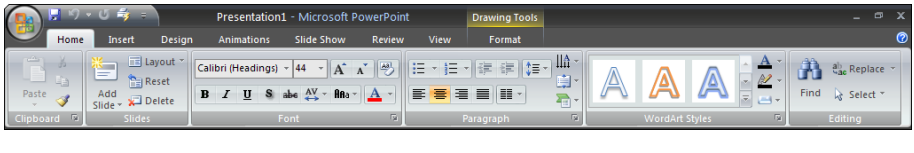


Рис. 22.6. *На рисунке представлена лента программы Microsoft PowerPoint 2007. Эта новая интерфейсная идиома сочетает иерархическую организацию системы меню с более прямолинейным и наглядным представлением, присущим панели инструментов*

можностей приложения (хотя визуальная насыщенность и вертикальная организация ряда элементов несколько замедляют такое сканирование). Подобно обычной панели инструментов, лента предоставляет достаточно прямой и наглядный доступ к функциям, хотя пользователю иногда приходится переключаться между вкладками, чтобы найти нужную кнопку. В некоторой степени его жизнь облегчает тот факт, что непосредственно рядом с выбранным объектом в рабочей области документа присутствует контекстная панель инструментов (более подробно читайте в главе 23), и вкладки автоматически сменяются, предоставляя доступ к нужным инструментам. Идиома ленты привлекательна и интересна, однако похоже, что в первых версиях Office 2007 она не до конца раскрыта. Поработав немного с этим пакетом, мы обнаружили, что постоянно охотимся за востребованными функциями. Возможно, ситуация улучшится: на момент написания этого текста пакету Office 2007 было всего шесть месяцев от роду.

Меню моментального действия

На заре существования платформ Mac и Windows в некоторых программах встречалась разновидность меню, которая по вполне понятным причинам вышла из употребления: **мгновенное меню**, или **меню с восклицательным знаком (bang menu)**. На жаргоне программистов слово **bang** означает «**восклицательный знак**», и, согласно устоявшемуся соглашению, название мгновенного меню верхнего уровня содержало восклицательный знак.

Как следует из названия, это обычный заголовок меню верхнего уровня; он располагается в строке меню рядом с другими заголовками меню, но ведет себя как *пункт меню*. Вместо того чтобы отображать раскрывающееся меню для последующего выбора, меню с восклицательным знаком немедленно выполняет свою функцию по щелчку мыши! Например, меню мгновенного действия, выполняющее печать, могло называться Печать!.

Такое поведение настолько неожиданно, что сразу же дезориентирует пользователя, а иногда и злит его. Заголовок меню с восклицательным знаком не имеет практически никакой педагогической ценности (разве что в качестве шоковой терапии). Такое меню только приводит пользователя в замешательство. А вот аналогичное незамедлительное действие кнопки на инструментальной панели никого не беспокоит. Разница в том, что кнопки-значки на панели инструментов подразумевают немедленное исполнение функции, потому что воспринимаются как *командные элементы управления*. Это классический пример того, как отход от стандартов и соглашений без достаточных причин способен создать значительное когнитивное сопротивление. Команды мгновенного действия должны располагаться на панелях инструментов.

Заблокированные пункты меню

Важное соглашение, касающееся меню, требует, чтобы пункты, недоступные в определенный момент времени или не имеющие отношения к работе с выбранным объектом, становились заблокированными. Заблокированное состояние обычно кодируется с помощью осветления текста пункта меню. Это полезная и ожидаемая идиома – она способствует выполнению обучающей функции меню, поскольку пользователям становится проще понимать, в каких контекстах бывают доступны те или иные команды.

Блокируйте пункты меню всегда, когда их функции теряют смысл.

Помечайте галочками пункты меню

Галочки рядом с пунктами меню применяются для переключения представлений интерфейса программы (например, для включения и выключения инструментальных панелей) или подстройки параметров отображения объектов (например, для вывода объектов в виде рамки или вместе с содержимым). Пользователи легко схватывают эту идиому. Она эффективна, потому что не просто создает командный элемент управления, но и указывает на состояние этого элемента.

Вероятно, данную идиому лучше всего применять в программах, имеющих достаточно простую структуру меню. В более сложном и изощренном приложении пространство меню будет дефицитным ресурсом: открывать и шерстить меню в поисках нужного пункта может стать утомительным для пользователя. Если параметры переключаются часто, они должны располагаться на панели инструментов. Если доступ к ним требуется лишь время от времени, а пространство в меню – на вес золота, все подобные атрибуты можно собрать в диалоговом окне, которое способно предоставить контекст и дополнительные инструкции (что будет весьма полезно для нечасто используемой функциональности).

Пункты меню с галочками гораздо более предпочтительны, чем **триггеры**, изменяющие свое состояние и всегда показывающие, какое состояние *не выбрано*. Проблема с пунктами-триггерами в меню точно такая же, как с кнопками-триггерами (глава 21): пользователь не может понять, что перед ним – выбор или описание состояния. Если пункт меню содержит слова «Отображать панель инструментов», означает ли это, что инструменты видны сейчас – или что выбор этого режима сделает их видимыми? Используя пункт меню с галочкой (наличие галочки означает, что панель видна), вы избавитесь от двусмысленности.

Пиктограммы в меню

Значки рядом с текстовыми надписями помогают пользователю узнавать функции, не читая текст меню, благодаря чему работа выполняется быстрее. Кроме того, они позволяют выработать полезные визуальные ассоциации с другими элементами управления, решающими те же самые задачи. Чтобы создать выразительный визуальный язык, снабжайте пункты меню такими же пиктограммами, что и соответствующие им кнопки на панели инструментов.

Используйте одинаковые пиктограммы для элементов управления, решающих одни и те же задачи.

Операционная система Windows предоставляет инструментальные средства для размещения графических изображений в меню. Однако очень немногие программы пользуются этой возможностью, чтобы создать простые и наглядные средства обучения. Например, все приложения из пакета Microsoft Office используют пиктограмму с изображением пустого листа бумаги для описания функции Новый документ на инструментальной панели. Тот же самый значок Microsoft поместила в меню Файл слева от пункта каскадного меню Создать→Пустой документ. Пользователь выстраивает ассоциацию между ними, скорее всего даже не подозревая об этом. В приложениях Office Microsoft проделала отличную работу по интеграции графики в меню, о чем наглядно свидетельствует рис. 22.3.

Клавиши быстрого доступа

Клавиши быстрого доступа, или «клавиатурные сокращения», – это простой способ вызывать функции с клавиатуры. Обычно такими клавишами становятся функциональные (например, F9) или сочетания клавиш, включающие служебные клавиши, в частности, Ctrl, Alt, Option и Command. По общепринятому стандарту, они выводятся справа от пунктов раскрывающихся меню, что позволяет пользователям изучать их в ходе работы с меню. Существуют стандарты клавиш быстрого доступа для Windows, Mac OS X и других платформ, однако реализация в каждом конкретном случае остается на совести проектировщика, и о клавишах быстрого доступа слишком часто забывают.

Следующие три совета помогут вам в создании удачных клавиатурных сокращений:

1. Следуйте стандартам.
2. Предусмотрите возможность повседневного их использования.
3. Покажите, как с ними обращаться.

Если для операции существуют стандартные клавиатурные сокращения, используйте их. Это, в частности, относится к набору операций

редактирования, которые можно наблюдать в меню Правка. Пользователи быстро начинают понимать, насколько проще воспользоваться комбинациями Ctrl + C и Ctrl + V, чем с помощью мыши открыть меню Правка, выбрать пункт Копировать, затем снова выбрать меню Правка и далее пункт Вставить. Не разочаровывайте пользователя при работе с вашей программой. Не забывайте и другие стандартные сочетания, такие как Ctrl + P для выполнения операции печати и Ctrl + S для сохранения документа.

Определить набор команд, которые будут использоваться ежедневно, порой бывает достаточно сложно. Необходимо отобрать те функции, которые, скорее всего, будут использоваться достаточно часто, и не забыть внедрить клавиши быстрого доступа в соответствующие им пункты меню. Хорошая новость: этот набор не будет слишком большим. Плохая новость: у каждого пользователя этот набор может быть своим.

Наилучший способ таков: создайте перечень доступных в приложении операций и разделите его на три группы – те операции, которые точно будут использоваться в повседневной работе любого пользователя, те, к которым наверняка будут обращаться редко, и все остальные. Операции из первой группы обязательно должны иметь клавиатурные сокращения, а операции из второй не должны иметь таковых. Сложнее всего будет разобраться с последней, третьей группой, и она неизбежно окажется самой большой. Вы можете повторно поделить эту группу на подгруппы и назначить наиболее часто используемым функциям наиболее простые клавиши доступа, такие как F2, F3, F4 и т. д. Более сложные для запоминания сочетания, такие как Alt + 7, следует назначать функциям, которые с меньшей вероятностью попадут в разряд операций повседневного использования.

Не забывайте отображать клавиатурные сокращения в меню. От сокращения будет мало толку, если пользователь сможет узнать о его существовании, только обратившись к руководству или оперативной справке. Размещайте клавиатурные сокращения в пунктах меню с правой стороны, где им и место. Поначалу пользователи не обратят на них внимания, но в конечном счете обнаружат и будут счастливы этому открытию уже как вечные середняки (см. главу 3). Это повысит их самооценку и даст возможность почувствовать себя человеком просвещенным. Подарите вашим клиентам это ощущение – результат того стоит, поверьте.

Некоторые программы предоставляют пользователям возможность самостоятельно настраивать клавиатурные сокращения, и во многих случаях это неплохая идея – и даже необходимость (особенно для опытных пользователей). Возможность настраивать клавиши быстрого доступа в монопольных приложениях, работа с которыми занимает большую часть времени, позволяет пользователям адаптировать программное обеспечение к своему стилю работы. Не забудьте наряду с другими функциями добавить в настройки возможность вернуться к значениям по умолчанию.

Мнемоники

Мнемоники – это еще один стандарт Windows (встречается также в графических интерфейсах UNIX), позволяющий выполнять команды с клавиатуры – в параллель с непосредственным управлением диалоговыми окнами и меню.

Стилевое руководство от Microsoft подробно описывает как мнемоники, так и клавиатурные сокращения, поэтому мы лишь подчеркнем, что их не следует упускать из виду. К мнемоникам обращаются с помощью клавиши <Alt>, клавиш управления курсором и клавиши, обозначаемой подчеркнутым символом в пункте меню или в заголовке. Нажатие клавиши Alt переводит приложение в режим ввода мнемоники, а курсорные клавиши могут использоваться для перемещения к нужному меню. После того как меню откроется, нажатие назначенной пункту меню клавиши приводит к выполнению соответствующей функции. Основное назначение мнемоник – обеспечить наличие клавиатурного эквивалента для каждой команды меню. По этой причине мнемоники должны быть определены для всех пунктов меню – особенно в приложениях, ориентированных на работу с текстом. Считайте их не столько удобством, сколько связью меню с клавиатурой. Помните: самые опытные пользователи в значительной степени полагаются на клавиатуру, поэтому дорожите их преданностью и предоставьте им непротиворечивые и тщательно продуманные мнемоники. Мнемоники нельзя считать факультативным элементом интерфейса.

Меню на других платформах

Наше изложение до сих пор касалось в основном подходов и стандартов традиционных платформ для настольных компьютеров. Разумеется, ситуация меняется при переходе к другой платформе. В таких устройствах, как мобильные телефоны и портативные компьютеры, часто приходится полагаться на меню как на основное средство выполнения команд. И хотя приложения в портативных компьютерах обычно представляют некоторые функции кнопками, из-за ограничений экранного пространства не всегда возможно использовать инструментальные панели и более прямолинейные идиомы. Потому для большинства функций единственным способом обеспечить к ним доступ является именно система меню.

В PalmOS и Windows Mobile присутствуют раскрывающиеся меню, во многом похожие на те, которые применяются в графических пользовательских интерфейсах настольных компьютеров, – у этих меню даже есть клавиши быстрого доступа. Разумеется, такой перенос полезен и оправдан, однако следует помнить, что некоторые идиомы не слишком хорошо ведут себя на маленьких экранах и могут не дружить с перьями и джойстиками. Во-первых, любой ценой следует избегать каскадных меню. Разместить на маленьком экране два открытых меню бок о бок практически невозможно, а если одно меню будет распола-

гаться поверх другого, это запутает и дезориентирует пользователя мобильного устройства. Во-вторых, ограничения экранного пространства могут лишить вас возможности снабжать пункты меню пиктограммами – а ведь пользователю, который идет пешком, гораздо проще идентифицировать элементы меню по пиктограммам, нежели по тексту.

Интерфейс мобильных телефонов и прочих устройств с маленькими экранами (скажем, глюкометров) по уровню развития еще не вступил в эпоху раскрывающихся меню, взаимодействие в нем реализуется посредством последовательных иерархических меню (о которых мы говорили ранее в этой главе). При проектировании для такой платформы набор инструментов весьма ограничен, и каждое принятое решение обретает большое значение. В приложениях, где варианты выбора ассоциированы с номерной клавиатурой (и где часть вариантов можно задвинуть в подменю «Другие»), принципиальное значение имеет ассоциирование наиболее востребованных функций с номерными клавишами. Последовательность функций должна быть логичной (сходные функции должны располагаться рядом), а наиболее востребованные функции необходимо помещать в начало списка.

23

Панели инструментов

Применяемые повсеместно панели инструментов – относительно новое слово в графических пользовательских интерфейсах. В отличие от многих других интерфейсных идиом, заслуга популяризации которых принадлежит компьютерам Apple Macintosh, панели инструментов впервые были использованы в интерфейсах популярных приложений компании Microsoft. Будучи важным дополнением к системе меню, панель инструментов зарекомендовала себя как эффективный механизм надежного и непосредственного доступа к функциям. В то время как меню содержит всеобъемлющие наборы инструментов и ключевой своей функцией имеет обучение, панели инструментов предназначены для наиболее востребованных команд и не очень-то помогают новичкам.

В этой главе мы обсудим достоинства и недостатки идиомы команд панели инструментов. Кроме того, мы поговорим о всплывающих подсказках и вариантах инструментальных панелей, таких как лента.

Панели инструментов: наглядные, мгновенно исполняемые команды

Типичная панель инструментов представляет собой набор кнопок-значков (пиктограмм, служащих в качестве кнопок), как правило, без текстовых подписей, расположенных на горизонтальной подложке непосредственно под строкой меню или на вертикальной подложке, прикрепленной к одной из вертикальных границ основного окна (рис. 23.1). По сути, инструментальная панель представляет собой одну строку (или колонку) всегда видимых графических элементов, использование которых приводит к немедленному выполнению действий.

Замечательные идеи в области проектирования пользовательского интерфейса часто рождаются во многих умах одновременно. Панель инструментов не является исключением: она появилась во многих про-

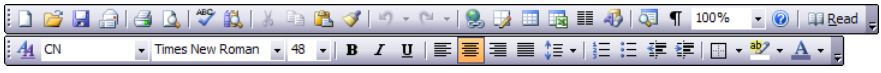


Рис. 23.1. *На этом изображении представлены панели инструментов Microsoft Word 2003: Стандартная и Форматирование. Обратите внимание, что кнопки-значки на панели инструментов не имеют ассоциированных с ними видимых подписей – это просто кнопки. Так экономится пространство и повышается читаемость*

граммах приблизительно в одно и то же время, и неизвестно, кто был первопроходцем. Ясно одно: преимущества панели инструментов немедленно стали очевидны всем. Ее изобретение одним махом решило проблемы раскрывающихся меню. Функции на панели инструментов всегда на виду, и достаточно одного щелка мыши, чтобы воспользоваться любой из них.

Панели инструментов и меню

Панели инструментов часто считают вариантом меню с ускоренным доступом к функциям. Такой аналогии трудно избежать: панели инструментов предоставляют доступ к функциям программы и оформлены обычно в виде горизонтальной строки в верхней части окна. Некоторые проектировщики, похоже, считают, что панели инструментов не просто служат дополнительным средством выполнять команды, а *идентичны* меню. Им кажется, что функции, доступные на инструментальных панелях, должны быть теми же самыми, что и в меню.

В действительности же во многих случаях панели инструментов должны преследовать совершенно иные цели, чем меню. Инструментальные панели и элементы управления на них следует проектировать так, чтобы обеспечить быстрый доступ к часто используемым функциям тем пользователям, которые уже овладели основными приемами работы с приложением. В силу своей лаконичности панели инструментов – обычно отнюдь не лучший способ сообщить начинающим о возможностях и принципах работы с приложением (хотя всплывающие подсказки в некоторой степени способны смягчить ситуацию). Меню предлагают гораздо более выразительный и полный обзор приложения и часто являются более подходящим средством обучения новичков.

Инструментальные панели обеспечивают возможность быстрого доступа к часто используемым функциям для опытных пользователей.

Мощь меню – в их многословности и полноте. Все, что необходимо пользователю, он может найти где-то в меню программы. Разумеется, именно это изобилие приводит к тому, что меню становятся большими и громоздкими. Чтобы предотвратить расходование экранного пространства,

меню должны большую часть времени находиться в свернутом состоянии и раскрываться только по запросу. Необходимость открывать меню не позволяет считать, что они содержат моментально исполняемые команды, которые всегда на виду. Меню есть компромисс: исчерпывающие подробности и мощь в обмен на пусть и небольшую, но неизменную дозу неудобств. С другой стороны, кнопки-значки на панелях инструментов не являются исчерпывающими и очевидными. При этом они постоянно находятся на виду, обеспечивают прямой доступ к связанным с ними функциям и обладают более высокой пространственной эффективностью по сравнению с меню.

Панели инструментов и элементы управления на панелях инструментов

Панель инструментов подарила нам **кнопку-значок** – счастливый брак между кнопкой и пиктограммой. Как визуальные мнемоники функций кнопки-значки превосходны. У начинающих пользователей они могут вызывать определенные сложности при интерпретации, но ведь они и *не предназначены* для новичков.

Пиктограммы и текст на панелях инструментов

Если кнопки-значки на панели инструментов действуют точно так же, как пункты раскрывающегося меню, то почему пункты меню практически всегда отображаются как текст, а кнопки почти всегда содержат лишь небольшие изображения? Хотя это различие было создано скорее неосознанно, за ним стоят весьма серьезные основания.

Текстовые метки, подобные тем, что мы видим в меню, могут быть очень точными и ясными (хотя являются такими не всегда). Обеспечивать точность и ясность – их основное назначение. Они могут выполнить свое предназначение, если пользователь будет читать их не торопясь и сосредоточенно. Как мы уже говорили в главе 14, чтение текста отнимает больше времени, чем распознавание изображений. В роли обучающего средства меню обязаны предлагать читателям точность и ясность: учитель, демонстрирующий неточность или невнятность, – плохой учитель. Дополнительные затраты времени и сил – разумная плата за обучение.

С другой стороны, качественные графические символы проще поддаются распознаванию, но им часто не хватает точности и ясности, присущей тексту. Пиктограммы могут быть неоднозначными, пока вы не изучите их смысл. Но единожды изучив этот смысл, вы уже не сможете легко его забыть – и распознавание будет происходить молниеносно, тогда как текст все равно каждый раз приходится прочитывать заново.

Панели инструментов в основном предназначаются для организации быстрого доступа к часто используемым инструментам, поэтому то, что позволяет их идентифицировать, должно быстро распознаваться опытными пользователями. Символы гораздо лучше подходят для решения этой задачи, чем текст. Кнопки-значки обладают отзывчивостью кнопок, а также быстро распознаются благодаря наличию изображений. Несмотря на малый размер, они обладают мощной функциональностью, но их самый большой плюс является по совместительству и самым большим минусом: речь идет о пиктограммах.

Полагаться на пиктограммы как средство передачи смысла разумно тогда, когда стороны заранее договорились о смыслах пиктограмм. Это совершенно необходимо, потому что значение пиктограммы любого вида по своей природе неоднозначно, пока не будет заучено. Многие проектировщики полагают, что должны изобрести для кнопок-значков такие визуальные метафоры, которые адекватно будут передавать их смысл начинающим пользователям. Это донкихотские поиски, которые отражают не только непонимание назначения панелей инструментов, но и тщетную надежду на волшебство метафор, которые мы обсуждали в главе 13.

Изображение на кнопке-значке не должно обучать пользователя, оно просто должно быть легко узнаваемым. Пользователи должны получать помощь в изучении назначения этих кнопок из других источников. Конечно, проектировщик обязан стремиться достичь обеих целей, но он не должен вводить себя в заблуждение. Очень часто практически невозможно убить двух зайцев сразу. Найти изображения, которые представляют *предметы*, намного проще, чем найти изображения, которые представляют действия или взаимоотношения. Изображение мусорной корзины, принтера или диаграммы интерпретируется довольно легко, но какой значок вы нарисуете, чтобы представить такие действия, как «применить выбранный стиль», «выполнить подключение», «преобразовать»? Когда доходит до дела, пользователь вполне может задаться вопросом: что означает изображение принтера? Оно может означать «поиск принтера», «изменение параметров печати» или «вывод сведений о состоянии принтера». Разумеется, после того как он узнает, что маленькое изображение принтера обозначает действие «напечатать одну копию текущего документа на принтере по умолчанию прямо сейчас», эти вопросы отпадут сами собой.

Проблема с подписями кнопок-значков

Идея разместить на инструментальной кнопке и текст, и графическое изображение может показаться весьма заманчивой. В подтверждение данной логики имеется исторический прецедент: первоначальные пиктограммы на рабочем столе Macintosh имели текстовые подписи, как и некоторые графические элементы управления в старых веб-броузерах. Пиктограммы полезны для быстрой классификации, но нам

все же необходим текст, чтобы *точно* понять, для чего предназначен каждый конкретный объект.

Проблема состоит в том, что одновременное использование текста и изображения является очень дорогим удовольствием в смысле экранного пространства. В большинстве случаев экранное пространство слишком ценно, чтобы расходовать его подобным образом. Принимая решение подписывать пиктограммы, проектировщики пытаются удовлетворить две группы пользователей, имеющие различные потребности. Одни хотят учиться в спокойной обстановке, не опасаясь ошибок; вторые знают, где острые углы, но им иногда требуется напоминание. Эффективным мостом между этими двумя группами пользователей становятся всплывающие подсказки.

Элементы управления на панели инструментов: обучение

Самая большая проблема, связанная с панелями инструментов, заключается в том, что, хотя расположенные на них элементы управления легко запоминаются и позволяют быстро работать, изначально их смысл совершенно непрозрачен. Каким образом начинающий пользователь сможет разобраться, что означают кнопки-значки и другие элементы управления, расположенные на панели инструментов?

Первая попытка: всплывающая справка

Именно компания Apple первой предложила решение, создав в операционной системе System 7 **всплывающую справку** (balloon help), которая выглядела как реплика персонажа из комикса и описывала назначение и результаты работы элемента интерфейса, над которым проходил курсор (эта функция известна также под названиями fly-over, roll-over, mouseover).

Несмотря на благие намерения проектировщиков, всплывающая справка была принята холодно. Из-за того, что справочная реплика появлялась без всякой паузы, стоило лишь курсору пройти над объектом, приложение становилось непригодным для использования. В результате система справки у Apple получилась режимной – пользователям приходилось выбирать режим: или изучать приложение, или пользоваться приложением. Нужно ли говорить, что подобный подход к организации обучающих режимов далек от оптимальности? Опытные пользователи, разумеется, обычно отключали всплывающую справку. Затем, чтобы воспользоваться незнакомыми функциями приложения, им приходилось открывать меню Help (Справка), включать всплывающую справку, указывать курсором на незнакомый объект, читать справку, возвращаться в меню, выключать всплывающую справку... Сплошная головная боль!

В итоге всплывающая справка так и не прижилась, и разработчики стали описывать только наиболее очевидные и наиболее распространенные функции, что окончательно поставило под сомнение ее полезность. В Mac OS X всплывающую справку похоронили, отдав предпочтение механизму всплывающих подсказок, сходному с тем, который получил широкое распространение в продуктах Microsoft.

Всплывающие подсказки

Не славившаяся особой изобретательностью в области пользовательских интерфейсов, Microsoft создала разновидность всплывающей справки, которая получила название всплывающей подсказки (tooltip) и стала одной из самых умных и наиболее эффективных идиом пользовательского интерфейса, какие мы когда-либо встречали (рис. 23.2). Поначалу может показаться, что всплывающие подсказки похожи на всплывающую справку, однако при близком рассмотрении можно отметить некоторые отличия в форме и поведении, которые с точки зрения пользователя радикально меняют дело.

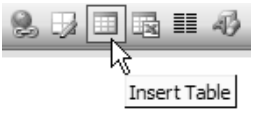


Рис. 23.2. *Эта всплывающая подсказка в Microsoft Word 2003 помогает пользователям, забывшим смысл пиктограммы. При этом нет необходимости тратить экранное пространство на подпись*

Во-первых, всплывающие подсказки имеют рассчитанную задержку и отображают полезную информацию только после того, как пользователь на секунду или около того задержит курсор над объектом. Этого времени пользователю хватает, чтобы указать на функцию и вызвать ее, не получив всплывающую подсказку. Пауза гарантирует, что подсказки не будут беспокоить пользователя, когда он перемещает курсор по экрану во время работы, и что, если пользователь забудет назначение редко используемой кнопки-значка, он сможет получить всю нужную информацию за полсекунды.

Всплывающие подсказки часто содержат всего одно слово или очень короткую фразу, описывающую функцию. Как правило, они не включают в себя изложение принципов использования объектов – предполагается, что остальные сведения пользователь получит из контекста. Это ярко демонстрирует разницу в намерениях проектирования Microsoft и Apple: Apple хотела, чтобы ее всплывающие справочные сообщения *обучали* новичков; в Microsoft же исходили из предположения, что начинающие пользователи будут обучаться работе с приложением иным способом, а всплывающие подсказки должны представлять собой просто краткие напоминания для опытных пользователей.

Интеллектуальные всплывающие подсказки Microsoft Office 2007 интегрировали в себя содержание справочной системы вполне в духе всплывающей справки. И хотя еще неясно, что получится из этой затеи, причин для провала нет, если только эта функция не станет мешать работе опытных пользователей. Качественная интеграция всплывающих подсказок, которым присуща контекстная чувствительность, с иными механизмами справки может лишь снизить налоги, связанные с освоением приложения.

Всплывающие подсказки делают элементы управления на панели инструментов гораздо более доступными для середняков, что и позволило панелям инструментов эволюционировать за рамки альтернативы командам меню. В результате панели инструментов стали основной идиомой выполнения команд в монопольных приложениях. Меню отошло на второй план как средство доступа к командам для новичков и инструмент для выполнения нечасто требующихся или сложных функций. Естественный порядок кнопок-значков как основная идиома, меню как дублер – такой подход значительно упрощает работу в монопольных приложениях. Microsoft Office 2007 развивает эту тему, заменяя все меню лентой, совмещающей наглядность и текстовую выразительность. Мы обсудим ленту далее в этой главе.

Предусматривайте всплывающие подсказки для всех элементов управления, расположенных на инструментальной панели или представленных пиктограммами.

Блокировка элементов управления на панели инструментов

Если назначение того или иного элемента управления, расположенного на инструментальной панели, теряет смысл в текущей ситуации, его необходимо заблокировать. Заблокированные кнопки не должны демонстрировать реакцию на действия, не должны нажиматься. Их следует окрашивать в серые тона, чтобы исключить возможное непонимание со стороны пользователя.

В некоторых приложениях заблокированные элементы управления вообще исчезают с панели инструментов, однако подобное поведение может иметь нежелательные эффекты. Пользователям свойственно запоминать размещение элементов управления. Если же кнопки-значки исчезают, надежная панель инструментов становится непредсказуемой и коварной идиомой, которая до обморока пугает новых пользователей и способна дезориентировать даже опытных.

Эволюция панели инструментов

После того как люди начали рассматривать инструментальную панель как нечто большее, чем просто быстрый вариант меню, ее потенциал

стал более очевидным. Проектировщики пришли к пониманию, что нет никаких причин ограничивать круг элементов управления, размещаемых на панелях инструментов, кнопками-значками – кроме привычки пользователей. Вскоре проектировщики начали изобретать новые идиомы специально для инструментальной панели. С появлением этих новых конструкций инструментальная панель действительно стала одним из первоочередных механизмов управления приложением, независимым от меню и во многих случаях превосходящим его.

После кнопки-значка на панели инструментов нашлось место для комбо-списка. В качестве примера можно привести комбо-списки «Стиль», «Шрифт» и «Размер шрифта» в Microsoft Word. Эти элементы выбора разместились на панели инструментов совершенно естественным образом. Они предлагают те же самые функциональные возможности, что и раскрывающиеся меню, но, кроме того, отображают текущий стиль шрифта, его название и размер. Такая идиома выводит дополнительную информацию, а пользователи при этом затрачивают гораздо меньше усилий.

Появление комбо-списка на панели инструментов стало прецедентом, и вслед за этой идиомой появились многочисленные идиомы, которые мы уже обсуждали в главе 21. Некоторые из этих идиом панели инструментов показаны на рис. 23.1.

Индикация состояния

Такое разнообразие элементов управления внесло свой вклад в расширение области применения инструментальной панели. Изначально, когда панель только появилась, она была просто средством быстрого доступа к часто используемым *функциям*. По мере дальнейшего развития элементы управления на панели стали отображать *текущее состояние* данных, управляемых программой. Вместо обычной кнопки-значка, нажатие на которую изменяет начертание шрифта с обычного на курсивный, стала использоваться кнопка, показывающая, имеет ли выбранный в данный момент текст курсивное начертание. В итоге кнопка-значок не только управляет изменением начертания текста, но и предоставляет информацию о текущем его состоянии.

Меню на панелях инструментов

Дальнейший рост разнообразия элементов управления, размещаемых на инструментальной панели, приводит нас к парадоксальной ситуации: на панель добавляется меню. Инструментальная панель Word, показанная на рис. 23.3, демонстрирует раскрывающееся меню Отменить. Мощные и сложные идиомы, подобные этой, отодвигают старомодную строку меню все дальше и дальше на второй план, оставляя за ней роль дополнительного средства выполнения команд.

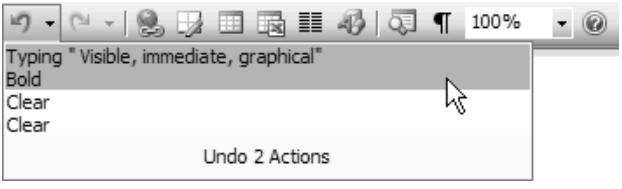


Рис. 23.3. *Теперь инструментальные панели содержат раскрывающиеся меню, такие как меню Отменить, показанное здесь. Это компактный способ предоставить доступ к мощной функции*

Подвижные панели инструментов

Компания Microsoft внесла в развитие инструментальной панели как идиомы пользовательского интерфейса значительно больший вклад, чем любой другой производитель программного обеспечения, и это находит свое отражение в уровне качества ее продуктов. Все инструментальные панели в ее офисном пакете могут настраиваться в весьма широком диапазоне. Каждая программа имеет стандартный набор инструментальных панелей, любую из них по отдельности пользователь может скрыть или сделать видимой. Если панель видима, ее можно на ходу разместить в одном из пяти положений. Панель может быть прикреплена – **запаркована** – у любой из четырех границ главного окна программы. Если оторвать панель от границы, она становится плавающей панелью, превращаясь в самостоятельное окно с заголовком (рис. 23.4).

Перемещая панели инструментов, пользователь может попасть в ситуацию, когда содержание одних панелей оказывается частично скрыто из-за наличия других панелей. В Microsoft изящно решили эту проблему с помощью комбинированной кнопки-значка и раскрывающегося меню, которое появляется только тогда, когда содержимое инструментальной панели частично скрыто, и обеспечивает доступ к скрытым элементам управления через раскрывающееся меню, как показано на рис. 23.5.

Настройка панелей инструментов

Вне всяких сомнений, в Microsoft осознают связанные с инструментальными панелями проблемы, возникающие из-за того, что на панелях размещаются часто используемые функции для всех пользователей, но для каждого конкретного пользователя этот набор функций – свой. По-видимому, в Microsoft было принято следующее решение: распространять программы с такой конфигурацией панелей инструментов, которая с наибольшей вероятностью удовлетворит типичного пользователя, и дать возможность другим пользователям настраивать панели в соответствии со своими предпочтениями. Однако ценность этого решения была несколько снижена за счет добавления в типовую

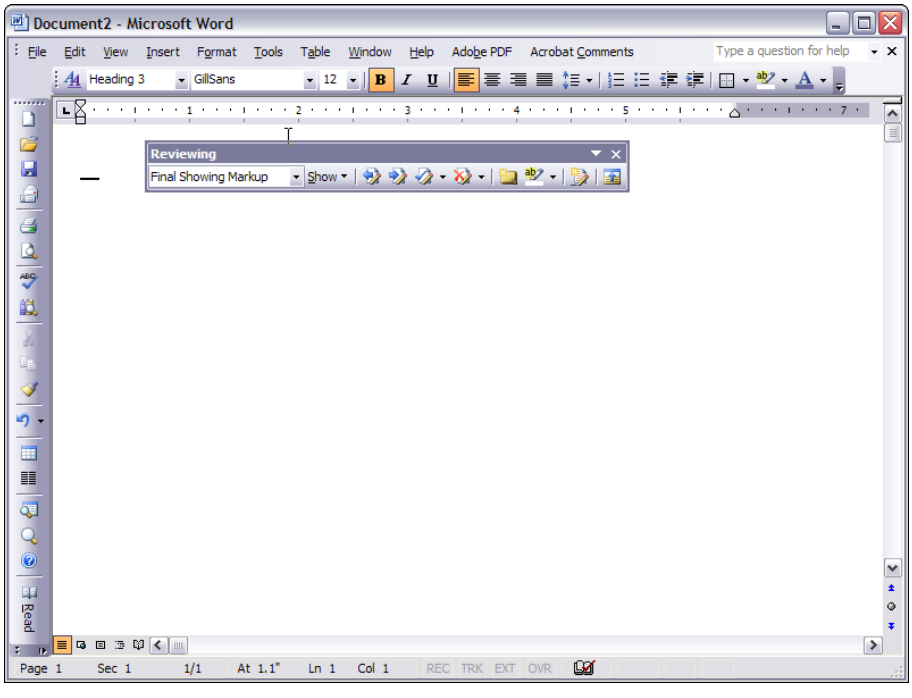


Рис. 23.4. *Панели инструментов можно парковать горизонтально (сверху), вертикально (слева), а можно отрывать от границы окна, создавая плавающие палитры инструментов*

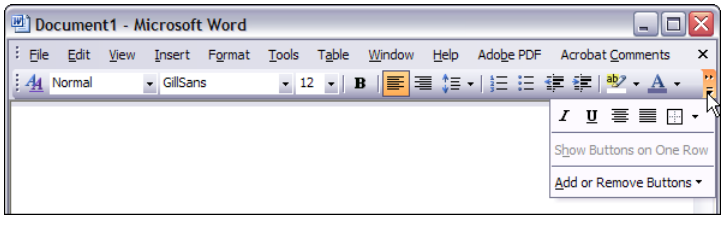


Рис. 23.5. *В Microsoft очень грамотно подошли к решению проблемы частичного перекрытия панелей инструментов: пользователь сохраняет возможность доступа ко всем скрытым элементам. Помимо этого, такой подход упрощает возможность настройки панели: опытные пользователи наверняка предпочтут выполнить настройку инструментальной панели, обратившись к пункту Настройка..., расположенному внизу раскрывающегося меню Добавить или удалить кнопки. Важно также отметить, что по умолчанию все панели запаркованы у границы окна, и пользователям не приходится беспричинно их перемещать – это было бы в чистом виде интерфейсным налогом*

конфигурацию кнопок, которые используются не так часто. Например, инструментальные панели Word по умолчанию содержат ряд кнопок, которые используются довольно редко. Среди них такие, как Автотекст или Добавить таблицу Excel. Они скорее являются фигурантами маркетингового списка особенностей продукта, нежели элементами, которые ежедневно требуются большинству пользователей. Хотя время от времени они могут понадобиться, подавляющая часть аудитории продукта использует их *достаточно редко*. Применение персонажей и сценариев помогает разобраться с подобными ситуациями (см. главы 5 и 6).

Более подготовленным пользователям Word предлагает широкие возможности по настройке панелей инструментов. С такой степенью гибкости панелей связана определенная опасность: беспечный пользователь может создать нечитаемую и непригодную для использования панель инструментов. Впрочем, чтобы что-то полностью разрушить, требуется приложить некоторые усилия. Люди вообще не склонны серьезно трудиться над созданием чего-то уродливого и сложного в обращении. Скорее, они будут вносить постепенные изменения на протяжении месяцев или даже лет. Microsoft расширила идиому таким образом, чтобы вы могли создавать собственные, совершенно новые панели инструментов. Обычным пользователям такая возможность определенно не нужна, но есть люди, которые оценят подобную гибкость.

Лента

Как уже говорилось ранее в этой главе и в главе 22, в пакете Office 2007 Microsoft представила новую идиому графического пользовательского интерфейса – ленту (рис. 23.6). По сути дела, лента является панелью с вкладками и текстовыми метками, описывающими группы функций, а также смешанным представлением кнопок-значков и текстовых команд. Вкладки обеспечивают объединение в группы, схожие с группами внутри меню (в Word 2007, к примеру, доступны группы Главная, Вставка, Разметка страницы, Ссылки, Рассылки, Рецензирование и Вид).

Несмотря на визуальное структурирование значительного количества функций, что само по себе ценно, нет веских оснований полагать, что лента является таким серьезным новшеством, как полагает сама компания Microsoft. (А если ее расположить иначе, она напоминает сред-

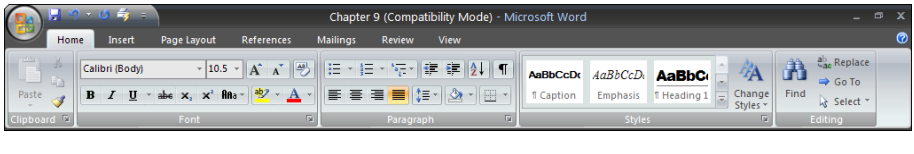


Рис. 23.6. *Лента в Microsoft Word 2007 заменяет систему меню панелью с вкладками. Она дает более наглядный доступ к функциям по сравнению с обычным меню, но недостаток текстовых подписей может ограничить ее эффективность в качестве средства обучения*

ство Inspectors, созданное Apple. К примеру, в iWeb есть палитра инструментов, содержимое которой меняется в зависимости от того, какой инструмент выбран. Это не вкладка, но ведет себя точно так же.)

Необходимо отметить, что практически полный отказ от текстовых команд, присущих традиционным меню (которые пользователям теперь приходится включать в Настройках), в пользу кнопок-значков может иметь неприятные последствия для пользователей, осваивающих новые продукты. Впрочем, на момент написания этого текста данной идиоме было всего несколько месяцев, так что оценивать ее успешность еще рано.

Контекстные панели инструментов

Действительно полезным развитием идеи панели инструментов является контекстная панель инструментов. Похожая на контекстное меню, доступное по правой кнопке мыши, такая панель отображает небольшую группу кнопок-значков рядом с курсором мыши. В некоторых реализациях представленные кнопки зависят от того, какой объект выделен. Если выделен текст, панель содержит кнопки форматирования текста; если выбран графический примитив, кнопки позволяют изменять свойства объекта. Вариант этой идиомы встроен в Microsoft Office 2007 (там он носит название «миниатюрной панели инструментов»), хотя похожие идиомы уже использовались в нескольких приложениях, включая Adobe Photoshop (где панель инструментов паркуется) и среду создания музыки Apple Logic (где панель инструментов представлена модальной курсорной палитрой).

24

Диалоговые окна

Как говорилось в главе 21, отличительной чертой некачественного проектирования взаимодействия является интерфейс, состоящий преимущественно из перегруженных элементами управления модальных диалоговых окон. Очень сложно обеспечить гладкое взаимодействие, заставляя пользователей продираться сквозь лабиринты диалогов. Если пользователь – повар, а приложение – кухня, то диалоговое окно – кладовая с продуктами. Роль кладовой вторична, как и роль диалоговых окон. Диалоги – актеры второго плана, а не звезды, и хотя они могут служить развитию действия, они не могут быть его основой.

Уместное применение диалоговых окон

Диалоговые окна заслоняют собой главное окно приложения. Диалоговое окно вовлекает пользователя в общение, отображая информацию и требуя от пользователя действий. Закончив просматривать или редактировать представленную информацию, пользователь может сохранить изменения или отказаться от них. Затем окно исчезает – и пользователь возвращается к главному окну приложения.

К сожалению, многие пользователи и программисты привыкли считать диалоговые окна основной идиомой графического пользовательского интерфейса (в основном из-за того, что диалоговые окна очень легко создавать). Многие приложения используют диалоговые окна как основной метод взаимодействия с программой (здесь мы не имеем в виду простые приложения, состоящие из одного диалогового окна, ведь в таких случаях диалоговое окно просто становится основным). В большинстве приложений пользователям приходится многократно переключаться между главным окном и диалоговыми окнами, что неизбежно утомляет и раздражает их.

ПРИНЦИП проектирования

Реализуйте основное взаимодействие в основном окне.

Когда приложение открывает диалоговое окно, оно временно прерывает основной рабочий процесс, рассеивая фокус внимания пользователя ради решения второстепенных задач. Если вы попросите приглашенных на ужин гостей временно оторваться от супа и проследовать в кладовку, течение беседы будет прервано, и этого определенно лучше избегать, если нет чертовски хорошей причины тащить их туда. Точно так же диалоговое окно нарушает гармоничное взаимодействие между пользователем и программой. Диалоговые окна, каково бы ни было их назначение, прерывают взаимодействие и заставляют пользователей реагировать на программу, а не управлять ею.

Иногда бывает полезно прервать поток пользователя и заставить его сосредоточить внимание на том или ином взаимодействии. Диалоговые окна хорошо подходят для выполнения обособленных функций: все запутанное, опасное и редко используемое можно без проблем поместить в отдельное окно. Это вдвойне полезно для преобразующих функций, которые вносят значительные и заметные изменения в состояние приложения. Такие изменения внешнего вида приложения следует отделять неким барьером от пользователей, не знакомых с ними. К примеру, функция, позволяющая полностью переформатировать документ, должна считаться преобразующим действием, изменяющим внешний вид. Диалоговое окно предотвращает случайное выполнение этой функции, выставляя большую и дружелюбную кнопку Отмена на видное место, а также обеспечивая пространство для вывода пояснений и предупреждений, связанных с применением опасных элементов управления. Диалоговое окно способно наглядно показать пользователю потенциальные эффекты выполнения функции посредством специальной миниатюры, демонстрирующей результат (и, разумеется, для подобных преобразований следует предоставлять надежную функцию отмены действия).

ПРИНЦИП проектирования

Применение диалоговых окон оправдано для функций, не входящих в основной поток взаимодействия.

Диалоговые окна хорошо подходят также для представления нечасто применяемых функций и настроек. Они изолируют подобные операции от других, необходимость в которых возникает чаще. Диалоговое окно обычно обеспечивает более широкий простор для элементов управления, чем другие варианты. В частности, здесь больше пространства для поясняющего текста, чем на панели инструментов.

Диалоговые окна хороши также для сведения воедино информации по одной теме, к примеру, свойств объекта предметной области – счета

или клиента. Они способны собирать информацию, связанную с определенной функцией, выполняемой программой, например с печатью отчетов. Очевидно, что пользователям удобно, когда вся информация и элементы управления, относящиеся к определенному предмету, собраны в одном месте, – им не приходится искать нужную информацию по всему интерфейсу, и это приводит к сокращению налогов, связанных с навигацией.

Диалоговые окна подходят для объединения элементов управления и информации, связанных с одним объектом предметной области или с одной функцией приложения.

Как и меню, диалоговые окна могут становиться хорошим средством изучения команд для новых пользователей. Поскольку диалоговые окна можно делать более многословными и структурированными, они способны служить альтернативным обучающим вариантом доступа к функциям, напрямую доступным в основном окне приложения через непосредственное манипулирование. Пример такого подхода можно найти в Microsoft Word. Пользователь, хорошо знакомый с идиомами этого приложения, умеет определять отступы, манипулируя небольшими бегунками на линейке документа. Эту идиому не так уж легко обнаружить, поэтому проектировщики Microsoft предусмотрели еще и команду Табуляция в меню Формат. Открывающееся по этой команде диалоговое окно предоставляет пользователям больше информации (однако следует отметить, что оно, к сожалению, не обучает пользователей применять идиому линейки).

Диалоговые окна служат двум хозяевам – постоянным пользователям, которые знакомы с программой и посредством таких окон получают доступ к более сложным и более опасным возможностям, и эпизодическим пользователям, которые не знакомы с возможностями и способами применения программы и используют диалоговые окна для изучения основ. Двойственная природа диалоговых окон означает, что они должны быть компактными и мощными, быстрыми и простыми и притом понятными и очевидными. Может показаться, что эти две цели вступают в противоречие, однако в действительности они способны дополнять друг друга. Скорость работы и мощные функции диалогового окна могут работать и на очевидность.

Основы применения диалоговых окон

В большинстве диалоговых окон содержатся поясняющий текст, интерактивные элементы управления и связанные с элементами управления текстовые метки. Существуют определенные базовые соглашения, однако разнообразие применений идиомы диалогового окна приводит к тому, что простых и жестких правил нет. Важно, чтобы в основу создания диалоговых окон были положены устоявшиеся подходы к по-

строению графических интерфейсов и уместное применение управляющих элементов. В частности, диалоговое окно должно демонстрировать явную визуальную иерархию, группировку, основанную на схожести элементов, и компоновку, построенную исходя из принятого порядка чтения (слева направо и сверху вниз для западной культуры). Более подробно практика визуального проектирования интерфейсов описывается в главе 14. Подробнее о корректном применении стандартных элементов управления мы рассказывали в главе 21.

Когда диалоговое окно открывается, оно должно отображаться поверх всех прочих графических объектов приложения, чтобы его появление не осталось незамеченным. При последующем взаимодействии с пользователем диалоговое окно может быть перекрыто другим окном или приложением, однако в любой момент должно быть очевидно, как вернуть его на первый план.

Любое диалоговое окно должно иметь заголовок, четко описывающий его назначение. Если окно является функциональным, заголовок должен сообщать о *действии*, то есть, скорее всего, содержать глагол. К примеру, если мы выполнили команду Вставка→Разрыв из меню Word, в заголовке открывшегося окна должно быть написано «Вставить разрыв». Что мы делаем? Мы *вставляем разрыв*! Мы ничего не рвем, так что «Разрыв» – неподходящий заголовок для этого окна. Подобное слово может запросто отпугнуть или запутать человека.

Используйте глаголы в заголовках функциональных диалоговых окон.

Если диалоговое окно используется для задания свойств объекта, заголовок должен содержать название или описание этого объекта. Диалоговое окно свойств в Windows работает именно таким образом. Когда мы запрашиваем окно «Свойства» для каталога «Backup», в заголовке написано: «Свойства: Backup». Точно так же, если диалоговое окно работает с выделением, может быть полезно вывести в заголовке усеченное выделение, чтобы помочь пользователю ориентироваться.

Используйте названия объектов в заголовках диалоговых окон свойств.

В большинстве стандартных диалоговых окон присутствует по меньшей мере одна **терминальная команда** – элемент управления, активация которого приводит к закрытию окна. В большинстве модальных диалогов есть две кнопки терминальных команд: OK и Отмена, хотя значок «Закрыть» в правом верхнем углу окна также является идиомой терминальной команды.

Существует техническая возможность создавать диалоговые окна, не имеющие терминальных команд. Некоторые окна в одностороннем по-

рядке создаются и уничтожаются программой, например, когда требуется информировать пользователя о прогрессе функции, выполнение которой требует долгого времени. Поэтому проектировщики часто не заботятся о терминальных командах. Как мы увидим далее, по многим причинам это нельзя считать качественным проектированием.

Модальные диалоговые окна

Существует два типа диалоговых окон: модальные и немодальные. Наиболее распространенная разновидность диалоговых окон – **модальные диалоговые окна**. После открытия такого диалогового окна создавшая его программа приостанавливает всю работу до тех пор, пока оно не будет закрыто. Попытка выбрать любое другое окно, принадлежащее программе, отвергается, о чем пользователя извещают грубым звуковым сигналом. На время работы модального диалогового окна становятся недоступными все элементы управления и объекты, расположенные в основном окне приложения. Разумеется, пока модальное диалоговое окно находится на экране, пользователь имеет возможность активизировать и запускать *другие* программы, но при этом само диалоговое окно может оставаться на экране неопределенно долго. Когда пользователь вернется к программе, модальное диалоговое окно по-прежнему будет ждать его действий.

Модальные диалоговые окна являются самыми понятными для пользователей (а равно и для проектировщиков). Порядок работы с модальным диалоговым окном весьма прост, окно как бы говорит пользователю: «Прервитесь на время и займитесь мною. Когда наше общение будет закончено, вы сможете продолжить свою работу». Жестко заданное поведение модального диалогового окна означает, что, хотя оно и может быть использовано неправильно, понято оно будет почти всегда верно. Программа может иметь чересчур большое число модальных диалоговых окон, они могут быть недостаточно выразительными или даже глупыми, но их назначение и возможности, как правило, достаточно понятны пользователю.

Действие некоторых модальных диалоговых окон распространяется на все приложение или на весь активный документ. Другие действуют только на текущее выделение, но пользователь в этом случае не может изменить выделение после открытия диалогового окна. Это наиболее важное различие между модальными и немодальными диалоговыми окнами.

Поскольку модальные диалоговые окна фактически останавливают работу приложения, из которого они вызваны, более точно их можно назвать **модальными для приложения**. Кроме того, существует возможность создать такое диалоговое окно, которое называется **модальным для системы**. Вызов такого диалогового окна приводит к приостановке всех приложений в системе. Прикладные программы никогда не

должны создавать подобные окна. Их единственное предназначение состоит в том, чтобы сообщить пользователю о наступлении катастрофических событий (таких как разрушение жесткого диска), затрагивающих целиком всю систему или процессы реального мира.

Немодальные диалоговые окна

Другая разновидность диалогового окна называется **немодальным диалоговым окном**. Такие окна используются значительно реже, чем их модальные собратья.

После открытия немодального диалогового окна создавшая его программа продолжает свою работу. Немодальное диалоговое окно не прерывает работу и не «замораживает» приложение. Разнообразные элементы управления, меню и панели инструментов основной программы остаются доступными и функциональными. Немодальные диалоговые окна также включают в себя терминальные команды, хотя соглашения по оформлению таких окон являются более расплывчатыми и запутанными, чем для модальных вариантов.

Немодальное диалоговое окно – создание намного более сложное в применении и трудное для понимания, главным образом потому, что его предназначение не всегда очевидно. После открытия немодального диалогового окна вы можете вернуться к работе с основной программой, в то время как само диалоговое окно остается открытым. Это означает, что вы можете изменять выделение, не закрывая немодальное окно. Если оно воздействует на текущий выбранный объект, вы можете выбрать объект, изменить его, выбрать другой объект, изменить его, выбрать третий объект и изменить все, что пожелаете. Примером может служить диалоговое окно «Найти и заменить» в Microsoft Word, которое позволяет отыскать некоторое слово в тексте (это слово автоматически будет выделено), изменить его и опять вернуться к окну, остающемуся открытым в процессе редактирования.

В некоторых случаях можно также перетаскивать объекты между основным окном и немодальным диалоговым окном. Эта особенность делает их действительно эффективным средством для создания палитр инструментов или палитр объектов в программах рисования.

Проблемы немодальных диалоговых окон

Большинство немодальных диалоговых окон имеют неуклюжую реализацию. Их поведение противоречиво и неочевидно. Визуально они очень похожи на модальные диалоговые окна, но функционально сильно отличаются. Существует несколько общепринятых соглашений, касающихся их поведения, особенно в части терминальных команд.

Чаще всего путаница возникает потому, что с поведением модальных диалоговых окон пользователи знакомы очень хорошо. Модальное окно

может отреагировать на текущее выделение в момент вызова. При этом диалоговое окно точно знает, что выделение не изменится за время его пребывания на экране. Для немодального окна, наоборот, вероятность того, что выделение будет изменено за время пребывания окна на экране, весьма высока. Что же диалоговому окну делать в этом случае? Как, например, должно вести себя немодальное диалоговое окно, предназначенное для работы с текстом, если мы выделим в основном окне приложения нетекстовый объект? Должны ли элементы диалогового окна стать недоступными? измениться? исчезнуть? Вопросы вроде этого требуют изощренных приемов проектирования, а также внимательного изучения потребностей, целей, ментальных моделей персонажа. Как следствие, немодальные диалоговые окна могут оказаться гораздо более сложными для проектирования и реализации, чем модальные, которые избегают описанных проблем, временно замораживая состояние приложения.

Более совершенные немодальные диалоговые окна: два решения

Для улучшения немодальных диалоговых окон мы предлагаем две стратегии проектирования. Первую проще принять, это паллиативное лекарство для распространенных заболеваний. Вторая более радикальна и предлагает скачок в эволюции. Как вы можете догадаться, мы предпочитаем эволюцию. (По счастью, как вы увидите ниже, в последние годы многие другие проектировщики тоже начали движение в этом направлении.)

Паллиативный подход

Если время и ресурсы, отведенные на решение проблем проектирования взаимодействия, ограничены, мы рекомендуем оставить немодальные диалоговые окна примерно в том же состоянии, в каком они находятся сейчас, но взять на вооружение два руководящих принципа и последовательно применять их ко всем немодальным диалоговым окнам.

Визуально выделяйте различия между модальными и немодальными диалоговыми окнами.

Если для создания немодального диалогового окна программист использует стандартные средства интерфейса прикладного программирования (API) Windows, в результате получится окно, которое визуально не отличается от модального. Мы должны сломать эту привычку. Проектировщик обязан обеспечить четкие визуальные отличия немодальных диалоговых окон – окрасить фон в другой цвет, сделать визуально отличными элементы управления или использовать другой цвет заголовка либо пиктограммы. Неважно, какой метод вы выберете, – важно, чтобы вы неукоснительно его придерживались.

Второй принцип гласит, что мы должны установить правильные и непротиворечивые соглашения по оформлению терминальных команд. Порой возникает ощущение, что каждый производитель, а то и каждый программист использует свою собственную методику для каждого диалогового окна. Эту какофонию подходов трудно оправдать. Некоторые окна предлагают кнопку Закрыть, другие – Применить, третьи – Готово, Отклонить, Принять, Да и даже OK. Разнообразие вариантов бесконечно. Существуют и такие, которые вообще обходятся без подобных кнопок и полагаются только на кнопку закрытия, расположенную в правом верхнем углу окна. Завершение немодального диалогового окна должно быть простой, легкой в освоении, непротиворечивой, пусть не везде одинаковой, но по меньшей мере узнаваемой в большинстве программ идиомой.

Используйте единообразные терминальные команды внутри немодальных диалоговых окон.

Одна из самых больших неприятностей заключается в изменении надписей на терминальных кнопках, например изменение надписи Отмена на Применить или Закрыть в зависимости от того, предпринимал ли пользователь какие-либо действия в немодальном диалоговом окне. Такое динамическое изменение надписей в лучшем случае дезориентирует и трудно поддается интерпретации, в худшем – пугает своей непонятностью. Эти надписи *никогда* не должны изменяться. Если пользователь не выбрал ничего внутри диалогового окна, но все же нажимает на кнопку OK, диалоговое окно должно считать, что пользователь велел «закрыть окно и не предпринимать никаких действий», поскольку по сути именно это пользователь и сделал. Модальные диалоговые окна предоставляют нам возможность явным образом отменить произведенные действия – с помощью кнопки Отмена. В немодальных диалоговых окнах обычно не удается использовать эту прямую идиому и приходится прибегать к функции отмены – так что изменение подписей с целью предупредить пользователей еще больше запутывает ситуацию.

Не допускайте динамического изменения надписей на терминальных кнопках.

Когнитивная сила *модальных* диалоговых окон состоит в жесткой однозначности кнопок OK и Отмена. В них кнопка OK означает: «Принять мой ввод и закрыть окно». Проблема в том, что для немодальных диалоговых окон не существует эквивалентного действия. Элементы управления в немодальном диалоговом окне постоянно активны, и эквивалентное понятие очень сложно сформулировать. Пользователь не связывает произведенные изменения с нажатием на кнопку Выполнить,

как он делает это в случае модального диалогового окна. В модальных диалоговых окнах кнопка Отмена означает: «Забыть о вводе и закрыть окно». Но поскольку изменения в немодальном диалоговом окне вступают в силу немедленно после нажатия соответствующей кнопки, не может быть и речи о таком понятии, как «Отменить все мои действия». В процессе работы с диалоговым окном может быть произведено значительное количество изменений в целом ряде объектов. Надлежащая идиома заключается в реализации функции отката изменений, которая может постоянно находиться на панели инструментов или в меню Правка и быть активной для всех немодальных диалоговых окон в приложении. Это вполне логично, ведь функция отмены недоступна, когда открыто модальное диалоговое окно, но доступна, когда открыто немодальное.

Единственная непротиворечивая терминальная команда для немодальных диалоговых окон – кнопка Закрыть. Любое немодальное диалоговое окно должно иметь кнопку Закрыть на одном и том же месте – например в правом нижнем углу. Это правило должно неукоснительно соблюдаться от окна к окну: кнопка должна находиться на одном и том же месте и иметь одну и ту же надпись. Эту кнопку нельзя деактивировать. Более того, если кнопка Закрыть приводит к выполнению какой-либо функции в дополнение к закрытию окна, вы по сути создали модальное диалоговое окно, которое должно следовать правилам построения модальных диалоговых окон.

В немодальном диалоговом окне часто присутствует ряд кнопок для немедленного вызова различных функций. Диалоговое окно не должно закрываться по нажатию на какую-либо из таких командных кнопок. Немодальное диалоговое окно потому так и называется, что всегда остается доступным для повторного использования и должно закрываться лишь в тот момент, когда будет нажата специальная, размещенная в определенном месте кнопка Закрыть.

Кроме всего прочего немодальные диалоговые окна должны очень экономно расходовать экранное пространство. Они будут оставаться на экране продолжительное время, находясь на переднем плане, и потому должны быть тщательно продуманы, чтобы не тратить пространство впустую на что-нибудь ненужное. По этой причине – особенно в случае плавающих палитр инструментов – лучшим решением будет оставить единственный элемент управления, отвечающий за закрытие окна, – кнопку в правом верхнем углу окна.

Эволюционный подход

Приведенные выше рекомендации дают нам весьма ограниченное решение. Оно, конечно, улучшает неуклюжее взаимодействие, но результаты по-прежнему оставляют желать лучшего. Существует более радикальное решение, которое избавит нас от болезней немодальных диалоговых окон.

С давних пор в обращении находятся лишь две идиомы для немодальных взаимодействий. Немодальные диалоговые окна появились первыми. Вторая немодальная идиома в области интерфейсов – панель инструментов – возникла не так давно, но ее применению обычно сопутствует успех. Идиома панели инструментов доказала свою эффективность и удобство. Что гораздо важнее – пользователи, похоже, понимают, что состояние панели инструментов отражает текущее выделение и что взаимодействие с элементами панели инструментов сразу и непосредственно влияет на выделенные объекты или на все приложение.

Здесь мы усматриваем возможность по-новому подойти к вопросу немодальной интерактивности. Панели инструментов, хотя и являются немодальными, не становятся таким источником головоломок, как немодальные диалоговые окна. Кроме того, они обладают двумя характерными чертами, отсутствующими у немодальных диалоговых окон. Во-первых, они визуально отличаются от диалоговых окон, а во-вторых, нет нужды беспокоиться по поводу их закрытия, поскольку они сохраняются на экране на протяжении всего времени работы программы, и потому им не нужны терминальные кнопки. Панели инструментов справляются также и с другими проблемами. Они невероятно эффективны в смысле занимаемого экранного пространства, особенно в сравнении с диалоговыми окнами, *и они не загораживают рабочую область*!

Немодальные диалоговые окна – это обычно плавающие окна, которые разрешают пользователю свободно перемещать их по экрану, но облагают его налогом, связанным с управлением окнами. Перемещение окон по экрану не является целью пользователя, поэтому вся связанная с этим работа – в чистом виде налог. Существенным продвижением к решению этой проблемы стали **паркуемые панели инструментов** (обсуждавшиеся в главе 23). Запаркованную панель можно захватить мышью, оттащить от границы окна – и она тут же превратится в **плавающую палитру** (известную также как *окно палитры*). Вы можете оставить ее в таком виде или перетащить к любой другой границе основного окна приложения, где она опять превратится в панель инструментов, **припарковавшись** к краю окна.

Палитра расширяет идею паркуемой панели инструментов дополнительными взаимодействиями. Более того, современные панели инструментов обычно содержат все приспособления, необходимые для создания высокоинтерактивной палитры. Маленький шажок от панели инструментов к палитре связан в основном с изменением композиции и распределением вертикального пространства.

Палитры широко применяются в графических приложениях, где немодальный доступ к инструментам абсолютно необходим пользователям для поддержания продуктивного состояния потока. Adobe Photoshop одной среди первых представила большой объем функций в виде немодальных палитр.

По мере того как число палитр растет, возникает потребность создать для них стандартный ареал обитания. Adobe Fireworks MX и прочие приложения, изначально созданные компанией Macromedia, одними из первых реализовали более удачную парковочную инфраструктуру, минимизирующую интерфейсные налоги на управление объектами на экране (рис. 24.1). Последние версии Photoshop подхватили данную идиому.

Последним шагом в эволюции этой новой немодальной командной идиомы стало появление **панели задач** (task bar), или **боковой панели** (sidebar), то есть панели окна приложения, предназначенной для организации доступа к функциям, которые раньше были доступны посредством диалоговых окон. Созданное Autodesk приложение 3ds Max стало одним из первых потребителей этой идиомы, позволив регулировать параметры объектов немодально – через боковую панель. Среди

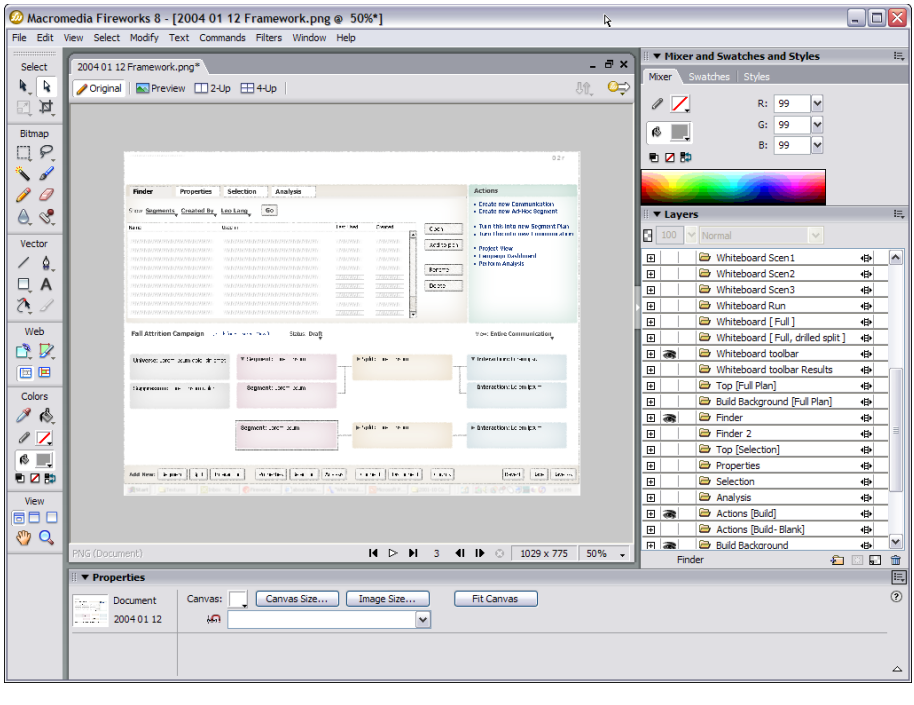


Рис. 24.1. *Припаркованные палитры в Adobe Fireworks MX реализуют взаимодействия, присущие обычным немодальным диалоговым окнам, однако не требуют от пользователей усилий и внимания, связанных с открыванием, перемещением и закрытием этих окон. Не требуется обладать развитым воображением, чтобы понять, что эти палитры очень схожи с панелями инструментов в том смысле, что используют стандартные элементы управления и приспособления для обеспечения прямого, наглядного, сквозного доступа к функциональности приложения*

популярных приложений, применяющих боковые панели, – Проводник Microsoft Windows и Internet Explorer, где применяются так называемые Explorer Bars, броузер Mozilla Firefox (Slide Bar), приложения iLife от Apple (Inspectors), а также Microsoft Office (Task Pane). Пожалуй, наиболее искренне приняла идиому боковых панелей Adobe Lightroom – практически вся функциональность этого приложения доступна немодально через боковые панели (рис. 24.2).

Боковые панели в качестве идиомы обладают большим потенциалом – они фигурируют во многих решениях компании Cooper, и их применение отнюдь не ограничивается боковыми сторонами экрана. Часто применяемый шаблон – специальная область свойств под панелью документа или рабочей областью. Она дает немодальный доступ к функциям изменения свойств выбранного объекта, минимизируя путаницу и интерфейсные налоги на управление экраном (рис. 24.3). Эту идио-

Рис. 24.2. *Боковые панели в Adobe Lightroom заменяют десятки диалоговых окон. Этот подход похож на вариант с палитрами, представленный на рис. 24.1, но, в отличие от палитр, боковая панель не требует, чтобы пользователи располагали ее на экране, и не позволяет пользователю откреплять себя или закрывать (хотя можно скрыть всю боковую панель). Это еще больше сокращает интерфейсные налоги на управление объектами и представляет собой значительный шаг вперед относительно использования диалоговых окон для обеспечения доступа к функциям приложения*

му можно с пользой применять и для функций, требующих подтверждения. Среди прочего мы применяли встроенные диалоговые окна в качестве средства создания правил для финансовых рынков.

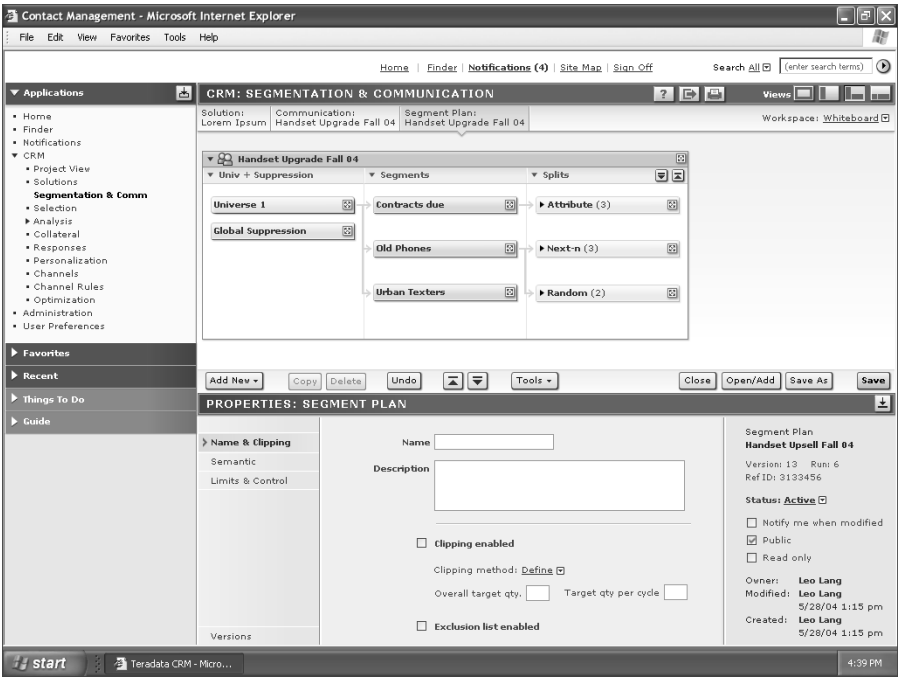


Рис. 24.3. *Это решение создано компанией Cooper для системы управления отношениями с клиентами (CRM – customer relationship management). Когда пользователь выбирает объект в рабочей области (верхняя часть экрана, слева), свойства объекта отображаются внизу. Тем самым пользователь сохраняет контекст и избавлен от интерфейсных налогов, связанных с управлением экраном*

Четыре назначения диалоговых окон

Источником самих понятий модальных и немодальных диалоговых окон является терминология программистов. Эти понятия оказывают влияние на проектирование, однако нам следует рассматривать диалоговые окна еще и с точки зрения целей пользователей. В этой связи выделяются четыре фундаментальных разновидности информации, которую полезно передавать посредством диалоговых окон: свойства, функции, процессы и сообщения.

Диалоговые окна свойств

Диалоговое окно свойств позволяет пользователям просматривать и изменять свойства или атрибуты выбранного объекта. Иногда атрибуты относятся к приложению или к документу в целом, а не к отдельному объекту.

Хорошим примером может послужить окно Шрифт редактора Word, представленное на рис. 24.4. Пользователь выделяет текст в главном окне, а затем открывает диалоговое окно посредством меню Формат. Окно позволяет изменять шрифтовые свойства выбранных символов. Диалоговые окна свойств можно считать панелями управления, на которых представлены элементы управления выделенным объектом. Диалоговые окна свойств бывают модальными и немодальными. Такое окно обычно работает с выделенным объектом, следуя структуре «объект – глагол»: пользователь выбирает объект, а затем посредством окна свойств задает настройки для выбранного объекта.

Рис. 24.4. *Диалоговое окно Шрифт в Microsoft Word 2003 – классический пример диалогового окна свойств. Пользователь выделяет текст в документе, открывает окно посредством команды меню – и возникает модальное окно с вкладками, содержащее всевозможные настройки шрифта для выделенного текста. Это одна из наиболее распространенных операций в текстовом процессоре. Почему мы вынуждены ходить за ней в другую комнату?*

Функциональные диалоговые окна

Функциональные диалоговые окна обычно вызываются из меню. Чаще всего это модальные диалоговые окна, которые управляют выпол-

нением единственной функции, такой как печать документа, изменение большого числа записей базы данных, вставка объектов или проверка правописания.

Функциональные диалоговые окна дают пользователю возможность не только запускать некоторое действие, но и влиять на характеристики этого действия. Во многих программах, когда пользователь запрашивает печать документа, используется окно «Печать», с помощью которого можно определить перечень печатаемых страниц, количество копий, указать принтер, на котором следует произвести печать документа, и выполнить другие настройки, имеющие непосредственное отношение к функции печати. Кнопка OK, завершающая работу с окном, не только подтверждает произведенные настройки и закрывает окно, но и запускает операцию печати.

Этот подход, получивший широкое распространение, объединяет в себе две функции – функцию настройки и функцию вызова операции. Но одно лишь то, что функцию *можно* настраивать, еще не означает, что пользователь *пожелает* ее настраивать перед каждым вызовом. Поэтому обычно эти функции лучше разделять (хотя, понятное дело, они должны оставаться тесно связанными).

Многие функции современных приложений являются гибкими и имеют массу параметров настройки. Если не разделять настройку и активизацию, пользователям, скорее всего, придется сталкиваться со сложными механизмами даже в тех случаях, когда они просто хотят выполнить рутинную операцию максимально легким способом.

Диалоговые окна процессов

Диалоговые окна процессов запускаются не по запросу пользователя, а самой программой. Они сообщают пользователю о том, что программа занята исполнением некоторой внутренней функции и производительность в других областях, скорее всего, будет снижена.

Когда программа запускает длительную с точки зрения человека процедуру, она обязана сообщить о том, что она будет занята какое-то время, но что в остальном все в порядке. Если программа не будет выводить подобных сообщений, пользователь в лучшем случае сочтет ее поведение грубым, а в худшем – подумает, что программа дала сбой и необходимо принимать радикальные меры.

Информируйте пользователя о том, что приложение занято.

Как уже говорилось в главе 19, многие программы сообщают о своей занятости с помощью указателя мыши, который принимает форму песочных часов. В таких случаях лучше использовать более информативное решение – диалоговое окно, отображающее ход выполнения

длительной операции (еще более удачное решение мы рассмотрим далее в этой главе).

Каждое диалоговое окно процесса должно решать четыре задачи:

- уведомить пользователя о выполнении длительной операции;
- успокоить пользователя, сообщая ему, что все в порядке;
- дать примерную оценку времени, которое займет выполнение операции;
- дать пользователю возможность прервать операцию и вернуть себе контроль над программой.

Само присутствие диалогового окна процесса отвечает первому требованию – оно сообщает пользователю, что протекает некоторый процесс. Третья задача может быть решена с помощью некоторой разновидности **индикатора хода выполнения операции (progress meter)**, который будет наглядно показывать в процентном соотношении выполненный и оставшийся объем работ. Решить вторую задачу гораздо сложнее. Программа может завершиться аварийно и оставить диалоговое окно на виду, вводя пользователя в заблуждение относительно состояния процесса. Диалоговое окно процесса должно непрерывно сообщать, что все в порядке, динамически изменяя индикатор. Индикатор должен отображать ход выполнения относительно **общего времени**, которое займет операция, а не относительно общего объема работ. Пятьдесят процентов одной операции могут по времени радикально отличаться от пятидесяти процентов другой операции.

В представлении пользователя исполнение длительной по времени операции совершенно естественным образом напоминает машину, которая жужжит себе и жужжит. Статическое диалоговое окно, которое просто объявляет о том, что компьютер выполняет «чтение с диска», вполне *может свидетельствовать* о начале выполнения длительной операции, но этого недостаточно, чтобы *убедить* пользователя в том, что операция действительно выполняется. Лучше всего отображать ход выполнения операции посредством анимации в диалоговом окне. В операционной системе Windows при перемещении, удалении или копировании файлов в диалоговом окне демонстрируется небольшой мультипликационный ролик, который изображает листы бумаги, перелетающие из одной папки в другую или в корзину (рис. 24.5). Эффект просто поразителен: у пользователя возникает полное ощущение, что компьютер действительно *что-то делает*. Ощущение, что все идет как надо, обычно возникает на интуитивном уровне, а не в результате умозаключений, и пользователи – даже подготовленные – чувствуют себя гораздо увереннее.

Индикатор хода выполнения операции от Microsoft отвечает (с натяжкой) только третьему требованию, показывая время, оставшееся до окончания операции. Ход выполнения одной операции отображается в одном диалоговом окне, хотя операция может затрагивать множест-

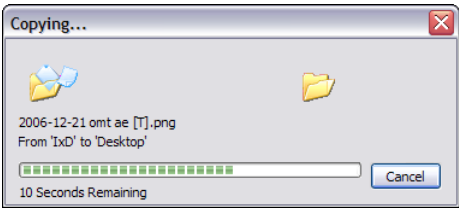


Рис. 24.5. *Здесь у Microsoft почти все получилось. В процессе выполнения любой операции перемещения, копирования или удаления файлов в Проводнике пользователь наблюдает вполне удачно спроектированное диалоговое окно с информацией о ходе выполнения операции. Оно дает некоторое представление о времени, оставшемся до завершения операции, и использует анимацию листов бумаги, летящих из левой папки в правую (или в корзину). Ментальные модели многих пользователей включают перемещение вещей внутри компьютера, а эта небольшая находка демонстрирует именно перемещение объектов. Приятно видеть в интерфейсе компьютера отражение механизмов его работы, выраженное через представления пользователя. Единственное, чего здесь не хватает, – информации о количестве оставшихся файлов; ее наличие усилило бы обратную связь с пользователем*

во файлов. В этом окне следовало бы показывать также счетчик оставшихся файлов (например, «осталось скопировать 12 файлов из 37»). На сегодня индикатор отображает лишь имя файла, который копируется в настоящий момент времени (примечательно, что в процессе установки самой Windows используется индикатор хода выполнения операции, показывающий, какое еще количество документов необходимо скопировать).

Обратите внимание, что в окне копирования (рис. 24.5) кроме всего прочего имеется кнопка Отмена. Предположительно она предназначена отвечать четвертому требованию – давать возможность отмены операции. Оценив время, которое займет операция, пользователь может решить отложить ее, что и позволяет сделать кнопка Отмена. Однако если пользователь осознал, что запустил команду по ошибке и ее следует прервать, его желание будет подразумевать не только прекращение операции, но и откат всех изменений, произведенных в ходе ее выполнения.

Если пользователь перетащил 25 файлов из каталога Alpha в каталог Bravo и в середине выполнения операции вдруг осознал, что на самом деле хотел переместить их в каталог Charlie, он нажмет на кнопку Отмена. К сожалению, в результате операция будет просто *прервана* – система посчитает это отказом от перемещения оставшихся файлов. Другими словами, если пользователь нажмет на кнопку Отмена после того, как 10 файлов уже перемещены, в каталоге Alpha останутся последние 15 файлов, а первые 10 окажутся в каталоге Bravo. Однако это не то, чего хочет пользователь. Кнопка, которая называется «Отме-

на», должна выполнять *отмену*. На языке пользователя это означает: «Я не желаю, чтобы что-то изменилось». Чтобы кнопка соответствовала выполняемому ею действию, ей следовало бы называться «Остановить копирование» или «Остановить перемещение». Но вместо этого она называется «Отмена», а потому просто обязана выполнять именно отмену. Для этого может потребоваться буферизация значительных объемов информации, и выполнение настоящей отмены запросто способно занять больше времени, чем само по себе перемещение, копирование или удаление. Но разве при этом довольно редком явлении дополнительные затраты времени не будут оправданны? В Проводнике (Windows Explorer) существует возможность полностью отменить копирование, перемещение или удаление, и нет причин, по которым кнопка Отмена не могла бы отменить выполненную часть операции.

Хорошей альтернативой стало бы наличие в окне двух кнопок – одной с надписью «Отмена», а другой – с надписью «Стоп». Тогда пользователь мог бы выбрать нужное действие.

Устранение окон с информацией о ходе выполнения операции

Поскольку диалоговое окно – это отдельная «комната», следует задаться вопросом, является ли процесс, ход выполнения которого отображается в окне, функцией, независимой от главного окна приложения. Если эта функция является составной частью объектов в главном окне, состояние ее выполнения также следует показывать в главном окне. К примеру, окно Windows с летающими страницами (см. рис. 24.5) привлекательно и уместно, однако разве копирование файлов не является фундаментальной функцией Проводника? Анимацию в этом случае можно было бы встроить прямо в главное окно Проводника. Странички могли бы летать по строке состояния или прямо по главному окну – из каталога в каталог.

Разумеется, создать диалоговое окно процесса гораздо легче, чем анимацию, встроенную в главное окно программы. К тому же в диалоговом окне удобно помещать кнопку Отмена, так что открытие окна на время выполнения длительной операции является разумным компромиссом. Но не следует упускать из виду, что, создавая окно, мы вынуждаем пользователя «ходить в другую комнату» для выполнения функции, относящейся к этой комнате. Это простое решение – но отнюдь не правильное решение. Веб-броузеры, такие как Mozilla Firefox и Microsoft Internet Explorer, предлагают гораздо более изящное решение. Поскольку загрузка веб-страниц есть интегральная часть работы броузеров, индикация хода этой операции осуществляется через строку состояния (рис. 24.6).

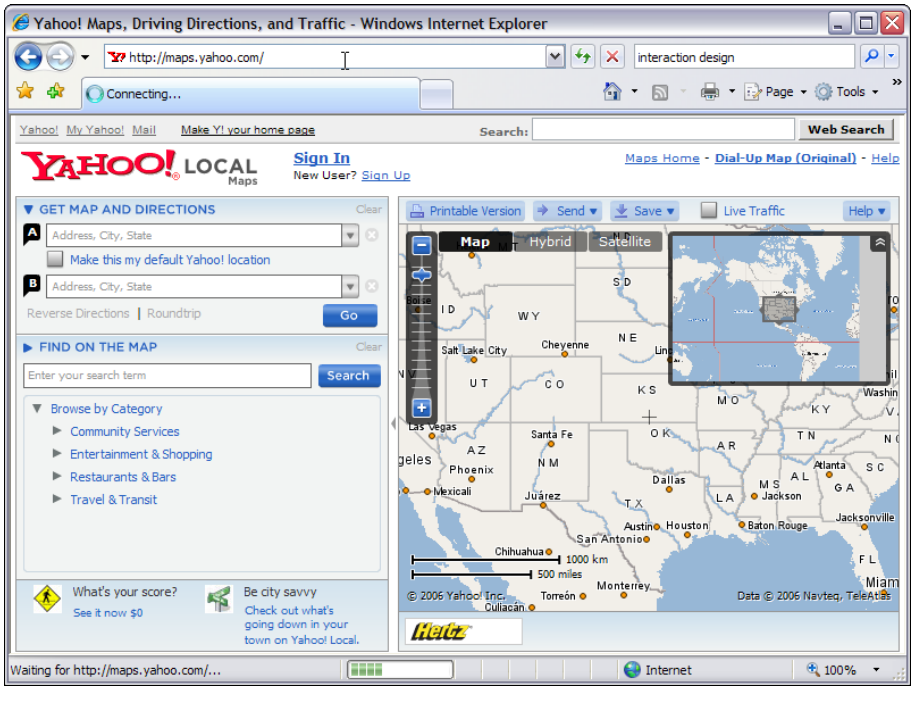


Рис. 24.6. *Веб-броузеры, такие как Internet Explorer, не открывают окно процесса всякий раз, когда загружают страницу. Вместо этого они отображают индикатор хода операции в строке состояния внизу окна. Это помогает пользователям понимать, что происходит, причем без перекрытия уже частично загрузившейся веб-страницы (которая тоже представляет интерес). Задумайтесь, уместны ли окна процессов как средство информирования пользователей вашего приложения*

Информирующие диалоговые окна

Информирующее диалоговое окно – коварный артефакт, которым, возможно, злоупотребляют больше, чем любым другим элементом графического пользовательского интерфейса. Как и окно процесса, оно запускается приложением без каких-либо просьб со стороны пользователя.

Показательный пример информирующего диалогового окна – вездесущие окна с сообщениями об ошибках. В заголовке такого окна обычно находится название программы, а в самом окне – краткое информационное сообщение, описывающее возникшую проблему. Довершают картину, как правило, пиктограмма, указывающая на класс или степень серьезности проблемы, и кнопка OK. Иногда в окно добавляется кнопка, которая позволяет вызвать интерактивную справку. Пример на рис. 24.7 получен в Microsoft Word.

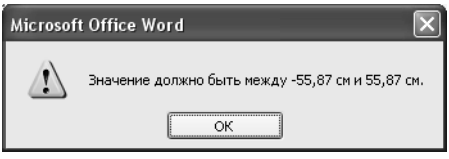


Рис. 24.7. *Это типичное информирующее диалоговое окно. Пользователь никогда не запрашивает его – программа сама принимает решение о необходимости открыть такое окно, когда она не в состоянии разрешить возникшую проблему или считает нужным похвастаться успешным завершением операции. В данном случае программа решила, что ей проще обвинить пользователя, чем попытаться самой справиться с проблемой. Пользователи же интерпретируют это сообщение следующим образом: «Размеры должны находиться в диапазоне от –55,87 до 55,87 сантиметров, а ты, фигляр и тупица, не знаешь такой простой вещи. Ты настолько глуп, что я даже пальцем не пошевелю, чтобы изменить размер за тебя!»*

Знакомое всем окно с сообщением – это обычное модальное диалоговое окно, которое приостанавливает дальнейшую работу программы до того момента, когда пользователь выполнит терминальную команду, например щелкнет по кнопке OK. Такое информирующее окно называется **блокирующим**, потому что программа не может продолжать работу, пока пользователь не отреагирует на информацию.

Кроме того, программа может в одностороннем порядке не только вывести окно, но и закрыть его. Такой тип окна называется **временным окном сообщения**, потому что программа сама закрывает окно и продолжает свою работу без вмешательства пользователя.

Временные информирующие диалоговые окна иногда применяются для вывода сообщений об ошибках. Программа, которая выводит сообщение об ошибке, чтобы известить пользователя о возникшей проблеме, может сама устранить ее или обнаружить, что ошибка была ликвидирована каким-либо другим способом. Некоторые программисты выводят окно с сообщением об ошибке или уведомительное диалоговое окно просто как предупреждение, например «Диск переполнен», и убирают его по истечении некоторого времени, например через 10 секунд. Такое поведение чревато проблемами юзабилити.

Сообщение об ошибке или сообщение, требующее подтверждения, *должно* приостанавливать программу. В противном случае пользователь может не успеть прочитать его полностью – он может в этот момент, например, отвести взгляд в сторону, отойти от компьютера или, что еще хуже, успеет лишь отметить факт появления сообщения – боковым зрением. Он будет совершенно оправданно обеспокоен тем, что, быть может, пропустил нечто важное, что обязательно напомнит о себе позже. Он начнет беспокоиться: «Что я пропустил? Были ли это важные сведения? Мне придется пожалеть, что не успел прочитать их?

Система нестабильна? Система готова дать сбой?» Это так даже в том случае, когда проблема уже решилась сама собой.

Если случившееся заслуживает того, чтобы о нем сообщить в форме диалогового окна, то определенно стоит гарантировать получение этого сообщения пользователем. Временное информирующее диалоговое окно не дает такой гарантии. Поэтому его никогда не следует использовать для вывода сообщений об ошибках или сообщений, требующих подтверждения.

ПРИНЦИП проектирования

Никогда не используйте временные диалоговые окна для вывода сообщений об ошибках или сообщений, требующих подтверждения.

Окна свойств и функциональные диалоговые окна вызываются по требованию пользователей, они служат нуждам пользователей. Однако приложение может самостоятельно выводить информирующие диалоговые окна, которые обслуживают приложение за счет пользователя. Сообщения об ошибках, уведомления и сообщения, требующие подтверждения, – это блокирующие информирующие диалоговые окна. Как станет ясно, их можно и нужно избегать в большинстве ситуаций.

Управление содержимым диалоговых окон

Даже если вы тщательно продумываете применение и организацию диалоговых окон, они легко могут переполняться свойствами, настройками и тому подобными элементами. Существует пара популярных стратегий управления содержимым, позволяющих сохранить полезность таких окон.

Диалоговые окна с вкладками

Много лет назад диалоговые окна с вкладками быстро стали стандартом в мире коммерческих программных продуктов. Эта временами полезная идиома превратилась по совместительству (и к глубокому сожалению) в удобный для программистов способ упихать много едва связанных функций в одно окно.

Она приносила и пользу: многие объекты приложений и предметных областей получили подходящие, насыщенные окна свойств, имеющие разумный размер и не перегруженные элементами управления (см. пример на рис. 24.8)

Многие функциональные диалоговые окна, ранее переполненные элементами управления, теперь более эффективно используют отведенное им экранное пространство. До появления диалоговых окон с вкладками проблема часто неуклюже решалась разворачивающимися и каскадными диалоговыми окнами, которые мы вскоре обсудим.

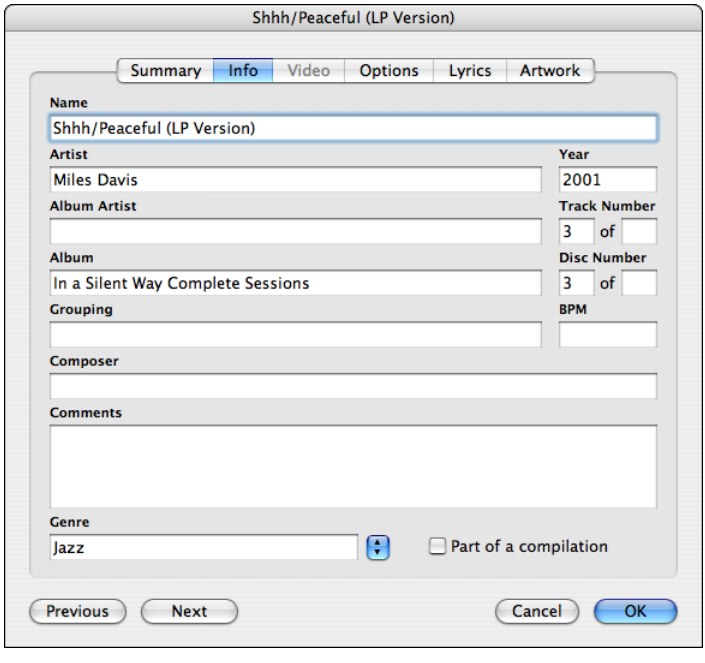


Рис. 24.8. *Это диалоговое окно с вкладками из iTunes. Сочетание различных свойств песни в одном диалоговом окне удобно для пользователей, которым не приходится искать большое количество данных в разных местах. Обратите внимание, что элементы управления, завершающие работу с окном, расположены за пределами панели с вкладками, в правом нижнем углу*

Диалоговые окна с вкладками позволяют программистам умещать большее количество элементов управления в одном окне, однако большее количество элементов управления не гарантирует, что пользователи посчитают интерфейс более простым или более мощным. Содержимое различных вкладок должно группироваться по какому-то осмысленному признаку, иначе данная возможность превращается в еще одно средство создавать продукты исходя из удобства программистов, а не потребностей пользователей.

Организация диалоговых окон с вкладками должна быть такой, чтобы позволять увеличивать глубину представления или широту охвата некой хорошо определенной темы. Чтобы увеличить широту охвата, в каждую вкладку следует помещать параллельные, альтернативные аспекты основной темы. Так, окно свойств песни в iTunes (рис. 24.8) представляет обширный набор свойств и настроек песни, которые создавали бы определенное неудобство, будучи помещенными на одной панели. Чтобы увеличить глубину, следует посвятить все вкладки глубокой проработке одного и того же аспекта темы. Распространенная вкладка Дополнительные свойства – пример такой стратегии.

Идиома вкладок удачна тем, что следует ментальной модели хранения, принятой многими пользователями: разнообразные элементы управления группируются в соседствующие панели с одним уровнем вложенности. Однако этой идиомой часто злоупотребляют.

Поместить множество элементов управления в одно диалоговое окно с вкладками очень легко, и возникает искушение добавлять в него все новые и новые вкладки. Окно Параметры в Microsoft Word (рис. 24.9) – яркий пример этой проблемы. Десять вкладок – слишком много, чтобы разместить их в одну строку, поэтому их расположили в две. Проблема с идиомой **многострочных вкладок** состоит в том, что пользователь вынужден проделывать большую работу для отыскания того единственного параметра, который требуется изменить. И хотя надписи на вкладках упрощают поиск, все равно приходится сканировать взглядом содержание нескольких вкладок, переключаясь между ними. Хуже того, когда пользователь щелкает по вкладке из строки на заднем плане, вся строка перемещается на передний план, отодвигая на задний план другие две строки. Мало кому из пользователей это нравится: человека приводит в замешательство тот факт, что вкладка, по которой щелкнули, «убегает» из-под курсора мыши.

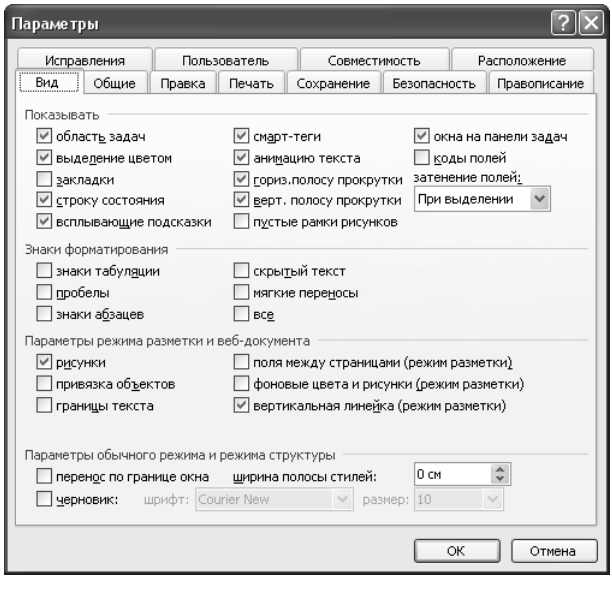


Рис. 24.9. *Окно Параметры в Word – пример злоупотребления идиомой диалоговых окон с вкладками. Проблема в том, что пользователю приходится проделывать большую работу, чтобы найти тот единственный параметр, который требуется изменить*

ПРИНЦИП проектирования

У всех идиом взаимодействия есть практические ограничения.

Многострочные вкладки иллюстрируют следующую аксиому проектирования пользовательских интерфейсов: у всех идиом, независимо от их пользы, есть практические ограничения. Группа из пяти радиокнопок может быть очень удачным решением, но группа из пятидесяти радиокнопок – это нелепо. Пять или шесть вкладок в одну строку – отлично, однако перенос вкладок на следующую строку уменьшает полезность идиомы.

Более удачным решением было бы использование нескольких окон и меньшего числа вкладок внутри каждого из них. В данном примере Параметры – слишком широкая категория, и объединение всех этих функций в одном месте не приводит ни к чему хорошему. Двенадцать панелей этого окна связаны друг с другом очень слабо, а потому нет необходимости иметь их под рукой сразу все, чтобы переключаться между ними. В таком решении, быть может, не будет элегантности, столь любимой программистами, но для пользователя оно гораздо удобнее.

ПРИНЦИП проектирования

Не создавайте многострочные вкладки.

Разворачивающиеся диалоговые окна

Пик популярности разворачивающихся диалоговых окон пришелся примерно на 1990 год; с тех пор они используются все реже и реже – в значительной степени благодаря широкому распространению панелей инструментов и окон с вкладками. Разворачивающиеся диалоговые окна до сих пор можно обнаружить в хорошо знакомых приложениях, например окно Найти и заменить в Microsoft Word.

В таких диалоговых окнах имеется кнопка с надписью Больше или Развернуть, в результате нажатия на которую диалоговое окно увеличивается в размерах, открывая доступ к дополнительным элементам управления. Открывшаяся часть диалогового окна предлагает функции, обычно предназначенные для опытных пользователей или для выполнения более сложных, но родственных операций. Окно Найти и заменить в Microsoft Word (рис. 24.10) – всем известный пример этой идиомы.

Разворачивающиеся диалоговые окна предлагают начинающим и неопытным пользователям роскошь игнорировать сложные возможности, которые не собьют с толку более опытных пользователей. Можно считать, что такие окна способны находиться в двух режимах – для новичков и для опытных пользователей. Однако проектировать их следует внимательно. Наличие в программе двух разных окон – одного для новичков, а другого для опытных пользователей – зачастую одно-

временно оскорбляет начинающих и создает определенные сложности для остальных. Как правило, разумно сделать так, чтобы окно сохраняло свое состояние между вызовами. Разумеется, следует не забыть и о кнопке Меньше или Свернуть, которая возвращает окно в простой режим (такая кнопка есть в окне Найти и заменить, рис. 24.10).

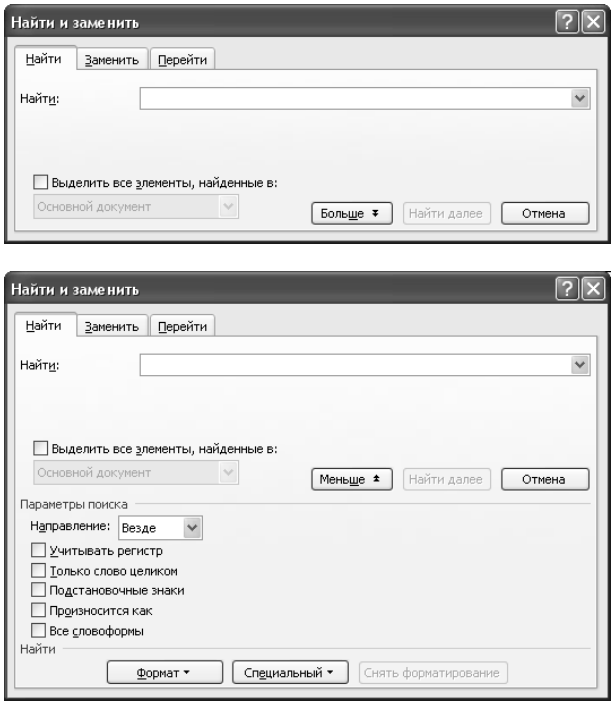


Рис. 24.10. *Окно Найти и заменить в Microsoft Word – пример разворачивающегося диалогового окна. Вверху показано исходное состояние окна, внизу – то, каким оно становится после нажатия на кнопку Больше*

Каскадные диалоговые окна

Каскадные диалоговые окна – дьявольская идиома, позволяющая элементам управления (обычно кнопкам) одного окна открывать дополнительные, вложенные окна. Второе окно при этом обычно перекрывает первое. Иногда второе окно может вызывать третье. Ну и мешанина! По счастью, каскадные диалоговые окна вышли из моды, хотя примеры до сих пор можно встретить. На рис. 24.11 приведен пример такого окна из Windows Vista.

Каскадные диалоговые окна попросту затрудняют понимание того, что происходит. Одна из проблем заключается в том, что второе окно перекрывает первое, по крайней мере, частично. Но это не самое главное – в конце концов, комбо-списки и контекстные меню тоже пере-

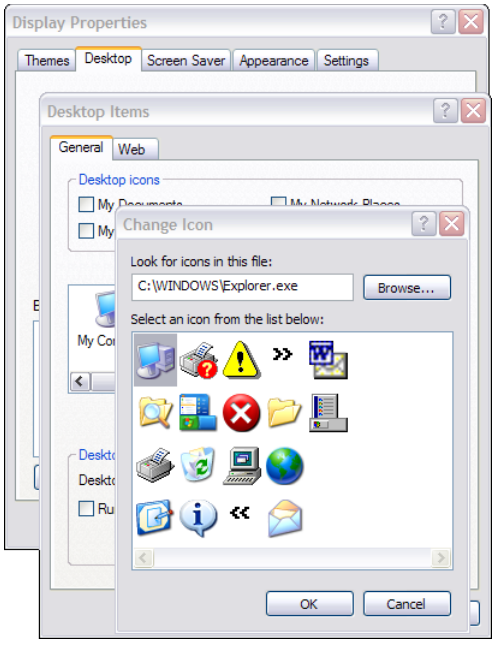


Рис. 24.11. *В операционной системе Windows до сих пор встречаются каскадные диалоги. Каждое диалоговое окно обладает собственными терминальными кнопками, и в результате возникает неоднозначность толкования и излишняя нагрузка на пользователя*

крывают часть окна, а само второе диалоговое окно можно отодвинуть в сторону. Основная путаница возникает из-за второго набора терминальных кнопок. Какова область действия каждой из кнопок Отмена и что мы подтверждаем кнопкой OK?

Сила диалоговых окон с вкладками – в их способности справляться с большими наборами элементов, тогда как каскадные диалоговые окна лучше подходят для детализации подачи материала. Проблема состоит в том, что чрезмерная глубина – основной признак избыточного усложнения интерфейса. Если вы обнаружите, что каскадные диалоговые окна в вашей программе необходимы для чего-нибудь иного, нежели представление малоизвестного материала, в котором ваши пользователи будут нуждаться редко, вам следует пересмотреть общую структуру взаимодействия – возможно, вы обнаружите ряд серьезных структурных изъянов.

Диалоговые окна способны стать верными помощниками в достижении целей пользователя, не оказываясь непреодолимыми преградами на его пути. Сохраняя простоту диалоговых окон и вызывая их только тогда, когда действительно необходимо хранить часть функций внутри отдельной «комнаты», вы сумеете сохранить поток внимания ваших пользователей, обеспечив им успех и снискав их благодарность.

25

Ошибки, уведомления, подтверждения

В главе 24 мы обсуждали информационные диалоговые окна, обычно создаваемые приложениями в одностороннем порядке, когда они сталкиваются с проблемой, или когда чувствуют себя неспособными принять решение самостоятельно, или вообще в любой момент, когда им есть о чем уведомить пользователя. Иными словами, эти диалоговые окна используются для вывода сообщений об ошибках, уведомлений и запросов на подтверждение – трех самых популярных в плане злоупотребления компонентов современных графических пользовательских интерфейсов. При правильном подходе к проектированию почти все такие диалоговые окна могут быть исключены. В этой главе мы исследуем вопрос о том, как и почему их следует исключать.

Диалоги сообщений об ошибках

Вероятно, никакой другой идиомой пользовательского интерфейса не злоупотребляют больше, чем сообщениями об ошибках. Обычно это плохо сформулированные, бесполезные, грубые сообщения, и, что хуже всего, они появляются, как правило, слишком поздно, чтобы помочь предотвратить ошибку. Пользователи не *хотят* получать сообщения об ошибках. Они хотят избежать *последствий* допущенных ошибок. Сказать, что желание избежать последствий эквивалентно желанию получать сообщения об ошибках, – все равно что утверждать, будто они хотят воздержаться от катания на горных лыжах, в то время как они на самом деле хотят избежать травм. Как указывает крупнейший авторитет в области юзабилити Дональд Норман, пользователи часто винят себя в ошибках, которые допустили проектировщики продукта. Если вы не получаете жалоб от пользователей, это еще не означает, что они радуются, когда получают сообщения об ошибках.

Допущение о том, что приложение не имеет права, даже по служебной надобности, отвергать ввод пользователя, считается такой страшной ересью, что многие проектировщики отметают его с порога. И все же мы считаем, что, если рассмотреть это допущение с позиций логики и с точки зрения пользователей, оно станет не только приемлемым, но и разумным.

Почему сообщений об ошибках так много

Первые компьютеры имели недостаточно памяти и мощности, были дорогими и не поддавались легкой программной настройке. С этими машинами работали ученые в белых халатах, которые с пониманием относились к нуждам центрального процессора и не обижались, получая сообщения об ошибках. Они знали, как трудна работа компьютера. Они ничего не имели против получения дампа оперативной памяти, сбоя, сообщения «Abort, Retry, Fail?» (Отмена, Повтор, Сбой?) или печально знаменитого сообщения «FU» (File Unavailable – файл недоступен). Так зарождалась традиция отношения программ к людям как к процессорам. С первых же дней развития компьютерных технологий программисты приняли как аксиому утверждение, что лучший способ взаимодействия программы с пользователем – требовать ввода данных и жаловаться, когда человеку не удается достичь того же уровня совершенства, который присущ центральному процессору.

Последствия этого подхода проявляются всякий раз, когда программа требует от пользователя, чтобы он вел себя так, как надо ей, вместо того чтобы адаптироваться к потребностям человека. Сильнее всего это заметно в вездесущих сообщениях об ошибках.

Что не так с сообщениями об ошибках

Сообщения об ошибках неизбежно прерывают работу модальными диалоговыми окнами. Многие проектировщики и программисты воображают, что их сообщения об ошибках предупреждают пользователя о серьезных проблемах. Это широко распространенное заблуждение. Большинство сообщений об ошибках информируют пользователя о неспособности программы вести себя гибко и являются признанием в глупом поведении со стороны приложения. Иными словами, в глазах большинства пользователей сообщения об ошибках выглядят не просто как прерывание программой рабочего процесса, а как *идиотское прекращение работы*. Мы можем значительно повысить качество наших интерфейсов, отказавшись от диалоговых окон с сообщениями об ошибках.

ПРИНЦИП проектирования

Диалоги с сообщениями об ошибках прекращают работу из-за ерунды; их следует избегать.

Люди ненавидят сообщения об ошибках

Люди обладают чувствами. Компьютеры – нет. Когда один программный компонент отвергает данные, переданные другим компонентом, компоненту-отправителю все равно. Он не сердится, не обижается, не просит совета. А вот люди очень злятся, когда им без обиняков говорят, что они идиоты.

Когда пользователь получает сообщение об ошибке, это выглядит так, словно другой человек обвиняет его в глупости. Пользователи ненавидят это чувство (рис. 25.1). Несмотря на неизбежную реакцию пользователей, большинство программистов лишь пожмут плечами и продолжат выводить сообщения об ошибках. Они не знают другого способа создать надежный программный продукт.

Многие программисты и проектировщики пользовательского интерфейса придерживаются ошибочного убеждения, что люди нуждаются в том, чтобы их извещали об их ошибках. Это допущение ложно в нескольких отношениях. Во-первых, оно игнорирует человеческую природу: мало кому хочется услышать от машины, что он неправ. Можно говорить, что это отрицание виновности, но такова правда: пользователи скорее обвинят того, кто принес дурную весть, чем самих себя.

Предположение, что пользователям *требуется* знать о факте своей ошибки, столь же ложно. Насколько вам важно знать, что вы указали недопустимый размер шрифта? Большинство программ в состоянии установить разумное значение самостоятельно.

Все мы считаем крайне невежливым сообщать другим, что они нарушили правила приличия. Сказать кому-то, что у него к зубам прилипли остатки пищи или что у него брюки расстегнуты, означает создать ситуацию, неловкую для обоих участников разговора. Воспитанные люди находят способ привлечь внимание собеседника к проблеме незаметно для окружающих. А вот программисты полагают, что большое и заметное диалоговое окно в центре экрана, которое прерывает работу пользователя и издает громкий неприятный звук, является достойным поведением.

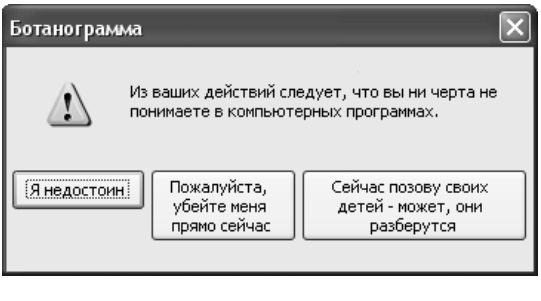


Рис. 25.1. *Неважно, насколько вежливо сформулированы ваши сообщения об ошибках, – интерпретированы они будут вот так*

И все-таки: чья это ошибка?

Принято считать, что сообщения об ошибках уведомляют пользователя, что он что-то сделал неправильно. Фактически же большинство сообщений об ошибках сообщают пользователю, что *компьютер* сел в лужу.

Пользователи совершают гораздо меньше существенных ошибок, чем принято думать. Типичные «ошибки» заключаются в том, что пользователь нечаянно ввел число, выходящее за установленные границы, или поставил пробел там, где компьютером пробел запрещен. Когда пользователь вводит что-то, непонятное для компьютера, чья в этом вина? Виноват ли пользователь, что он недостаточно хорошо умеет пользоваться программой, – или же виновата программа, не разъяснившая ему причины и следствия?

Информацию, введенную в неправильном порядке, программа обычно считает ошибочной, однако у людей в аналогичных ситуациях проблем не бывает. Люди умеют ждать, пока рассказ не закончится. Программа же приходит к ложному заключению, что не соответствующий установленному порядку ввод неверен, и выдает зловещее сообщение об ошибке.

Если, например, пользователь выпишет счет и не укажет идентификационный номер клиента, большинство приложений отвергнут введенные данные. Они остановят работу из-за ерунды, полагая, будто пользователь должен указать номер клиента *немедленно*. Но ведь программа вполне может принять транзакцию, предположив, что правильный номер будет введен позже или что оператор просто пытается создать новую учетную запись клиента. Программа могла бы сообщить посредством немодальной связи, что номер не опознан, а затем проследить, чтобы пользователь ввел информацию, необходимую для исправления идентификационного номера клиента, до конца сеанса или даже до конца отчетного периода. Именно так работают люди. Они не вводят «плохие» данные – они просто вводят данные в том порядке, к которому программа не готова.

Если человек забудет разъяснить компьютеру все до мелочей, компьютер может после некоторой разумной паузы более настойчиво потребовать необходимую информацию. В конце рабочего дня или недели программа может визуально выделить неким образом все противоречивые транзакции. Ей вовсе не обязательно прерывать работу сообщением об ошибке. Ведь она запомнила транзакции – и их можно найти и исправить. Так работают системы ручной обработки информации. Почему компьютеризированная система не может поступить подобным образом? Зачем останавливать весь рабочий процесс только потому, что чего-то не хватает? Пока пользователь в курсе того, что некоторые счета нужно привести в порядок, проблем не будет. Хитрость в том, чтобы информировать его, не прерывая работу. Эту идею мы более подробно обсудим позже.

Если бы на месте программы был живой человек, который устроил бы забастовку посреди бухгалтерии потому, что мы передали ему не до конца заполненную форму, мы вышли бы из себя. Будь мы его начальниками, мы тут же стали бы искать замену такому упрямому, мелочному и самодовольному клерку. «Возьми форму, – сказали бы вы ему, – и впиши, чего там не хватает». Авторам приходилось пользоваться справочными системами, которые требуют указать вместе с телефоном код города даже тогда, когда адрес человека уже введен. Не надо много ума, чтобы в такой ситуации сделать разумное предположение о коде города. Если пользователь вводит новое имя и указывает уже известный программе город, она может выяснить код, посмотрев на записи о других 25 адресатах, живущих в этом городе. Конечно, если вы введете город, которого еще нет в базе данных программы, это может поставить ее в тупик. Но так ли трудно обратиться к справочнику в Интернете или хотя бы держать в памяти список тысячи крупнейших городов страны с их телефонными кодами?

Программисты, вероятно, заявят протест: «Программа может ошибиться. Она не знает наверняка. Есть города с несколькими кодами. Программа не должна делать предположения без одобрения пользователя!» Все не совсем так.

Если мы попросим помощника ввести контактный телефон клиента в программу-справочник и не сообщим ему код города, он выполнит поручение, полагая, что код выяснится до того, как потребность в нем станет неотложной. А еще он может самостоятельно выяснить код в справочнике. Предположим, что клиент живет в Лос-Анджелесе, и в справочнике неоднозначная информация: код либо 213, либо 310. Если помощник ворвется к нам в кабинет с криками: «Прекратите работу! Код города клиента неоднозначен!», то мы испытаем острейшее желание уволить его и нанять сотрудника, у которого коэффициент умственного развития все-таки превышает комнатную температуру. Почему программный продукт должен ввести себя иначе, нежели нормальный работник? Человек в такой ситуации может просто вписать в поле кода города *213/310*. Когда мы будем звонить клиенту, нам придется уточнить код, но пока что можно жить спокойно.

Снова слышны крики протеста: «Но поле кода города вмещает только три цифры! Оно не может принять запись „213/310“!» О да, это ужасно. Вы хотите сказать, что дизайн пользовательского интерфейса, созданный на основе внутренней модели реализации (жестко ограниченного размера поля ввода), заставляет вас отвергать естественное человеческое поведение в пользу оскорбительной компьютерной логики, сопровождаемой унизительными сообщениями об ошибках? Проще говоря, источником сообщений об ошибках является неспособность приложений вести себя разумно, а отнюдь не промахи пользователей.

Сообщения об ошибках не работают

По иронии судьбы сообщения об ошибках *не удерживают пользователей от совершения ошибок*. Мы воображаем, будто пользователи избегают проблем благодаря нашим надежным сообщениям об ошибках, направляющим их на путь истинный, – но это заблуждение. Что они действительно делают, так это предохраняют от проблем *программу*. В большинстве программ сообщения об ошибках стоят, как часовые, вовсе не там, где пользователь наиболее уязвим, а в самых чувствительных местах программы, и самозабвенно служат идее, что программа важнее любого пользователя. Имея дело с нашими программными продуктами, пользователи сталкиваются с массой проблем независимо от качества и количества сообщений об ошибках. Все, на что способно сообщение об ошибке, – это удержать меня от ввода буквенных символов в числовое поле. Оно не удержит меня от ввода недопустимого числа – это гораздо более трудная задача проектирования.

Избавляемся от сообщений об ошибках

Мы не можем исключить сообщения об ошибках, просто выбросив ту часть кода, которая выводит на экран диалоговое окно с сообщением, и позволив программе завершаться аварийно в случае возникновения проблем. Мы должны перепроектировать приложения таким образом, чтобы они не были восприимчивы к этой проблеме. Мы должны заменить сообщение об ошибке более надежным кодом, который не допускает возникновения ошибочных ситуаций, а не просто жалуется, что дела идут не в точности так, как нужно. Проведя некое подобие вакцинации, мы вырабатываем у программы иммунитет к проблеме, и только после этого можем выкинуть сообщение, которое извещало об этой проблеме. Чтобы избавиться от сообщения об ошибке, мы должны снизить вероятность совершения этой ошибки пользователем. Вместо того чтобы воспринимать сообщения об ошибках как норму, мы должны думать о них как об исключительных решениях редких проблем, то есть как о хирургии вместо аспирина. Мы должны считать их идиомой крайних мер.

Каждый хороший программист знает, что если модуль A передаст недопустимые данные модулю B, то последний должен немедленно и явным образом отвергнуть эти данные с соответствующей индикацией ошибки. Отсутствие такой реакции было бы серьезным недостатком интерфейса между этими двумя модулями. Однако люди – не модули программного кода. Программный продукт не просто обязан перестать отвергать введенные человеком данные, вывешивая сообщение об ошибке, – проектировщики продукта должны пересмотреть само понятие «недопустимых данных». Когда данные исходят от человека, программа обязана исходить из того, что ввод корректен, – просто потому, что человек важнее программы. Вместо того чтобы отвергать ввод пользователя, программа должна изо всех сил стараться разобраться в непо-

нятных данных и привести их в согласованный вид. Программа, возможно, знает, что происходит внутри компьютера, – но только пользователь понимает положение вещей в реальном мире. В конце концов, реальный мир важнее того, что думает компьютер.

Предотвращаем ошибки

Сделать так, чтобы пользователь не мог ошибиться, – лучший способ избавиться от сообщений об ошибках. С помощью ограничивающих элементов ввода, таких как счетчики и раскрывающиеся списки, можно лишить пользователей возможности вводить ошибочные числа. Вместо того чтобы заставлять пользователя вводить информацию с клавиатуры, предоставьте ему список, из которого он будет выбирать. Например, не требуйте от него код штата – дайте ему список всех кодов штатов, а еще лучше – нарисуйте карту. Иными словами, исключите малейшую возможность ввода неверного кода штата.

Делайте так, чтобы ошибки были невозможны.

Другой превосходный способ избавиться от сообщений об ошибках – сделать программу достаточно сообразительной, чтобы она не требовала лишнего. Многие сообщения об ошибках имеют такой вид: «Неправильный ввод. Пользователь должен ввести XXXX». Если программа знает, что именно должен ввести пользователь, почему бы ей не ввести это самой, перестав трепаться попусту? Вместо того чтобы требовать от пользователя ввести имя файла и давать ему возможность ошибиться, заставьте программу запомнить, с какими файлами он работал раньше, и выведите их список. Еще один пример: сделайте так, чтобы программа получала дату от системных часов, а не требовала ее пользователя.

Несомненно, все подобные решения потребуют дополнительной работы программистов. Но ведь именно программисты получают деньги за то, чтобы пользователь был удовлетворен, – а никак не наоборот. Если программист считает пользователя всего лишь еще одним устройством ввода, легко позабыть, кто кому служит в мире проектирования программных продуктов.

Пользователи компьютеров безразличны к проблемам, стоящим перед программистами. Они не понимают технических причин появления диалогового окна с сообщением об ошибке. Все, что они видят, – это нежелание программы вести себя по-человечески. Они воспринимают все сообщения об ошибках как вариант того, что показано на рис. 25.2.

Одной из проблем сообщений об ошибках является то, что они уведомляют о неприятностях *по факту*. Они говорят: «Случилось нечто плохое, и все, что *вы* можете сделать, – признать катастрофу». Полезность таких сообщений сомнительна. И все эти диалоговые окна снаб-

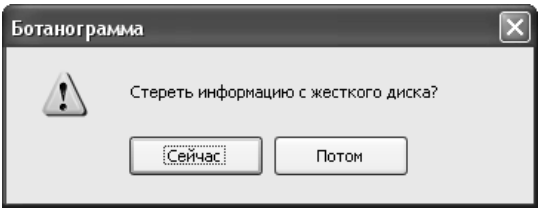


Рис. 25.2. *Именно так большинство пользователей воспринимает диалоговое окно с сообщением об ошибке. Оно представляется допросом в стиле Кафки, когда каждый следующий ответ все глубже погружает в яму возмездия и раскаяния*

жены кнопкой OK, требующей от пользователя стать соучастником преступления. Подобные сообщения об ошибках вызывают в памяти сцену из старых фильмов о войне. Злополучный солдат, пробирающийся по рисовому полю, наступает на противопехотную мину. Он и его товарищи отчетливо слышат щелчок взрывного механизма – и солдат понимает, что сейчас он в безопасности, но стоит лишь убрать ногу с мины – она сработает, и он потеряет какую-то большую и полезную часть тела. Примерно то же чувствуют пользователи при виде большинства сообщений об ошибках: им хочется оказаться за тысячи километров от этого места – где-нибудь в реальном мире.

Положительная обратная связь

Одной из причин, затрудняющих изучение программных продуктов, является отсутствие положительной обратной связи. Люди быстрее учатся на положительной обратной связи, чем на отрицательной. Они хотят использовать свои программы правильно и эффективно, у них есть мотивация стремиться к тому, чтобы программы работали на них. Им не нужно, чтобы их били по рукам, когда они оплошали. Им нужно, чтобы их поощряли или хотя бы извещали, когда они добились успеха. Одобрение повышает их самооценку – и они платят программе хорошим ответным отношением.

Защитники отрицательной обратной связи могут привести множество примеров ее эффективности при управлении поведением людей. Доказательства неоспоримы, однако эффективная карательная обратная связь почти всегда помогает удержать от того, что *не* следует делать: превышение скорости, супружеская измена, уклонение от уплаты налогов. Однако когда речь идет о *помощи* людям в их деле, положительная обратная связь гораздо лучше. Если вы когда-нибудь учились кататься на лыжах, то знаете, что орущий инструктор мало чем способен помочь.

Программа унижает пользователей, когда сообщает, что они ошиблись.

Не следует забывать, что мы говорим о недостатках отрицательной обратной связи, исходящей от компьютера. Отрицательная обратная связь, исходящая от другого человека, хотя и неприятна, но может быть оправданной в определенных обстоятельствах. В конце концов, тренер закаляет разум спортсмена, готовя его к соревнованиям, а строгий учитель готовит ученика к превратностям взрослой жизни. Но получить отрицательную обратную связь от машины – оскорбление. Старшина и учитель – люди, которые как минимум имеют реальный опыт и определенные заслуги. Но слышать от программы, что ты все делаешь не так, – оскорбительно и унизительно. Пользователи вполне справедливо ненавидят, когда их оскорбляют и унижают. Внутри компьютера нет ничего такого, что давало бы ему право унижать и оскорблять пользователя-человека. Мы прибегаем к отрицательной обратной связи лишь в силу устоявшейся традиции.

Разве исключений нет?

С ростом наших технологических возможностей растет гибкость и мобильность компьютерной аппаратуры. Современные компьютеры можно подсоединять к сетям и периферийным устройствам и отсоединять от них, даже не выключая питания. Это означает, что теперь аппаратные устройства могут появляться и исчезать в любой момент. Принтеры, модемы и файловые серверы приходят и уходят, как прилив и отлив. С развитием беспроводных сетевых технологий WiFi и Bluetooth наши компьютеры получили возможность часто подключаться к сетям и отключаться от них. Разве это ошибка, если вы попытались распечатать документ и обнаружили, что принтер не подключен? Разве это ошибка, если файл, с которым вы работаете, расположен на дисководе, недоступном в данный момент?

Ни одну из этих ситуаций нельзя считать ошибочной. Если вы откроете файл на сервере и станете его редактировать, а затем пойдете в ресторан пообедать и прихватите с собой ноутбук, программа должна понять, что нормальное место хранения файла сейчас недоступно, и предпринять какие-то разумные действия. Она, например, может воспользоваться беспроводной сетью и VPN-подключением, чтобы установить удаленное соединение с сервером, или просто сохранить изменения, сделанные вами за время обеда, на локальном диске, чтобы синхронизировать их с версией на сервере, когда вы вернетесь в офис. Как бы то ни было, это нормальное поведение, а не ошибка, и вы не обязаны всякий раз в подобной ситуации говорить компьютеру, что ему делать.

Можно исключить почти все диалоговые окна с сообщениями об ошибках. Если вы подойдете к задаче с идеей о том, что от сообщения об ошибке надо обязательно избавиться, а код переписать соответствующим образом, вы увидите справедливость этого утверждения. Вас удивит, как мало изменений придется внести в программу, чтобы достичь этой цели. В тех редких случаях, когда отказ от сообщения об ошибке

потребовал бы серьезных изменений в коде, приходится идти на компромисс и продолжать использовать диалоговые окна. Однако программистам пора рассматривать такие компромиссы как свои недоработки, крайнее средство.

При всем том определенно существуют ситуации, в которых важно время реагирования; в таких ситуациях пользователей обязательно нужно уведомлять о происходящем, настойчиво требуя их внимания. К примеру, если во время торгов управляющий инвестициями менеджер захочет провести некие сделки до конца дня, но отдаст распоряжение уже после закрытия рынка, то следует прервать его деятельность и сообщить, что сделки не будут обработаны до открытия рынка следующего дня. Возможно, при этом условии менеджер захочет отменить сделки.

Улучшаем сообщения об ошибках: последнее средство

Для тех случаев, когда перепроектирование приложения с целью избавиться от сообщений об ошибках может обойтись слишком дорого, мы предлагаем несколько способов повышения качества диалоговых окон этих сообщений. Пользуйтесь описанными ниже рекомендациями лишь в крайнем случае, когда нет никакой возможности избавиться от окна с сообщением об ошибке и отсутствуют другие разумные варианты.

Диалоговое окно с сообщением об ошибке должно быть *вежливым*, *информативным*, *действенным*. Никогда не забывайте, что сообщение об ошибке свидетельствует о неспособности программы выполнить *свою* работу и отвлекает пользователя от выполнения *его* работы. Сообщение об ошибке должно быть исключительно вежливым. Оно никоим образом не имеет права намекать на причастность пользователя к возникновению проблемы, поскольку с точки зрения пользователя это будет неправдой.

Окно с сообщением об ошибке должно всесторонне охарактеризовать проблему. Иными словами, столкнувшись с проблемой, программа должна предоставить пользователю информацию, которая необходима для принятия решения по устранению этой проблемы. Следует четко описать проблему, возможные варианты решения, действия, которые предпримет программа по умолчанию, и сообщить, утрачены ли данные, и если да, то какие именно.

Однако было бы неправильно со стороны программы полностью переложить проблему на плечи пользователя и умыть после этого руки. Она обязана предложить как минимум одно решение здесь же, в диалоговом окне. Окно должно содержать кнопки, позволяющие справиться с проблемой тем или иным способом. Если, скажем, недоступен принтер, диалоговое окно должно предложить возможность подключить другой принтер или отложить печать. Если база данных безнадежно разрушена, программа должна предложить пользователю вос-

становление базы данных, сообщив ему, сколько времени это займет и какие побочные эффекты могут возникнуть.

На рис. 25.3 приведен пример разумного сообщения об ошибке. Обратите внимание: сообщение вежливое, информативное и действенное. Оно не содержит ни малейшего намека на то, что поведение пользователя не вполне безупречно.

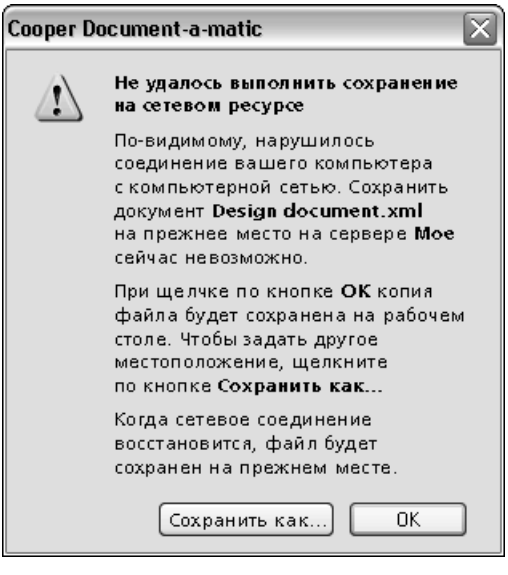


Рис. 25.3. *Если необходимо создавать окно с сообщением об ошибке, оно должно выглядеть приблизительно так. Это сообщение вежливо и понятно описывает проблему и предлагает хороший вариант решения. Кроме того, оно описывает, к каким последствиям приведет нажатие на кнопки, предложенные окном*

Диалоговые окна уведомлений: сообщение об очевидном

Как и диалоговые окна с сообщениями об ошибках, уведомления и подтверждения прерывают работу (часто из-за ерунды), однако в их задачу не входит сообщение о сбоях. **Уведомление** сообщает пользователю о действии, предпринятом программой, а **подтверждение**, кроме того, дает пользователю полномочия отменить это действие. Эти диалоговые окна вылезают как сорная трава в большинстве программ, и от них следует избавляться, как от сообщений об ошибках, давая дорогу более полезным идиомам.

Уведомления обычно нарушают один из базовых принципов проектирования: *диалоговое окно – это еще одна комната; не ходите в комна-*

ту без веской причины (глава 20). Даже если пользователя необходимо проинформировать о предпринятом программой действии, зачем для этого ходить в другую комнату?

Говоря более конкретно, приложение либо должно иметь достаточно смелости и уверенности в себе, либо не предпринимать действий без прямого указания пользователя. Например, если программа сохраняет файл пользователя на диске автоматически, она должна быть уверена, что все делает правильно. Она обязана сообщить пользователю о том, что сделала, не прерывая, однако, для этого работу. Если программа не уверена, что ей следует сохранить файл, пусть не сохраняет его и возложит это действие на пользователя.

И наоборот: если пользователь дает программе прямое указание сделать что-то, например перетаскивает файл в мусорную корзину, ей не следует прерывать работу ради ерундового объявления о том, что перемещение в корзину прошло успешно. Программа должна сопроводить действие адекватной визуальной обратной связью. Для тех случаев, когда пользователь сделал этот жест по ошибке, программа должна ненавязчиво предлагать ему надежную функцию отмены, чтобы он мог отменить ошибочное действие.

Идея уведомлений – в том, чтобы информировать пользователя. Это замечательная цель, но нельзя достигать ее ценой разрушения плавного потока взаимодействия.

На рис. 25.4 показано, как уведомления могут приносить больше хлопот, чем пользы. Диалоговое окно Найти и заменить (то, которое находится на заднем плане) уже требует, чтобы пользователь щелкнул по кнопке Отмена, когда поиск завершится, но диалоговое окно с уведомлением, наложенное поверх диалогового окна Найти и заменить, добавляет еще одну кнопку, разрушающую состояние потока. Чтобы вернуться

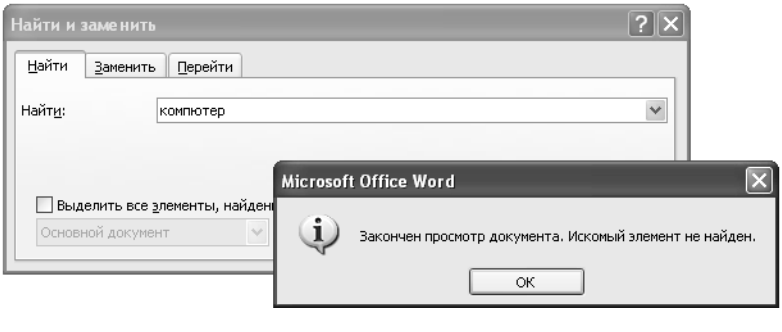


Рис. 25.4. *Типичное диалоговое окно с уведомлением. Оно является ненужным, неуместным и прерывает поток взаимодействия из-за ерунды. Программа Word закончила поиск в документе. Следует ли сообщать об этом факте через отдельный механизм вместо механизма поиска? Если нет, то зачем здесь создается еще одно окно?*

к работе, пользователю придется щелкнуть сначала по кнопке OK в диалоговом окне уведомления, а затем по кнопке Отмена в окне поиска. Если бы информационный аспект уведомления был встроен в диалоговое окно поиска, пользователю стало бы вдвое легче жить.

Диалоговые окна с уведомлениями столь распространены потому, что их легко создавать. Большинство языков программирования позволяют вывести окно с сообщением при помощи одной строки кода. А вот создание анимированного индикатора состояния, интегрированного в основное окно, может потребовать тысячи и более строк кода. В такой ситуации нельзя рассчитывать, что программисты сделают правильный выбор. Налицо конфликт интересов, и проектировщикам следует точно указывать в спецификациях, в каком месте интерфейса следует выводить информацию. Проектировщики должны проследить за тем, чтобы ни один пункт спецификаций не был принесен в жертву быстрому написанию кода. Представьте себе, что подрядчики на стройке в одностороннем порядке решат обойтись без ванной комнаты, потому что с прокладкой труб слишком много хлопот. О последствиях легко догадаться...

Разумеется, программный продукт должен информировать пользователя о своих действиях. Он должен предлагать визуальную индикацию в главном окне, чтобы информация о положении дел была доступна пользователю в любой момент, когда он того пожелает. Вывод уведомления о некоем незапрошенном действии – это просто плохое решение. Вывод уведомления о *запрошенном* действии – это уже патология.

Программный продукт обязан вести себя гибко и прощать пользователя, но он не должен лебезить и угодничать. Диалоговое окно на рис. 25.5 – классический пример уведомления, которое хорошо бы

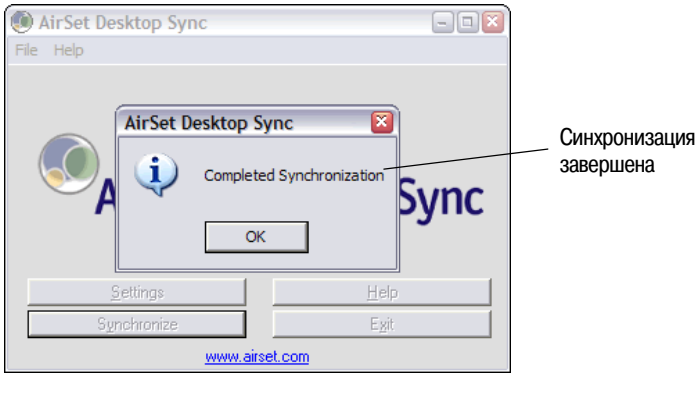


Рис. 25.5. *Это окно из AirSet Desktop Sync излишне угодливо. Мы попросили выполнить синхронизацию – и наша работа тут же прервана вот этим чрезвычайно важным сообщением. Да, прекрасно, что программе удалось-таки справиться со своей работой, но зачем нас-то отвлекать? Видимо для того, чтобы у нас была возможность признать ее заслуги*

пристрелить, чтобы мы не мучились. Оно констатирует, что приложение успешно завершило синхронизацию, – и это единственная причина его существования. Диалоговое окно выводится через несколько секунд после того, как пользователь дал команду синхронизации. Оно прерывает нашу работу, чтобы объявить об очевидном, – как будто приложение желает получить поощрение за свою тяжкую работу. Если бы так себя вел человек, мы сочли бы работу с ним некомфортной, а его самого – заносчивым. Разумеется, обратная связь необходима, но действительно ли для нее нужно дополнительное окно, которое придется закрывать вручную?

Диалоговое окно подтверждения

Когда приложение не чувствует уверенности в своих действиях, оно нередко просит у пользователя одобрения и выдает диалоговое окно, подобное показанному на рис. 25.6. Иногда такое **подтверждение** предлагается пользователю потому, что программа сомневается в его действиях. В других случаях программа чувствует себя недостаточно компетентной для принятия решения и предлагает пользователю сделать выбор. Окна подтверждений всегда появляются по инициативе программы и никогда – по инициативе пользователя. Это означает, что они часто не связаны с целями пользователя и являются отражением модели реализации.

Демонстрация модели реализации перед пользователями – верный способ создать неприятный и некачественный продукт. Это означает, что диалоговые окна подтверждений неуместны. Они возникают в коде

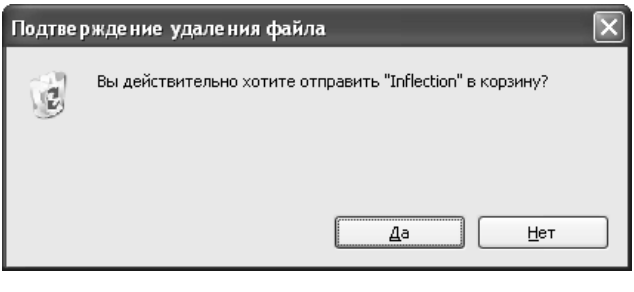


Рис. 25.6. *Каждый раз, когда мы удаляем в Windows файл, мы получаем это диалоговое окно с вопросом о том, уверены ли мы в своих действиях? Да, мы уверены. Мы всегда уверены. А если мы ошибемся, мы ожидаем, что операционная система восстановит файл. Специально для этого в Windows существует Корзина. Зачем тогда выдается диалоговое окно подтверждения? Если окно подтверждения выводится постоянно, пользователи привыкают подтверждать действия программы не глядя. Поэтому когда оно, в конце концов, сообщит о нависшей угрозе, пользователь по привычке «даст добро»... Сделайте одолжение своим пользователям – не создавайте диалоговые окна подтверждений*

программы тогда, когда программист заходит в тупик. Обычно это случается, если программе требуется принять решительные меры, а программист чувствует, что не может взять на себя ответственность за них. Иногда эти решительные меры связаны с каким-то условием, которое распознала программа, но чаще они определяются действиями пользователя. Как правило, диалоговое окно подтверждения появляется тогда, когда исполняемая команда либо необратима, либо может привести к нежелательным последствиям.

Подтверждения перекладывают ответственность на плечи пользователя. Пользователь доверил приложению работу – и оно должно не просто выполнить ее, но выполнить как следует. Правильное решение – сделать действие легко обратимым и обеспечить достаточно понятную немодальную обратную связь, чтобы неприятности не заставали пользователя врасплох.

В ходе разработки продукта по мере роста объема программного кода программисты обнаруживают все больше ситуаций, с которыми они не в силах справиться адекватно. В этих местах программисты в одностороннем порядке практически неосознанно пишут код, перекладывающий ответственность на плечи пользователя. Необходимо внимательно следить за этим процессом, поскольку программисты склонны создавать диалоговые окна даже после согласования спецификаций на интерфейс пользователя: они просто не считают диалоговые окна частью интерфейса.

Диалоговое окно, которое кричало: «Волк!»

Подтверждения иллюстрируют интересный парадокс человеческой психики: они срабатывают, только если их не ожидают. Это обстоятельство не кажется важным, пока вы не изучите контекст. Если запрашивать подтверждение в обычных ситуациях, пользователь быстро привыкает к таким диалоговым окнам и выдает подтверждение не глядя. Подобная реакция становится делом столь же обычным, как появление самих диалоговых окон. Если в какой-то момент возникнет действительно неожиданная и опасная ситуация, к которой *следует* привлечь внимание пользователя, пользователь просто в силу привычки по-быстрому расправится с подтверждением. Подобно мальчику из притчи, который кричал: «Волк!», – диалоговое окно подтверждения не подействует в случае реальной опасности, потому что оно слишком много раз кричало, когда никакой опасности не было.

Чтобы диалоговые окна подтверждения работали, они должны появляться в тех случаях, когда пользователь почти наверняка щелкнет по кнопке Нет или Отмена, и они *ни в коем случае* не должны появляться тогда, когда пользователь скорее щелкнет по кнопке Да или OK. И с этой точки зрения они начинают выглядеть довольно бессмысленными, не правда ли?

Избавляемся от подтверждений

Три принципа проектирования дают нам возможность избавиться от диалоговых окон подтверждений. Лучше всего придерживаться простого правила: делайте, а не задавайте вопросы. Проектируя программный продукт, придайте ему уверенности (разумеется, положив в основу этой уверенности исследование пользовательской аудитории, описанное в главе 4). Пользователи с уважением отнесутся к немногословности и уверенным действиям вашей программы.

Не расспрашивайте – действуйте.

Конечно, если программа уверенно делает то, что может не понравиться пользователю, она должна предоставить ему возможность отмены нежелательной операции. Каждое действие программы должно быть обратимым. Вместо того чтобы заранее спрашивать разрешения с помощью подтверждения, просто позвольте пользователю выполнить команду Стоп-и-Отмена в тех редких случаях, когда действия программы оказались неуместными.

Большинство ситуаций, которые кажутся нам несовместимыми с командой отмены, в действительности вполне допускают применение этой команды. Хороший пример – удаление файла или запись нового файла поверх существующего. Имеющийся файл можно переместить во временный каталог, где он будет храниться месяц или около того, пока не подвергнется физическому удалению. Корзина в Windows использует подобную стратегию, но файлы из нее не удаляются автоматически через месяц, и пользователям приходится выносить мусор вручную.

Все действия должны быть обратимыми.

Действовать в спешке и заставлять пользователя спасать ситуацию с помощью операции отмены – не идеальный подход. Ваше приложение способно предоставить пользователю достаточно информации, чтобы он не стал выполнять команду, приводящую к нежелательным результатам, и не забыл выполнить необходимую команду. Программа должна предлагать обогащенную визуальную обратную связь, чтобы пользователь постоянно находился в курсе дел, – подобно тому как приборный щиток информирует водителя о состоянии автомобиля.

Помогайте пользователям избегать ошибок при помощи немодальной обратной связи.

Иногда возникают ситуации, в которых действительно невозможно использовать функцию отмены. Являются ли они законным основанием для выдачи окна подтверждения? Не обязательно. Более удачный подход – защищать пользователей, как мы защищаем водителей на автострадах – с помощью последовательной и понятной разметки. Часто есть возможность встраивать замечательные немодальные предупреждения прямо в интерфейс. В качестве примера рассмотрим диалоговое окно Adobe Photoshop (рис. 25.7), которое сообщает нам, что документ больше области печати. Почему программа дожидалась этого момента, чтобы информировать нас о данном факте? Почему бы не показывать на экране границы области печати все время (пока пользователь не скроет их)? Почему бы не выделять те части изображения, которые не могут быть напечатаны, когда пользователь наводит указатель мыши на кнопку Печать на панели инструментов? Прозрачная немодальная обратная связь (см. следующий раздел) является лучшим способом решения подобных проблем.

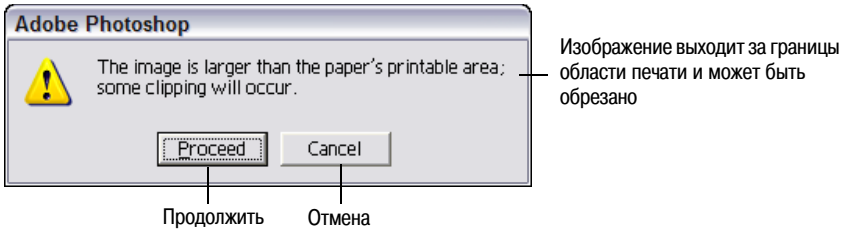


Рис. 25.7. *Это диалоговое окно приносит мало пользы и появляется слишком поздно. Почему бы программе не обозначать границы области печати пунктирными линиями прямо в интерфейсе? Нет никаких причин терроризировать пользователей подобными окнами*

Гораздо чаще действительно необратимых действий встречаются действия легкообратимые, но все же сопровождаемые диалогами подтверждения. Подтверждение на рис. 25.6 – яркий пример ненужного окна такого типа. Нет ни одной причины просить подтверждения для переноса файла в Корзину: она для того и существует, чтобы можно было отменить операцию удаления файлов.

Заменяем диалоговые окна обогащенной немодальной обратной связью

Большинство компьютеров (и многие устройства) оборудованы дисплеями высокого разрешения и качественными аудиосистемами. Тем не менее лишь немногие программы (кроме игровых) пытаются применять эти средства для передачи пользователю полезной информации о состоянии программы, пользовательских задач, а также операционной системы и периферийных устройств в целом. Существует обшир-

ный набор инструментов, способных информировать пользователя, однако проектировщики и программисты ограничиваются только одним примитивным инструментом – диалоговыми окнами. Нет нужды говорить, что информация о некоторых тонкостях состояния *вообще* не доносится до пользователей: ведь даже самые неумелые проектировщики знают, что пользователи не любят, когда диалоговые окна выскакивают *постоянно*. Однако постоянная обратная связь пользователям нужна. Просто канал передачи информации должен быть другим.

В этом разделе мы обсудим **обогащенную немодальную обратную связь** – информацию, появляющуюся в главном окне вашего приложения, не прерывающую рабочий процесс программы и пользователя и позволяющую практически полностью избавиться от надоедливых диалоговых окон.

Обогащенная немодальная обратная связь

Пожалуй, самым важным типом немодальной обратной связи является **обогащенная визуальная немодальная обратная связь** (ОВНОС). Эта обратная связь является *обогащенной* в том смысле, что она предоставляет пользователю подробнейшую информацию о состоянии или свойствах процесса или объекта в данном приложении; она является *визуальной*, потому что идиоматически (а часто еще и динамически) использует экранные пикселы; и она является *немодальной*, поскольку информация всегда присутствует на экране, не требуя для просмотра и понимания ни специальных действий, ни переключения режимов.

К примеру, в Microsoft Outlook 2007 небольшая пиктограмма рядом с именем отправителя электронного сообщения наглядно показывает, можно ли в данный момент поговорить с пользователем через Интернет или по телефону, если вдруг выяснится, что разговор в реальном времени будет удобнее беседы по электронной почте. Эта небольшая пиктограмма (а также возможность начать диалог по команде из контекстного меню) означает, что, когда пользователям понадобится выяснить, можно ли сейчас поговорить, им не придется для этого запускать чат-приложение и искать в нем имя отправителя. Это так просто и удобно, что пользователь буквально перестает задумываться о таких вещах. Другой вариант этой стратегии, реализованный для клиента компании Cooper, можно увидеть на рис. 25.8. А вот еще один пример, на этот раз с платформы Mac: когда вы загружаете файл из сети Интернет, он появляется на рабочем столе в виде пиктограммы с динамически обновляемым индикатором состояния загрузки, который наглядно показывает, сколько процентов файла уже загружено.

Последний пример ОВНОС мы позаимствовали из мира компьютерных игр – это «Цивилизация» (Civilization) Сида Мейера (Sid Meier). Интерфейс игры, представляющий собой карту мира, который вы как лидер новой цивилизации пытаетесь заселить и завоевать, дает массу примеров ОВНОС. Так, ОВНОС используется для вывода информации

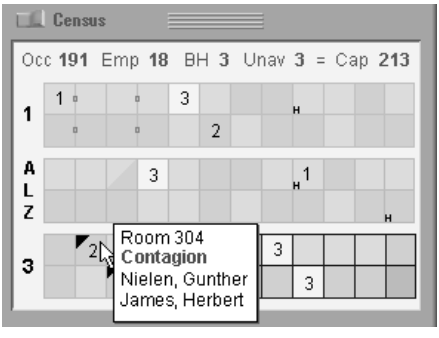


Рис. 25.8. *Эта панель является частью интерфейса информационной системы долгосрочного наблюдения за больными, созданной компанией Cooper, и являет собой хороший пример ОВНОС. Диаграмма представляет все палаты стационара. Цветовая кодировка отражает мужские, женские, смешанные и свободные палаты; цифры обозначают количество свободных кроватей; маленькие квадратики между палатами – общие ванные. Черные треугольники обозначают проблемы со здоровьем, а буквой «Н» обозначаются занятые кровати. В данном случае ОВНОС дополняется всплывающими подсказками, содержащими номер палаты, фамилии больных и дополнительную важную информацию о палате и больных. Верхняя часть окна содержит сводку о количестве палат, кроватей и медперсонале. У этого интерфейса весьма короткая кривая обучения. Однажды ознакомившись с ним, медсестры и менеджеры с первого взгляда оценивают положение дел в больничном корпусе*

о развитии города: если город достаточно развит, его архитектура более современна; чем он крупнее, тем крупнее и красивее соответствующая ему пиктограмма; если возникают беспорядки, над городом поднимается дым. Отдельные военизированные и гражданские единицы демонстрируют свой статус визуально с помощью маленьких идентификаторов здоровья и силы. Даже ландшафт в игре предлагает ОВНОС: пунктирные линии обозначают сферы влияния и перемещаются соответственно перемещению войск и росту городов. Топография местности меняется по мере того, как прокладываются дороги, вырубаются леса и добываются полезные ископаемые. Хотя в игре имеются диалоговые окна, большая часть информации, необходимой для понимания того, что происходит, доводится до пользователя без слов и диалоговых окон.

Представьте себе, что все объекты, информация о которых доступна в приложении или операционной системе, могут сообщать о своем состоянии таким образом: пиктограммы принтеров показывают, насколько процесс печати близок к завершению; пиктограммы дисков и съемных носителей информации отражают степень их заполненности; при выделении объекта для перетаскивания все объекты, способные его принять, визуально выражают свою готовность.

Подумайте об объектах внутри вашего приложения: какие у них есть атрибуты (особенно изменяющиеся динамически) и какая информация об их состоянии важна для пользователей? Выясните, как можно ее представить. Заметив и изучив это представление, пользователь будет понимать его с первого взгляда. (Кроме того, у пользователя должен быть способ запросить подробную информацию.) Поместите эту информацию в главное окно вашей программы в виде ОВНОС – и вы увидите, как много диалоговых окон можно исключить из повседневного использования!

Следует сделать одно важное замечание относительно обогащенной визуальной немодальной обратной связи: она не предназначена для новичков. Даже если вы добавите всплывающие подсказки с текстовым описанием всех визуальных индикаторов (что следует сделать), необходимо, чтобы пользователи вначале заметили их, а потом поняли, каков их смысл. Визуальная немодальная обратная связь обнаруживается пользователями не сразу. Обнаружив ее, они очень радуются, но до этого момента им требуется поддержка меню и диалоговых окон, чтобы находить нужные функции. Это означает, что ОВНОС, когда она используется в качестве замены уведомлений и предупреждений о серьезных проблемах, должна быть безусловно понятна и заметна для пользователей. Убедитесь, что этот тип обратной связи визуально выделяется на фоне менее критичной информационной обратной связи.

Звуковая обратная связь

В организациях, занимающихся обработкой данных, клерки часами сидят перед компьютерами и вводят данные. Эти пользователи погружены в исходные документы, а данные набирают, практически не глядя на экран. Если такой клерк допустит ошибку, ему следует сообщить об этом как через изображение, так и посредством звукового извещения. Тогда он сможет использовать слух для контроля своей работы, а его взгляд будет сосредоточен на документе.

Звуковая обратная связь, которую мы предлагаем, – это не те гудки, которые сопровождают сообщения об ошибках. Более того, это вообще не гудки. Звуковой индикатор, который мы имеем в виду, – *тишина*. Проблема с большинством звуковых индикаторов в настоящее время заключается в том, что пока еще преобладает мнение, будто отрицательная звуковая обратная связь предпочтительнее положительной.

Отрицательная звуковая обратная связь: объявление об ошибке пользователя

Люди часто выдвигают против звуковой обратной связи тот аргумент, что пользователям она не нравится. Пользователей раздражают звуки, раздающиеся из компьютера, и они не хотят, чтобы компьютер гудел на них. Microsoft и Apple пытались улучшить качество уведомляющих звуков, нанимая композиторов (включая легендарного Брайана Ино

(Brian Eno), который работал над Windows 95), но никакой самый теплый амбиент[1] в мире не изменит того факта, что эти звуки передают отрицательные и зачастую оскорбительные сообщения.

Выдача звукового сигнала при возникновении проблемы называется **отрицательной звуковой обратной связью**. Сообщения об ошибках в большинстве систем обычно сопровождаются пронзительными звуками, и поэтому звуковая обратная связь стала ассоциироваться именно с ошибками. Эти звуки публично объявляют об ошибках пользователя. Они сообщают окружающим, что вы сделали нечто отвратительно глупое. Эта идиома настолько ненавистна пользователям, что большинство разработчиков программных продуктов в настоящее время безоговорочно придерживаются мнения о том, что звуковая обратная связь не должна являться частью пользовательского интерфейса. Однако нет ничего более далекого от истины. Проблемы порождает отрицательный аспект обратной связи, а не звуковой ее аспект.

В отрицательной обратной связи есть несколько недостатков. Поскольку отрицательная обратная связь применяется в момент обнаружения проблемы, она естественным образом походит на сигнал тревоги. Сигналы тревоги специально делаются громкими, нестройными, вызывающими беспокойство. Предполагается, что они разбудят спящих, когда их дом загорится и когда их жизнь окажется в опасности. Тревожные сигналы подобны страховым случаям: мы надеемся, что никогда не столкнемся с ними. К сожалению, пользователи постоянно делают что-нибудь такое, с чем программы не способны разобраться, и эти действия стали нормальной частью общения пользователя и программы. Тревожным сигналам не место в этом повседневном общении. Ведь автомобильная сигнализация не срабатывает, когда мы перестраиваемся в другой ряд, не включив сигнал поворота. Наибольшего же порицания, вероятно, заслуживает то, что отрицательная звуковая обратная связь подразумевает, будто успех следует встречать тишиной. Но людям нравится знать, что они все делают правильно. Им *необходимо* знать о неудачах, но это не означает, что им нравится слышать о них. К системам с отрицательной обратной связью люди относятся, как правило, хуже, чем к системам с положительной.

Имея выбор между отсутствием звука и звуком отрицательной обратной связи, человек выберет первое. Но имея выбор между отсутствием звука и тихим и приятным звуком положительной обратной связи, большинство предпочтет обратную связь. Мы никогда не давали пользователям шанса попробовать высококачественную положительную звуковую обратную связь, и неудивительно, что люди ассоциируют звуки с неудачными интерфейсами.

---

[1] Амбиент – музыкальный жанр. – *Примеч. ред.*

Положительная звуковая обратная связь

Почти каждый объект и система в реальном мире обозначают звуком удачу, а не провал. Когда мы закрываем дверь, мы по звуку понимаем, что защелка сработала. Тишина означает, что дверь еще не заперта. Когда мы с кем-то разговариваем, то его реплики «Да» и «Угу» дают нам понять, что он нас, как минимум, слышит. Если собеседник молчит, у нас есть основания подозревать неладное. Когда мы поворачиваем ключ зажигания и ничего не слышим, мы понимаем, что возникла проблема. Когда мы, включив копировальный аппарат, не слышим характерного гула, мы понимаем, что он неисправен. Даже устройства, считающиеся бесшумными, издают какие-то звуки. Поворачивая кран газовой плиты, мы слышим шипение выходящего газа и звук возгорания. Электрические плиты по природе своей не столь дружелюбны, и пользоваться ими труднее из-за полного отсутствия звуков. О своем состоянии они сообщают световыми индикаторами.

Когда успешное применение инструмента подтверждается звуком, это называется **положительной звуковой обратной связью**. Наши программные инструменты в основном молчаливы. Все, что мы слышим, – это щелчки клавиш. Кстати! Это положительная звуковая обратная связь. Каждый раз, когда вы нажимаете на клавишу, вы слышите тихий, но позитивный звук. Производители клавиатур могли бы выпускать абсолютно бесшумные клавиатуры, но не стали, потому что нам нужна звуковая обратная связь, чтобы мы знали, насколько успешно работаем. Обратная связь не должна быть изощренной. Эти щелчки не несут в себе особой информации, но они несут ощущение постоянства. Если щелчка не было, мы понимаем, что плохо нажали на клавишу. Подлинная ценность положительной звуковой обратной связи в том, что отсутствие звука является крайне хорошим индикатором проблемы.

Эффективность положительной звуковой обратной связи связана с человеческой чувствительностью. Никому не нравится слышать о своих провалах. Диалоговые окна с сообщениями об ошибках говорят пользователю, что он совершил промах, а это отрицательная обратная связь. Тишина может сообщить пользователю об ошибке безо всяких окон. Она чрезвычайно эффективна, поскольку программе не приходится обижать пользователя, чтобы добиться своей цели.

Наши программы должны давать нам постоянные, почти незаметные звуковые подсказки, как это делают клавиатуры. Программы были бы гораздо дружественнее и проще в использовании, если бы издавали едва слышные, но легко идентифицируемые звуки, подтверждающие правильность действий пользователя. Приложение могло бы подавать слабый сигнал каждый раз, когда пользователь вводит допустимые данные, и издавать звуки, подтверждающие успешное заполнение форм. Если программа не может распознать, что вводит пользователь, ей следует сохранять молчание, тем самым тонко информируя пользователя о проблеме, и дать ему возможность исправить ввод, не застав-

ляя его испытывать смущение и не задевая самолюбие. Когда пользователь начинает перетаскивать пиктограмму, компьютер может издавать тихий звук скольжения предмета по поверхности. Если объект проходит над областями, готовыми его принять, этот факт можно отметить отрывистым звуком. Когда пользователь, наконец, отпустит кнопку мыши, его можно вознаградить ободряющим сигналом из динамиков – или встретить молчанием, если он отпустил объект в неподходящем месте.

Компьютерные игры, как правило, лидируют в применении не только визуальной обратной связи, но и положительной звуковой обратной связи. Mac OS X хорошо справляется с задачей предоставления негромкой положительной звуковой обратной связи при сохранении файлов и завершении операции перетаскивания. Конечно, громкость звуковой обратной связи должна быть подходящей для конкретной ситуации. Windows и Mac предлагают стандартный регулятор громкости, так что по крайней мере одно препятствие в предоставлении удачной звуковой обратной связи устранено, однако звуковая обратная связь не должна заглушать проигрываемую на компьютере музыку.

Обогащенная немодальная обратная связь – один из мощнейших инструментов в распоряжении проектировщиков взаимодействия. Замена надоедливых бесполезных диалоговых окон неброскими и мощными немодальными методами общения может превратить программу, которую пользователи презирают, в программу, которую они обожают. Только представьте себе, насколько более совершенными станут ваши приложения, если применить OBHOC и другие механизмы немодальной обратной связи!

26

Проектирование для различных потребностей

Как было сказано в первой части книги, персонажи и сценарии помогают нам сосредотачивать усилия проектирования на целях, поведении, потребностях и ментальных моделях реальных пользователей. В дополнение к этой общей фокусировке усилий по проектированию, которую обеспечивают персонажи, существуют также последовательные и доступные для обобщения шаблоны пользовательских потребностей, которые следует учитывать при проектировании продуктов. В данной главе мы рассмотрим некоторые стратегии удовлетворения этих распространенных потребностей.

Командные векторы и рабочие наборы

При классификации потребностей пользователей с различным уровнем подготовки весьма полезны понятия командных векторов и рабочих наборов. **Командные векторы** – это самостоятельные методы, которыми пользователи могут давать команды программе. Визуальные элементы для непосредственного манипулирования, раскрывающиеся и контекстные меню, элементы управления на панели инструментов и клавиатурные сокращения – все это примеры командных векторов.

Хороший пользовательский интерфейс предоставляет пользователю **множественные командные векторы**, в которых важные функции приложения представлены в разных формах, чтобы обеспечить пользователя параллельными возможностями управлять приложением: команды меню, кнопки на панели инструментов, клавиатурные сокращения и элементы непосредственного манипулирования. Такая избыточность позволяет пользователям с разными навыками и предпочте-

ниями выбирать способ управления программой сообразно своим способностям и склонностям.

Незамедлительные и обучающие векторы

Незамедлительные векторы – это элементы непосредственного манипулирования, такие как кнопки и панели инструментов. Между щелчком по кнопке и результатом выполнения функции нет задержки. Непосредственное манипулирование оказывает также прямое воздействие на информацию. Ни меню, ни диалоговые окна не обладают свойством незамедлительности – они требуют выполнения одного или нескольких промежуточных шагов.

Среди командных векторов есть такие, которые поддерживают новичков лучше остальных. Как правило, максимальную поддержку оказывают меню и диалоговые окна, и поэтому мы назовем их **обучающими векторами**. Начинающие пользователи выигрывают от педагогических свойств меню, когда им необходимо сориентироваться в незнакомой программе, а вечные середняки стремятся отказаться от меню и ищут более непосредственные эффективные векторы.

Рабочие наборы и персонажи

Поскольку в памяти каждого пользователя невольно откладываются часто используемые команды, вечные середняки запоминают некоторое подмножество команд, называемое **рабочим набором**. У каждого пользователя свой рабочий набор, хотя весьма вероятно, что он значительно пересекается с рабочими наборами других пользователей, сходным образом использующих приложение. В Excel, например, почти каждый пользователь будет вводить формулы и метки, задавать шрифты и печатать. В то же время рабочий набор Салли может включать работу с графикой, а рабочий набор Элиотта – работу со связанными электронными таблицами.

Хотя, строго говоря, не существует стандартного рабочего набора, покрывающего потребности всех пользователей, исследование и моделирование пользователей и способов, которыми они используют продукт, позволяет получить минимальное подмножество, в отношении которого проектировщики могут быть уверены, что оно требуется большинству пользователей. Такой **минимальный рабочий набор** можно определить методами целеориентированного проектирования – посредством сценариев, раскрывающих функциональные потребности ваших персонажей. Эти потребности напрямую отображаются в содержимое минимального рабочего набора.

Команды, составляющие рабочий набор любого пользователя, – это наиболее часто используемые команды. Пользователь хочет, чтобы доступ к ним был особенно быстрым и простым. Так что проектировщик обязан, как минимум, предоставить незамедлительные команд-

ные векторы для минимального рабочего набора наиболее вероятных пользователей данного приложения.

Хотя минимальный рабочий набор для программы почти наверняка является частью полного рабочего набора любого пользователя, индивидуальные предпочтения конкретного пользователя и условия его работы диктуют то, какие дополнительные функциональные возможности должны быть предусмотрены. Даже созданный на заказ программный продукт для корпоративного использования может предложить набор функций, из которого каждый пользователь выберет для себя подходящие. Это означает, что разработчик, предоставив прямой доступ к минимальному рабочему набору, должен также предоставить способы включения внутрь незамедлительных векторов других команд. Аналогичным образом, незамедлительные команды требуют наличия обучающих векторов, помогающих в освоении интерфейса новичкам. Это подразумевает, что множественные командные векторы следует предусмотреть для большинства функций в интерфейсе.

Существует исключение из правила множественных векторов. Опасные команды (такие как Стереть все, Отказаться от изменений и другие) не должны иметь легко доступных множественных командных векторов. Их следует защищать посредством меню и диалоговых окон в соответствии с принципом проектирования из главы 10: *прячьте рычаги катапультирования*.

Перевод новичков в середняки

Дональд Норман предлагает еще один полезный взгляд на командные векторы. В своей книге «The Design of Everyday Things»[1] (Norman, 1989) Норман использует термины **информация в окружении** и **информация в голове** для обозначения разных способов, которыми пользователи обращаются к информации. Говоря об информации в окружении, Норман подразумевает ситуации, когда объем информации, достаточный для выполнения действия, доступен в окружении и интерфейсе. Киоск с картой центра города является примером информации в окружении: нам не нужно помнить, где находится та или иная достопримечательность, поскольку ее легко найти с помощью карты. Противоположность этому – информация в голове, то есть знания, которые мы приобрели в результате обучения или просто запомнили. Они как короткая дорога через задворки, не обозначенная ни на одной карте. Информацией в голове проще и быстрее пользоваться, чем информацией в окружении, но вы несете ответственность за то, чтобы помнить ее и своевременно обновлять. Доступ к информации в окружении медленнее и сложнее, но этот способ очень надежен.

---

[1] Русский перевод выходил под двумя названиями: Д. Норман «Дизайн промышленных товаров». – Пер. с англ. – М.: Вильямс, 2008; Д. Норман «Дизайн привычных вещей». – Пер. с англ. – М.: Вильямс, 2006. – *Примеч. ред.*

Векторы окружения и ментальные векторы

Обучающий вектор неизбежно заполнен информацией из окружения, поэтому он является **вектором окружения**. И наоборот: клавиатурные сокращения образуют **ментальный вектор**, поскольку работа с ними требует запоминания функций и их клавиатурных эквивалентов, то есть задействует информацию в голове. Векторы окружения необходимы новичкам, а также опытным пользователям, когда те обращаются к сложным или редко вызываемым функциям. Ментальные векторы нужны середнякам и еще сильнее нужны опытным пользователям.

Например, после того как вы переехали на новое место жительства, вам, вероятно, потребуется карта – вектор окружения. Пожив на новом месте пару дней, вы откажетесь от карты, потому что запомнили дорогу домой – это ментальный вектор. С другой стороны, хотя вы хорошо знаете свой дом, если вам потребуется настроить нагреватель воды, вы будете вынуждены прочитать инструкцию к нему – вектор окружения (вы ведь не изучили ее наизусть, когда въехали в этот дом).

Наши отношения с программным продуктом строятся аналогично. Мы легко запоминаем функциональные возможности и команды, которыми пользуемся часто, и не запоминаем подробности команд, которые бывают нам нужны редко. Это означает, что любой часто используемый вектор автоматически становится кандидатом в ментальный вектор. Например, при ежедневном использовании мы перестаем вчитываться в пункты меню, а лишь распознаем образы: «Открыть второе меню и выбрать самый нижний пункт в предпоследнем разделе». Распознавание образов человеческий мозг выполняет быстрее чтения. Мы читаем только для того, чтобы убедиться в правильности выбора.

Векторы запоминания

Новички с готовностью пользуются векторами окружения, однако по мере приобретения опыта и перехода в категорию вечных середняков они формируют собственные рабочие наборы – и (обучающие) векторы окружения начинают их утомлять. В качестве содержимого рабочих наборов пользователи предпочитают более прямые ментальные векторы. Это естественное и уместное желание, и если мы претендуем на создание удобного программного продукта, то должны удовлетворить его. Решение состоит из двух частей. Во-первых, мы должны создать ментальный вектор, дополнительный к вектору окружения, и, во-вторых, мы должны предоставить пользователю путь, пройдя по которому, он найдет ментальный вектор, соответствующий любому вектору окружения. Этот путь сам является вектором и называется **вектором запоминания**.

Существует несколько способов создавать векторы запоминания для пользователей. Наименее эффективный – упомянуть этот вектор только в руководстве пользователя. Чуть лучший, но также неэффектив-

ный метод состоит в упоминании вектора в справочной системе приложения. Эти методы перекладывают бремя поиска вектора запоминания на пользователя. Кроме того, они заставляют пользователя самостоятельно приходить к мысли, что ему нужно найти этот вектор.

Самые лучшие векторы запоминания встроены непосредственно в интерфейс или, по крайней мере, доступны в интерфейсе приложения посредством его собственных векторов окружения. Последнее может быть с минимальными затратами реализовано с помощью добавления пункта Клавиатурные сокращения в меню Справка. Этот пункт открывает раздел справочной системы, описывающий имеющиеся клавиатурные сокращения. Преимущество такого метода в том, что он очевиден, а следовательно, подходит для обучения. Новички видят, что существуют множественные командные векторы и что для их изучения предлагается легкодоступный источник информации. Пункт меню Клавиатурные сокращения должен быть в любой программе.

Предлагайте информацию о клавиатурных сокращениях в меню Справка.

Встраивание векторов запоминания непосредственно в основной интерфейс реализуется проще, чем может показаться. В меню большинства программ уже присутствуют два таких вектора. По стандартам Microsoft, типичное приложение Windows имеет два клавиатурных ментальных вектора – мнемоники и клавиатурные сокращения. Например, в Microsoft Word мнемоникой для команды сохранения является <Alt>+<F>+<S>. Вектор запоминания для этой мнемоники заключается в подчеркивании букв F и S в названии меню и пункта меню соответственно. Клавиатурным сокращением сохранения является <Ctrl>+<S>, оно явно обозначено справа от соответствующего пункта меню и действует как вектор запоминания.[1]

[1] Отдельной задачей для проектировщика является организация поведения мнемоник и клавиатурных сокращений при создании локальных версий приложения. Поскольку меню при локализации приложения подлежат переводу, то и мнемоники неизбежно меняются, при этом не всегда удается сохранить интуитивность исходной мнемоники. Так, описанная выше мнемоника для команды сохранения файла в русской версии Microsoft Word сохраняет свою интуитивность: <Alt>+<Ф>+<С>, а вот мнемоникой для доступа к команде меню Вставка является буква а, что совершенно неинтуитивно (сравните с оригинальной мнемоникой I для команды меню Insert). Клавиатурные сокращения по сложившейся традиции обычно остаются без изменений. В некоторых программах можно обнаружить решение, которое трудно признать удачным: буквы латинского алфавита в записи клавиатурной комбинации справа от команды меню заменяются кириллическими буквами, находящимися на тех же клавишах, причем иногда это зависит от того, какая раскладка клавиатуры активна в данный момент. *(Продолжение на след. стр.)*

Ни один из этих векторов не мешает работе новичка. Начинающий пользователь, возможно, даже не заметит их наличия, пока не станет пользователем среднего уровня. В конце концов он обратит внимание на эти визуальные подсказки и заинтересуется их смыслом. Большинство разумных людей (то есть большинство пользователей) поймут смысл клавиатурных сокращений без посторонней помощи. Разобраться с мнемоникой чуть сложнее, но если пользователь узнает о назначении клавиш <Alt> (случайно или из какого-то источника), ему не составит труда запомнить эту идиому и применять ее, когда потребуется.

Как помнят читатели из главы 23, **кнопки-значки** – отличный прием, задействующий маленькие пиктограммы для создания векторов запоминания, позволяющих пользователю осуществить переход от меню к панели инструментов. Пиктограмма, идентифицирующая некоторую функцию или возможность, должна присутствовать на каждом элементе пользовательского интерфейса, имеющем отношение к этой функции, – на пункте меню, на кнопке, в диалоговом окне, при каждом упоминании в тексте справки и тексте печатной документации. Вектор запоминания, образованный наглядными символами, встречающимися в интерфейсе, является самым эффективным методом, хотя он недостаточно широко эксплуатируется в отрасли в целом.

Персонализация и настройка

Проектировщикам взаимодействия часто приходится принимать решение относительно того, давать ли пользователю настраивать продукт сообразно своим предпочтениям. Здесь легко оказаться в тупике, пытаясь угодить потребностям тех пользователей, которым требуется такая настройка, и при этом не породить навигационных проблем, связанных с перемещением или исчезновением из интерфейса знакомых пользователю элементов. Решение таково: рассмотреть этот вопрос в ином свете.

---

[1] *(Продолжение)* Не меняя сути происходящего (физически нажимаются те же клавиши), такое решение только запутывает пользователя: сочетание <Ctrl>+<P> для команды меню Печать не является для русскоязычного пользователя вполне интуитивным, но все же обладает большей мнемоничностью по сравнению с <Ctrl>+<З>. Возможно, наиболее близким к идеальному решению было бы наличие двух комплектов клавиатурных сокращений (оригинального и локализованного, в котором команда печати вызывалась бы, скажем, сочетанием <Ctrl>+<П>) с возможностью переключения между ними. Нелишним будет также напомнить разработчикам, что мнемоники и клавиатурные сокращения должны работать (и работать единообразно) вне зависимости от текущей клавиатурной раскладки (это правило нарушалось, например, в ранних версиях Adobe Photoshop, отказывавшихся реагировать на клавиатурные команды при активной кириллической раскладке). – *Примеч. науч. ред.*

Люди любят изменять окружение по своему вкусу. Даже новички (не говоря уж о вечных середняках) любят ставить на программу свое клеймо, настраивая ее так, чтобы она выглядела и вела себя в соответствии с их предпочтениями и личным вкусом. Люди делают это по той же причине, по какой украшают свое рабочее место фотографиями жены и детей, комнатными растениями, репродукциями картин, афоризмами или комиксами про Дилберта.

Декорирование стабильных объектов (стен) придает им индивидуальность, оставляя их при этом на месте. Вы можете отличить конкретный коридор от всех других потому, что именно на его стене висит репродукция Эшера. **Персонализацией** мы будем называть украшение стабильных объектов.

Персонализация делает наши рабочие места более симпатичными и узнаваемыми. Она делает их более человечными и приятными для работы. То же справедливо в отношении программных продуктов: возможность украшать приложение радует пользователя и становится для него полезным средством навигации.

С другой стороны, от *перемещения* стабильных объектов может пострадать навигация. Если на выходных в вашу компанию явятся рабочие и переставят все рабочие отсеки, то в понедельник вам будет нелегко найти свое рабочее место – и даже комиксы про Дилберта тут не помогут. (Важность стабильных объектов для навигации обсуждается в главе 11.)

Противоречие? Не совсем. Украшение стабильных объектов способствует навигации, а перемещение их – препятствует. **Настройкой** называется перемещение, добавление и удаление стабильных объектов.

Настройка желательна для более опытных пользователей. Вечные середняки, определившись со своим рабочим набором функций, захотят настроить интерфейс таким образом, чтобы эти функции было легче находить и выполнять. Они также захотят настроить и саму программу, чтобы упростить и ускорить работу, но во всех случаях настройка будет носить умеренный характер.

Экспертам настройка необходима. Они уже не нуждаются в традиционных средствах навигации, поскольку очень хорошо знакомы с продуктом. Эксперты могут работать с программой по нескольку часов в день, и, более того, она может быть основным приложением, необходимым им для работы.

Перемещение элементов управления на панели инструментов – это разновидность персонализации. Но три крайних левых инструмента на панелях инструментов многих программ, соответствующие операциям Создать файл, Открыть файл и Сохранить файл, являются сейчас настолько распространенными, что их можно отнести к стабильным объектам. Пользователь, который перемещает их, *настраивает* программу в той

же степени, в какой персонализирует ее. Таким образом, граница между настройкой и персонализацией является не вполне четкой.

Изменение цвета объектов на экране – это, несомненно, задача из области персонализации. Операционная система Windows в этом смысле всегда шла навстречу пользователю, позволяя ему независимо изменять цвета элементов оконного интерфейса, включая цвет и рисунок рабочего стола. Windows предоставляет также пользователям *полезную* возможность сменить системный шрифт. Персонализация **идиосинкратически**[1] **модальна** (см. следующий раздел): одним людям она нравится, другим – нет. Вы обязаны учесть пожелания обеих категорий.

Инструменты персонализации должны быть простыми и удобными в обращении, давая пользователю возможность предварительного просмотра результатов его действий. Кроме того, действия персонализации должны легко отменяться. Диалоговое окно, позволяющее пользователям изменять цвет элементов управления, обязано предлагать и функцию, возвращающую все настройки к исходным значениям.

Большинство конечных пользователей не будут громко протестовать, если им не предоставят возможность настраивать программу, которая и так хорошо справляется со своей работой. Некоторые действительно опытные пользователи, возможно, почувствуют себя обделенными, но они, тем не менее, будут работать с программой и ценить ее по достоинству, если она работает так, как они того ожидают. Однако в некоторых случаях гибкость абсолютно необходима. Если вы проектируете продукт для быстро эволюционирующих рабочих процессов, становится чрезвычайно важным, чтобы этот продукт мог эволюционировать с такой же скоростью, как среда его применения.

Возможность настройки высоко ценят также IT-менеджеры. Она позволяет им ненавязчиво подталкивать корпоративных пользователей к использованию общих методов работы. Эти администраторы ценят возможность создавать макрокоманды и функции в меню и на панели инструментов, заставляющие стандартный программный продукт работать в соответствии с принятыми в компании процессами, методами и стандартами. Многие IT-менеджеры в своем решении о покупке программного продукта исходят из степени настраиваемости этого продукта. Если они покупают десять или двадцать тысяч лицензий на программу, то по праву считают, что должны иметь возможность адаптировать ее к своему стилю работы. Именно поэтому, а не по чьей-то прихоти приложения Microsoft Office входят в число наиболее гибко настраиваемых коробочных программных продуктов на рынке.

---

[1] *Идиосинкразия* – болезненная реакция, непереносимость, возникающая у некоторых людей на раздражители, которые у большинства других людей подобных явлений не вызывают. В психологии термин используется метафорически для обозначения психологической несовместимости, непереносимости некоторыми людьми друг друга. – *Примеч. науч. ред.*

Идиосинкратически модальное поведение

Пользовательское тестирование неоднократно демонстрировало, как аудитория расходится во взглядах на эффективность той или иной идиомы, разделяясь на две примерно равные части. Половина пользователей явно предпочитает одну идиому, а вторая половина – другую. Такое четкое разделение аудитории по предпочтениям на две или несколько крупных групп указывает, что эти предпочтения **идиосинкратически модальны**.

Фирмы-разработчики также не демонстрируют единодушия в вопросах такого рода. Один лагерь внутри компании может выступать за использование в интерфейсе меню, другой – за использование кнопок-значков. Они пререкаются и спорят по поводу относительных достоинств этих методов, в то время как ответ очевиден: применяйте оба!

Когда пользовательская аудитория разделена предпочтениями относительно идиом, проектировщики *обязаны* предложить обе идиомы. Необходимо удовлетворить обе группы. Нельзя удовлетворить одну половину аудитории, оставив в негодовании другую, – и при этом совершенно неважно, к какой группе относите себя вы или ваши разработчики.

Реализация системы меню в Windows – блестящий пример того, как следует угождать идиосинкратически модальным желаниям. Одним нравятся меню в стиле раннего Macintosh: вы щелкаете мышью по строке меню, чтобы раскрыть нужное меню, а затем, не отпуская кнопку, перемещаете указатель мыши вниз и отпускаете кнопку только на нужном пункте. Другие находят эту процедуру неудобной и предпочитают выполнять ее, не удерживая кнопку мыши нажатой. Windows удовлетворяет такие пожелания, позволяя пользователю щелкнуть в строке меню и отпустить кнопку, чтобы нужное меню открылось. После этого пользователь может перемещать мышь (с отпущенной кнопкой) к нужному пункту. Щелкнув по пункту и отпустив кнопку, пользователь выбирает пункт и закрывает меню. При этом у пользователя по-прежнему есть возможность выбирать пункт меню щелчком и перетаскиванием. Великолепие этих идиом – в том, что они мирно сосуществуют. Любой пользователь может свободно применять обе идиомы или принципиально придерживаться только одной. Программу не требуется перенастраивать. Не нужно указывать индивидуальные предпочтения: все работает и так.

В Windows 95 корпорация Microsoft добавила в поведение меню третью идиосинкратически модальную идиому. Пользователь может щелкнуть кнопкой и отпустить ее, как и прежде, но теперь он может перемещать указатель мыши вдоль строки меню – и разделы меню будут поочередно открываться. Удивительно, но все три идиомы по-прежнему реализованы бесшовно. В настоящее время Mac тоже поддерживает все три идиомы.

Локализация и глобализация

Создание приложений для различных языков и культур ставит перед проектировщиками ряд специфических проблем. И здесь снова хорошим ориентиром могут служить командные векторы.

Незамедлительные векторы, такие как непосредственное манипулирование и кнопки-значки на панели инструментов, являются идиоматическими (глава 20) и визуальными, а не текстовыми, поэтому они без особых сложностей поддаются глобализации. Конечно, проектировщикам следует проделать подготовительную работу, чтобы убедиться, что выбранные для этих идиом цвета и символы не имеют в других культурах какого-то особого смысла, не заложенного разработчиком. (Например, в Японии крестик в окошке флажка будет интерпретирован скорее как *отсутствие* выделения, чем как выделение.) Однако в целом неметафорические идиомы в глобализованных интерфейсах вполне безопасны.

Обучающие векторы, такие как пункты меню, подписи к полям ввода, всплывающие подсказки и инструкции, зависят от языка и, таким образом, являются объектом локализации, осуществляемой посредством перевода. При создании интерфейсов, подлежащих локализации, следует принять во внимание следующие моменты:

- В некоторых языках слова и фразы длиннее, чем в других (например, текстовые метки, написанные по-немецки, в среднем значительно длиннее написанных по-английски).
- В некоторых языках, особенно азиатских, может вызвать сложности алфавитная сортировка.
- Формат вывода даты и соглашения о применении 12- и 24-часовых систем счисления времени в разных странах различны.
- Символы, отделяющие десятичную часть от целой в числах и триады в денежных суммах, могут быть разными. (В некоторых странах использование точки и запятой в этих случаях противоположно принятому в США.)
- В ряде стран принято нумеровать недели в году (например, в середине декабря идет 50-я неделя). Кроме того, не во всех странах придерживаются григорианского календаря.

К переводу пунктов меню и сообщений в диалоговых окнах следует подходить целостно. Важно убедиться, что в результате перевода интерфейсы не утратили связности. Переведенные в отрыве от контекста пункты меню и метки могут вызывать путаницу, когда окажутся рядом с другими независимо переведенными элементами интерфейса. Семантику интерфейса следует сохранять на всех уровнях.

Коллекции и шаблоны

Не каждый пользователь приложения, имеющего дело с документами, способен создавать документы с нуля. Однако большинство программ предлагают пользователям лишь простейшие инструменты вроде молотка, пилы или стамески. Некоторым пользователям этого достаточно, но другим нужно больше. Они предпочли бы получить уже сколоченный стул или стол, который можно отшлифовать и покрыть лаком.

В качестве примера рассмотрим программу, позволяющую формировать собственную газету на основе информации из Интернета. Некоторые пользователи с удовольствием поместят в центре первой страницы новости спорта. Однако большинство, вероятно, предпочтет более традиционный подход с событиями в мире на первой странице и спортом на последней. Но даже эти традиционалисты не откажутся от возможности разместить в газете местные новости или сообщения о том, что интересно лично им. У них должна быть возможность взять полуфабрикат газеты и слегка изменить его, чтобы получить свою собственную версию. Конструирование целой газеты с чистого листа будет скучной задачей для всех, кроме тех, в ком спит журналист.

Иными словами, в любом приложении необходимо позволить пользователям выбирать дизайн или исходную структуру документа из коллекции шаблонов, когда у них нет необходимости или желания создавать документ с нуля.

Предложите пользователям коллекции шаблонов, готовых к применению.

Некоторые программы уже предлагают коллекции готовых шаблонов, но многим приложениям еще только предстоит это сделать. Пустые листы пугают большинство людей, а пользователи не должны вынужденно терпеть то, что им не нравится. Коллекции базовых шаблонов являются удачным решением.

Справка

Как и бумажная документация, справочная система программы предназначена для вечных середняков. Хорошая оперативная справка крайне важна, но она не должна становиться костылем для продукта. Качественно спроектированный интерфейс должен значительно сократить потребность пользователей в справке.

Сложную программу с большим количеством функций и возможностей необходимо сопроводить справочным документом, с помощью которого пользователи, желающие расширить свое понимание продукта, смогут найти конкретные ответы на свои вопросы. Этим документом может быть как печатное руководство, так и оперативная справка. На-

печатанное руководство пользователя удобно, дружелюбно, его можно листать и брать с собой. Электронная справочная система позволяет легко выполнять поиск, чуть менее удобна, очень мало весит и мало стоит.

Указатель

Никто не читает руководство пользователя как роман, так что высококлассный и удобный справочник – это прежде всего качественные инструменты поиска информации в нем, и в первую очередь – указатель. В конце печатного руководства имеется предметный указатель, с которым приходится работать вручную. Оперативная справка предлагает автоматические средства поиска по указателю.

Мы подозреваем, что в очень немногих приложениях указатель оперативной справки был создан профессионалом в этой области. Сколько бы ни было записей в указателе вашей программы, их количество, скорее всего, можно удвоить. Более того, указатель следует составлять исходя из функциональных возможностей программы, а не на основе текста справки. Это нелегко, поскольку требует участия высококвалифицированного составителя указателей, хорошо знакомого со всеми функциями программы. Иногда легче переработать и улучшить интерфейс, чем создать действительно хороший указатель.

Список записей указателя едва ли не более важен, чем текст самих записей. Пользователь скорее простит плохо написанный элемент указателя, чем отсутствие нужного элемента. Для каждой из тем указатель обязан содержать как можно больше синонимов. Приготовьтесь к тому, что указатель получится огромным. Пользователь, пытающийся разрешить свою проблему, будет думать: «Как мне покрасить эту ячейку в черный цвет?», а не «Как мне установить значение *затенения* этой ячейки равным 100%?» Если запись указателя озаглавлена «затенение», пользователь ее не найдет. Чем больше вы думаете о целях пользователя, тем лучше ваш указатель будет отражать мысли, приходящие в голову пользователя, которому требуется что-то найти. Отличный образец указателя можно найти в книге Ирмы С. Ромбаур и Мэрион Ромбаур Беккер (Irma S. Rombaur & Marion Rombaur Becker) «The Joy of Cooking» (Rombaur and Becker, 1975). Нам не приходилось встречать более полного и надежного указателя.

Клавиатурные сокращения и обзоры

Практически ни одна оперативная справка не включает в себя раздел, посвященный клавиатурным сокращениям. Это пункт меню Справка, выбор которого выводит на экран перечень клавиатурных сочетаний, соответствующих различным функциям и инструментам программы. Такой перечень является необходимой составляющей любой оперативной справки, поскольку предоставляет вечным середнякам то, в чем

они больше всего нуждаются, – доступ к функциям. Инструменты и команды нужны им больше, чем подробные инструкции.

Другим недостающим ингредиентом оперативной справки являются **обзоры**. Пользователи хотят знать, как работает команда Создать макрос, а справочная система выдает им бесполезные сведения о том, что это функция, которая позволяет ввести в систему макрокоманду. Что действительно требуется – так это информация о возможностях этой команды, о ее действии, о плюсах и минусах, о том, почему этой командой следует пользоваться, причем информация эта должна быть представлена как в абсолютных формулировках, так и в сравнении с аналогичными конкурирующими продуктами.

Не для новичков

Многие справочные системы созданы исходя из идеи, что они должны оказывать помощь новичкам. Однако их задача не в этом. Начинающих важно снабдить руководством по «быстрому старту», однако оперативную справку следует ориентировать на пользователей, которые уже успешно работают с продуктом, но хотят расширить свои горизонты, то есть на вечных середняков.

Немодальная и интерактивная справка

Всплывающие подсказки относятся к немодальной оперативной справке и являются невероятно эффективным средством помощи пользователю. С другой стороны, стандартные справочные системы реализуются в виде самостоятельных программ, окна которых не намного меньше окна основной программы. Если бы вы попросили человека пояснить работу с приложением, он стал бы указывать пальцем на объекты на экране, дополняя свой рассказ. Отдельная программа, перекрывающая своим окном основную, на это не способна.

Мастера

Мастера – это идиома, рожденная компанией Microsoft. Они быстро завоевали популярность среди программистов и проектировщиков интерфейсов. Мастер пытается гарантировать пользователю успешное выполнение задачи, проводя его через последовательность диалоговых окон. Эти окна служат множественным вектором сложной процедуры, которая «обычно» используется для решения данной задачи. Например, специальный мастер в PowerPoint помогает пользователю создать презентацию.

Мастера нравятся программистам, поскольку позволяют обращаться с пользователем как с периферийным устройством. Каждое диалоговое окно мастера задает пользователю один-два вопроса – и в конечном итоге приложение делает то, что требовалось. Мастер наглядно демонстрирует тактику допроса пользователя, которой придерживается про-

грамма, нарушая принцип проектирования из главы 10: *не задавайте вопросы – предоставляйте выбор*.

Мастер создается как пошаговая процедура, а не как осмысленный диалог между пользователем и программой. Пользователь здесь находится в роли дирижера оркестра роботов: он размахивает палочкой, задавая темп, но не располагает никаким другим средством повлиять на происходящее. Неудивительно, что мастера быстро деградировали до уровня окон с подтверждениями. Пользователь знает, что ему достаточно щелкать по кнопке Далее, не анализируя события. Хуже всего то, что мастера часто задают неясные, загадочные вопросы. Пользователь, который не знает, что такое IP-адрес, не сможет его указать не только в диалоге, но и в мастере.

Более удачной альтернативой мастеру была бы простая функция-автомат, которая, не задавая пользователю лишних вопросов, делала бы свое дело. Например, если надо создать презентацию, функция должна создать ее и затем позволить пользователю отредактировать презентацию с помощью стандартных средств. Практикуемая типичным мастером тактика допроса не является ни дружелюбной, ни вселяющей в пользователя уверенность, ни приносящей сколь-нибудь заметную пользу. Часто мастер не удосуживается даже объяснить пользователю, что вообще происходит.

Мастера создавались с целью улучшения пользовательских интерфейсов, но во многих случаях эффект оказался прямо противоположным. Они выдают программистам лицензию на создание интерфейса для сложных функциональных возможностей на основе голой модели реализации, вселяя в них надежду, что «потом все можно будет упростить с помощью мастера». Это в изрядной степени напоминает стандартный прием перекладывания ответственности на пользователей: «Мы обязательно задокументируем это в руководстве пользователя».

«Интеллектуальные» агенты

Возможно, не следует долго распространяться о Скрепыше и его родственниках, поскольку сама корпорация Microsoft пошла против своего творения, продвигая Windows XP (при этом, однако, Скрепыш там *по-прежнему* существует). Скрепыш – это пережиток исследования, проведенного Microsoft в процессе создания MS Bob – «интуитивного», близкого к реальному миру, насыщенного метафорами интерфейса, подобного интерфейсу программы Magic Cap от фирмы General Magic (см. главу 13). Пространство MS Bob населяли антропоморфные анимированные персонажи, которые вступали в разговор с пользователем, пытаясь оказать ему помощь. Это был один из самых впечатляющих провалов Microsoft в области интерфейсов. Скрепыш – потомок тех агентов-помощников, и он вызывает у пользователей ничуть не меньшее раздражение.

Корень проблемы с «интеллектуальными» агентами в том, что, задействуя анимированный антропоморфизм, программа повышает уровень пользовательских ожиданий в отношении разумности этих агентов. Если они не способны удовлетворить ожидания пользователей, пользователи быстро начинают злиться – как злятся покупатели на продавца, который заявляет, что разбирается в товарах, но после двух-трех простых вопросов оказывается полным профаном.

Подобные создания быстро всем надоели и превратились в отвлекающий фактор. Пользователи Microsoft Office садятся за компьютер, чтобы работать, а не развлекаться, наблюдая за ужимками и оплошностями справочной системы. В большинстве случаев пользователю требуются более конкретные, менее отвлекающие от работы и заслуживающие большего доверия методы оказания помощи.

Послесловие: несколько слов о сотрудничестве

Мы считаем, что целеориентированный метод состоит из четырех частей: процессы, шаблоны, принципы и практика. Эта книга посвящена в основном первым трем составляющим. В заключение мы хотели бы поделиться некоторыми мыслями относительно *практики* проектирования взаимодействия.

Проектирование взаимодействия – занятие, как правило, сложное и малоупорядоченное (что не мешает нам его любить). Проектировщиков часто просят помочь создать образ или определение чего-то, что раньше никогда не существовало, настаивая при этом на применении новой технологии и крайне сжатых сроках. Им приходится глубоко изучать предметные области, искать баланс противоречивых приоритетов, вникать в возможности и ограничения предложенной им технологии, а также бизнес-контекст проекта в целом.

От таких задач голова идет кругом – и это чувство заставило нас создать подход, отличающийся сугубой методичностью. Работая в соответствии с процессами, описанными в первой части книги, мы знаем, что в нужный момент у нас под рукой окажется вся необходимая информация, которая даст ответы на правильно поставленные вопросы, ощущение предсказуемости и возможность проследить источники всех решений вплоть до требований, сценариев, персонажей и результатов исследований. Шаблоны и принципы полезны, поскольку помогают нам не тратить силы на изобретение колеса.

Однако этого мало. Процессы, шаблоны и принципы являются необходимым для успешного проектирования продуктов условием, но не являются условием достаточным. Наличие этих инструментов не избавляет проектировщиков от необходимости проявлять изобретательность, уметь представить себе новые реалии, анализировать свой опыт и использовать здравый смысл, чтобы отличать хорошие решения от плохих. Не существует готовых рецептов творчества, гарантирующих успешность найденных проектировочных решений. И даже если вы уверены, что концепция хороша, ее качественное воплощение потребует значительного объема тяжелой работы, усердия и навыков. Один из самых сложных и хаотических (но в то же время и самых приятных)

аспектов этой тяжелой работы – сотрудничество с другими членами команды, создающей продукт, включая людей, занятых деловой стороной вопроса.

Как уже говорилось, для проектирования взаимодействия необходимы сведения от тех, кто принимает бизнес-решения, от маркетологов, технологов, аналитиков, от (возможно, огромной) армии разработчиков, тестировщиков, сотрудников службы поддержки и установщиков, а также от многих других людей. Вместе с тем процесс проектирования взаимодействия оказывает влияние на всех этих людей. Если этот процесс протекает в вакууме, создающая продукт команда не будет иметь четкого представления о направлении движения, и мудрость и опыт ее участников не будут использованы на благо проектирования. Как мы писали в главе 4, проектировщикам следует стремиться понять цели, идеи, ограничения различных участников проекта, используя интервью с заинтересованными лицами на ранних стадиях этапа исследований. Кроме того, ключевых заинтересованных лиц необходимо вовлекать в процесс проектирования на всех его этапах, чтобы обеспечить прозрачность процесса и возможности для обмена мнениями, а также получить поддержку при принятии проектных решений.

Мы выделяем три важные группы, с которыми проектировщикам взаимодействия следует наладить особенно интенсивное взаимодействие. Во-первых, это люди, которые принимают решения в деловой сфере и, как говорилось во введении, в конечном счете, отвечают за прибыльность и успех продукта; от них обычно зависит финансирование процесса проектирования. Важно работать в тесном контакте с этими людьми в момент запуска проекта, чтобы уяснить их видение продукта и определить задачи этапа исследования пользовательской аудитории, а затем на этапах выработки требований и проектирования инфраструктуры (см. главы 6 и 7), чтобы выработать и закрепить согласованное видение продукта. Важно также вовлечь людей, принимающих бизнес-решения, в обсуждения на этапе детализации, поскольку обсуждаемые варианты решений могут повлиять на бюджет и сроки разработки.

Во-вторых, проектировщики взаимодействия должны плотно сотрудничать с программистами и другими технологами (например, с инженерами-механиками и электротехниками, если речь идет о разработке физического продукта). Без талантов и способностей этих людей даже лучшие решения проектирования ничего не будут стоить. Цель такого сотрудничества – обеспечить идеальную стыковку проектирования и реализации, убедившись, что проектные решения учитывают технические и стоимостные ограничения, задействуют возможности, предоставляемые технологией, и в полной мере донесены до программистов. Кроме того, опыт технолога часто позволяет ему указать на какие-либо новые возможности, о которых проектировщики не имели представления прежде.

Наконец, проектировщики взаимодействия должны тесно сотрудничать со всеми прочими участниками проекта, действия которых влияют на общий опыт взаимодействия пользователя с продуктом. В зависимости от проекта речь может идти о разработчиках стратегии, исследователях пользовательской аудитории и рынка, промышленных дизайнерах, графических дизайнерах, технических писателях, дизайнерах упаковки, а иногда даже о людях, проектирующих общую инфраструктуру продажи продукта. Цель этого сотрудничества – обеспечить гармоничное сочетание всех аспектов пользовательского опыта и гарантировать, что работа не будет направлена на реализацию противоречивых целей и не будет вестись на разных языках проектирования – а значит, не запутает пользователей и не затруднит восприятие продукта.

Вовлекать в работу эти три важные группы лучше всего следующими двумя способами: во время формальных совещаний, соответствующих завершению фаз процесса, и на частых неформальных рабочих встречах, где вырабатываются, изучаются и оцениваются новые идеи. Рабочие совещания приобретают особое значение для второй группы (технологов) после начала кристаллизации проектных решений, а для третьей группы становятся чрезвычайно важными на ранних стадиях генерации идей и на последующих стадиях процесса. Мы кратко обсуждали взаимоотношения между проектировщиками взаимодействия, промышленными дизайнерами и графическими дизайнерами в главах 4, 5, 6 и 7.

Когда-то мы полагали, что этап проектирования следует завершить до того, как начнется написание кода. Опыт научил нас, что такой подход плохо сочетается с реальностью из-за агрессивных сроков разработки, а также, что более важно, из-за необходимости доказывать жизнеспособность проектных решений. Однако мы по-прежнему считаем, что *все аспекты продукта должны быть спроектированы до начала разработки.* Этим мы хотим сказать, что проектирование пользовательского опыта, опирающееся на собранные данные, обязано предшествовать разработке, но при этом даже в условиях жесткого итеративного процесса всегда существует возможность упорядочить работу таким образом, чтобы разработчики уже прорабатывали отдельные аспекты продукта в то время, когда проектировщики еще трудятся над другими его аспектами. К сожалению, такой подход оказывается гораздо менее упорядоченным, чем четкое разделение проектирования и разработки со строго последовательным их выполнением. Мы обнаружили, что для реализации этого подхода в обязательном порядке необходимо непрерывное взаимодействие технических специалистов, проектировщиков и бизнесменов в области расстановки приоритетов на всем протяжении проектирования и разработки.

Успех продукта, отвечающего человеческим потребностям, требует тщательной координации усилий большого количества людей. Мы

пришли к выводу, что проектировщики взаимодействия, чтобы достичь желаемого результата, должны в конечном итоге взять на себя значительную ответственность за тонкую балансировку многочисленных сил, влияющих на процесс создания продукта. Надеемся, что представленные в этой книге инструменты помогут вам создавать великолепные цифровые продукты, действительно удовлетворяющие пользователей и клиентов.

Приложение
Принципы проектирования

Глава 1

- Проектирование взаимодействия – не гадание на кофейной гуще.

Глава 2

- Пользовательский интерфейс должен следовать пользовательской ментальной модели, а не модели реализации.
- Целеориентированное взаимодействие отражает пользовательские ментальные модели.
- Пользователи не понимают булеву алгебру.
- Не копируйте артефакты механической эры в пользовательских интерфейсах без учета возможностей информационной эры.
- Существенные изменения должны приносить значительные улучшения.

Глава 3

- Никто не желает оставаться начинающим.
- Оптимизируйте для середняков.
- Считайте пользователей людьми очень умными, но очень занятыми.

Глава 5

- Не заставляйте пользователей чувствовать себя дураками.
- При проектировании каждого интерфейса сосредотачивайтесь на единственном ключевом персонаже.

Глава 6

- Определите, *что* должен делать продукт, прежде чем проектировать, *каким образом* он это будет делать.
- На ранних стадиях проектирования считайте интерфейс волшебным.

Глава 7

- Никогда не демонстрируйте вариант, который не нравится вам самим, потому что именно этот подход может понравиться заинтересованному лицу.
- Пользовательский опыт целостен – форму и поведение продукта следует проектировать согласованно.

Глава 9

- Решения о выборе технической платформы следует соотносить с работой по проектированию взаимодействия.
- Оптимизируйте монопольные приложения для работы на полном экране.
- Интерфейс монопольного приложения должен придерживаться консервативного визуального стиля.
- В монопольных приложениях следует применять обогащенные средства ввода.
- Разворачивайте документы в монопольных приложениях на полный экран.
- Временные приложения должны быть простыми, понятными, четкими.
- Временное приложение следует ограничивать одним окном и одним представлением.
- Временное приложение должно восстанавливать предыдущее положение и предыдущую конфигурацию.
- Киоски следует оптимизировать в расчете на пользователей, не имеющих опыта работы с ними.

Глава 10

- Каким бы замечательным ни был ваш интерфейс, чем его меньше, тем лучше.
- Хорошо оркестрованные пользовательские интерфейсы прозрачны.
- Следуйте ментальным моделям пользователей.
- Меньше – лучше.
- Позволяйте пользователям управлять, не принуждайте их к диалогу.
- Держите инструменты под рукой.
- Обеспечивайте немодальную обратную связь.
- Проектируйте наиболее вероятное, будьте готовы к возможному.
- Представляйте информацию с учетом контекста.
- Организуйте непосредственное манипулирование и графический ввод.

- Отображайте состояния объектов и статус приложения.
- Избегайте ненужных сообщений.
- Не используйте диалоговые окна, чтобы сообщить, что все нормально.
- Избегайте чистого листа.
- Просите прощения, а не разрешения.
- Отделяйте функции от их настройки.
- Не задавайте вопросы – предоставляйте выбор.
- Прячьте рычаги катапультирования.
- Оптимизируйте скорость реакции; предупреждайте о задержках.

Глава 11

- Сокращайте налоговое бремя при любой возможности.
- Не приваривайте дополнительные колеса к велосипеду намертво.
- Не прерывайте работу из-за ерунды.
- Не заставляйте пользователей просить разрешения.
- Разрешайте ввод везде, где имеется вывод.
- Адаптируйте интерфейс под типичную навигацию.
- Пользователи готовы прикладывать усилия соразмерно результату.

Глава 12

- Компьютер работает, а человек думает.
- Программный продукт должен вести себя как тактичный человек.
- Следует запоминать все, что выбирает пользователь.

Глава 13

- Люди предпочитают добиваться успеха, а не всеведения.
- Все идиомы необходимо изучать; хорошую идиому достаточно изучить только один раз.
- Никогда не подгоняйте интерфейс под метафору.

Глава 14

- Основой визуального интерфейса являются визуальные шаблоны.
- Элементы, различающиеся поведением, должны различаться и визуально.
- Доносите до пользователей функцию и поведение визуально.
- Удаляйте элементы, пока продукт не сломается, а затем верните последний удаленный элемент на место.

- Графически представляйте тип объекта; текстом описывайте сам объект.
- Придерживайтесь стандарта, если нет действительно стоящей альтернативы.
- Единство оформления не подразумевает косности в решениях.

Глава 17

- Управление дисками и файлами не входит в список целей пользователей.
- Сохраняйте документы и настройки автоматически.
- Помещайте файлы туда, где пользователи смогут их найти.
- Диски – технологический трюк, а не конструктивный элемент.

Глава 18

- В ошибках может и не быть вашей вины, но вы несете за них ответственность.
- Выполняйте проверку, а не редактирование.

Глава 19

- Полноценная визуальная обратная связь – ключ к успешному непосредственному манипулированию.
- В задачах, связанных с навигацией и выделением, обеспечивайте поддержку как мыши, так и клавиатуры.
- Используйте курсорные подсказки для отражения смысла служебных клавиш.
- Одиночный щелчок выбирает информационный объект либо изменяет состояние элемента управления.
- Двойной щелчок означает одинарный щелчок с последующим действием.
- Если курсор находится над объектом или данными, событие «кнопка нажата» должно приводить к выделению этого объекта или данных.
- Когда курсор находится над элементом управления, событие «кнопка нажата» означает подготовку к действию, событие «кнопка отпущена» приводит к выполнению действия.
- Передавайте отзывчивость элементов интерфейса с помощью визуальных подсказок.
- Применяйте курсорные подсказки для передачи отзывчивости элементов интерфейса.
- Выделение должно быть визуально отчетливым и недвусмысленным.
- Потенциальные цели должны визуально демонстрировать свою готовность принимать объекты.

- Курсор при перетаскивании должен визуально ассоциироваться с объектом-источником.
- Любая цель перетаскивания, до которой можно добраться прокруткой, должна быть доступна посредством автоматической прокрутки.
- Компенсируйте дребезг перетаскивания.
- Любая программа, в которой требуется точное выравнивание, должна предлагать пользователю верньер.

Глава 20

- Диалоговое окно – это еще одна комната; не ходите в комнату без веской причины.
- Встраивайте функции в то окно, где они применяются.
- Полезность любой идиомы взаимодействия зависит от контекста.

Глава 21

- Большое количество диалоговых окон, перегруженных элементами управления, не гарантирует качество пользовательского интерфейса.
- Используйте ссылки для навигации, а кнопки и кнопки-значки – для выполнения действий.
- Выделяйте наиболее важные элементы списков с помощью пиктограмм.
- Ни в коем случае не используйте горизонтальную прокрутку текста.
- Применяйте ограничивающие элементы управления для получения ограниченных данных.
- Чтобы отобразить информацию, недоступную для редактирования, используйте элементы, предназначенные только для вывода.

Глава 22

- Используйте меню для обучения пользователей.
- Блокируйте пункты меню всегда, когда их функции теряют смысл.
- Используйте одинаковые пиктограммы для элементов управления, решающих одни и те же задачи.

Глава 23

- Инструментальные панели обеспечивают возможность быстрого доступа к часто используемым функциям для опытных пользователей.
- Предусматривайте всплывающие подсказки для всех элементов управления, расположенных на инструментальной панели или представленных пиктограммами.

Глава 24

- Реализуйте основное взаимодействие в основном окне.
- Применение диалоговых окон оправдано для функций, не входящих в основной поток взаимодействия.
- Диалоговые окна подходят для объединения элементов управления и информации, связанных с одним объектом предметной области или с одной функцией приложения.
- Используйте глаголы в заголовках функциональных диалоговых окон.
- Используйте названия объектов в заголовках диалоговых окон свойств.
- Визуально выделяйте различия между модальными и немодальными диалоговыми окнами.
- Используйте единообразные терминальные команды внутри немодальных диалоговых окон.
- Не допускайте динамического изменения надписей на терминальных кнопках.
- Информируйте пользователя о том, что приложение занято.
- Никогда не используйте временные диалоговые окна для вывода сообщений об ошибках или сообщений, требующих подтверждения.
- У всех идиом взаимодействия есть практические ограничения.
- Не создавайте многострочные вкладки.

Глава 25

- Диалоги с сообщениями об ошибках прекращают работу из-за ерунды; их следует избегать.
- Делайте так, чтобы ошибки были невозможны.
- Программа унижает пользователей, когда сообщает, что они ошиблись.
- Не расспрашивайте – действуйте.
- Все действия должны быть обратимыми.
- Помогайте пользователям избегать ошибок при помощи немодальной обратной связи.

Глава 26

- Предлагайте информацию о клавиатурных сокращениях в меню Справка.
- Предложите пользователям коллекции шаблонов, готовых к применению.

Библиография

[Alexander, 1964] Alexander, Christopher. *Notes on the Synthesis of Form.* Harvard University Press, 1964.

[Alexander, 1977] Alexander, Christopher. *A Pattern Language.* Oxford University Press, 1977.

[Alexander, 1979] Alexander, Christopher. *The Timeless Way of Building.* Oxford University Press, 1979.

[Bertin, 1983] Bertin, Jacques. *Semiology of Graphics.* University of Wisconsin Press, 1983.

[Beyer and Holtzblatt, 1998] Beyer, Hugh, and Holtzblatt, Karen. *Contextual Design.* Morgan Kaufmann Publishers, 1998.

[Borchers, 2001] Borchers, Jan. *A Pattern Approach to Interaction Design.* John Wiley and Sons, 2001.

[Borenstein, 1994] Borenstein, Nathaniel S. *Programming As If People Mattered.* Princeton University Press, 1994.

[Buxton, 1990] Buxton, Bill. «The 'Natural' Language of Interaction: A Perspective on Non-Verbal Dialogues.» Laurel, Brenda, ed. *The Art of Human-Computer Interface Design.* Addison-Wesley, 1990.

[Carroll, 1995] Carroll, John M. ed. *Scenario-Based Design.* John Wiley and Sons, 1995.

[Carroll, 2000] Carroll, John M. *Making Use: Scenario-based Design of Human-Computer Interactions.* The MIT Press, 2000.

[Constantine and Lockwood, 1999] Constantine, Larry L., and Lockwood, Lucy A. D. *Software for Use.* Addison-Wesley, 1999.[1]

[Constantine and Lockwood, 2002] Constantine, Larry L., and Lockwood, Lucy A. D. *forUse Newsletter* #26, October 2002.

[Cooper, 1999] Cooper, Alan. *The Inmates Are Running the Asylum.* SAMS/Macmillan, 1999.[2]

---

[1] Константайн Л., Локвуд Л. «Разработка программного обеспечения». – Пер. с англ. – СПб.: Питер, 2004.

[2] Купер А. «Психбольница в руках пациентов. Почему высокие технологии сводят нас с ума и как восстановить душевное равновесие», дополненное издание. – Пер. с англ. – СПб.: Символ-Плюс, 2009.

[Crampton and Tabor, 1996] Crampton Smith, Gillian, and Tabor, Philip. «The Role of the Artist-Designer.» Winograd, Terry, ed. *Bringing Design to Software*. Addison-Wesley, 1996.

[Csikszentmihalyi, 1991] Csikszentmihalyi, Mihaly. *Flow: The Psychology of Optimal Experience*. HarperCollins, 1991.

[DeMarco and Lister, 1999] DeMarco, Tom, and Lister, Timothy R. *Peopleware*. Dorset House, 1999.[1]

[Dillon, 2001] Dillon, Andrew. «Beyond Usability: Process, Outcome and Affect in Human Computer Interaction.» Paper presented at the Lazerow Lecture at the Faculty of Information Studies, University of Toronto, March 2001. Находится по адресу *www.ischool.utexas.edu/~adillon/publications/beyond_usability.html*.

[Gamma, et al., 1995] Gamma, Erich, et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1995.[2]

[Garrett, 2002] Garrett, Jesse James. *The Elements of User Experience*. New Riders Press, 2002.[3]

[Gellerman, 1963] Gellerman, Saul W. *Motivation and Productivity*. Amacom Press, 1963.

[Goodwin, 2001] Goodwin, Kim. «Perfecting Your Personas.» *Cooper Newsletter*, July/August 2001.

[Goodwin, 2002] Goodwin, Kim. «Getting from Research to Personas: Harnessing the Power of Data.» User Interface 7 West Conference, 2002.

[Goodwin, 2002a] Goodwin, Kim. Cooper U Interaction Design Practicum Notes. Cooper, 2002.

[Grudin and Pruitt, 2002] Grudin, J., and Pruitt, J. «Personas, Participatory Design and Product Development: An Infrastructure for Engagement.» *PDC'02: Proceedings of the Participatory Design Conference*, 2002.

[Heckel, 1994] Heckel, Paul. *The Elements of Friendly Software Design*. Sybex, 1994.

[Horn, 1998] Horn, Robert E. *Visual Language*. Macro Vu Press, 1998.

[Horton, 1994] Horton, William. *The Icon Book: Visual Symbols for Computer Systems and Documentation*. John Wiley & Sons, 1994.

---

[1] Демарко Т., Листер Т. «Человеческий фактор: успешные проекты и команды». – Пер. с англ. – СПб.: Символ-Плюс, 2005.

[2] Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. «Приемы объектно-ориентированного проектирования. Паттерны проектирования». – Пер. с англ. – СПб.: Питер, 2001, 2007.

[3] Гарретт Д. «Веб-дизайн: книга Джесса Гарретта. Элементы опыта взаимодействия». – Пер. с англ. – СПб.: Символ-Плюс, 2008.

[Johnson, 2000] Johnson, Jeff. *GUI Bloopers.* Morgan Kaufman Publishers, 2000.

[Jones and Marsden, 2006] Jones, Matt, and Marsden, Gary. *Mobile Interaction Design.* John Wiley & Sons, 2006.

[Kobara, 1991] Kobara, Shiz. *Visual Design with OSF/Motif.* Addison-Wesley, 1991.

[Korman, 2001] Korman, Jonathan. «Putting People Together to Create Good Products.» *Cooper Newsletter*, September 2001.

[Krug, 2000] Krug, Steve. *Don't Make Me Think!* New Riders Press, 2000.[1]

[Kuniavsky, 2003] Kuniavsky, Mike. *Observing the User Experience.* Morgan Kaufmann Publishers, an Imprint of Elsevier, 2003.

[Kuutti, 1995] Kuutti, Kari. «Work Processes: Scenarios as a Preliminary Vocabulary.» Carroll, John M., ed. *Scenario-based Design.* John Wiley and Sons, Inc., 1995.

[Laurel, 1991] Laurel, Brenda. *Computers as Theatre.* Addison-Wesley, 1991.

[Lidwell, Holden, and Butler, 2003] Lidwell, William; Holden, Kritina; Butler, Jill. *Universal Principles of Design.* Rockport Publishers, 2003.

[MacDonald, 2003] MacDonald, Nico. *What Is Web Design?* Rotovision 2003.

[McCloud, 1994] McCloud, Scott. *Understanding Comics.* Kitchen Sink Press, 1994.

[Mikkelson and Lee, 2000] Mikkelson, N., and Lee, W. O. «Incorporating user archetypes into scenario-based design.» *Proceedings of UPA 2000* (2000).

[Miller, 1968] Miller, R. B. Response time in man-computer conversational transactions. *Proc. AFIPS Fall Joint Computer Conference,* Vol. 33, 267–277 (1968).

[Mitchell and Shneiderman, 1989] Mitchell, J. and Shneiderman, B. Dynamic versus static menus: An exploratory comparison. *SIGCHI Bulletin*, Vol. 20 No. 4, 33–37 (1989).

[Moggridge, 2007] Moggridge, Bill. *Designing Interactions.* The MIT Press, 2007.

[Morville, 2005] Morville, Peter. *Ambient Findability.* O'Reilly Media, 2005.[2]

[Mulder and Yaar, 2006] Mulder, Steve, and Yaar, Ziv. *The User Is Always Right.* New Riders Press, 2006.

---

[1] Круг С. «Веб-дизайн: книга Стива Круга. Не заставляйте меня думать!», 2-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2008.

[2] Морвиль П. «Тотальная видимость. Как наши находки меняют нас». – Пер. с англ. – СПб.: Символ-Плюс, 2008.

[Mullet and Sano, 1995] Mullet, Kevin, and Sano, Darrell. *Designing Visual Interfaces*. Sunsoft Press, 1995.

[Nass and Reeves, 1996] Nass, Clifford, and Reeves, Byron. *The Media Equation: How People Treat Computers, Television, and New Media Like Real People and Places*. Cambridge University Press, 1996.

[Nelson, 1990] Nelson, Theodor Holm. «The Right Way to Think about Software Design.» Laurel, Brenda, ed. *The Art of Human-Computer Interface Design*. Addison-Wesley, 1990.

[Newman and Lamming, 1995] Newman, William M., and Lamming, Michael G. *Interactive System Design*. Addison-Wesley, 1995.

[Nielsen, 1993] Nielsen, Jakob. *Usability Engineering*. Academic Press, 1993.

[Nielsen, 2000] Nielsen, Jakob. *Designing Web Usability*. New Riders Press, 2000.[1]

[Nielsen, 2002] Nielsen, Jakob. UseIt.com (веб-сайт).

[Norman, 1989] Norman, Donald. *The Design of Everyday Things*. Currency Doubleday, 1989.[2]

[Norman, 1994] Norman, Donald. *Things That Make Us Smart*. Perseus Publishing, 1994.

[Norman, 1998] Norman, Donald A. *The Invisible Computer*. The MIT Press, 1998.

[Norman, 2005] Norman, Donald. *Emotional Design*. Basic Books, 2005.

[Papanek, 1984] Papanek, Victor. *Design for the Real World*. Academy Chicago Publishers, 1984.[3]

[Perfetti and Landesman, 2001] Perfetti, Christine, and Landesman, Lori. «The Truth About Download Times» UIE.com, 2001.

[Pinker, 1999] Pinker, Stephen. *How the Mind Works*. W. W. Norton & Company, 1999.

[Preece, Rogers, and Sharp, 2007] Preece, Jenny; Rogers, Yvonne and Sharp, Helen. *Interaction Design*. John Wiley & Sons, 2007.

[Raskin, 2000] Raskin, Jeff. *The Humane Interface*. Addison-Wesley Professional, 2000.[4]

---

[1] Нильсен Я. «Веб-дизайн: книга Якоба Нильсена». – Пер. с англ. – СПб.: Символ-Плюс, 2000.

[2] Норман Д. «Дизайн промышленных товаров». – Пер. с англ. – М.: Вильямс, 2008; Норман Д. «Дизайн привычных вещей». – Пер. с англ. – М.: Вильямс, 2006.

[3] Папанек В. «Дизайн для реального мира». – Пер. с англ. – М.:Издатель Д. Аронов, 2004.

[4] Раскин Д. «Интерфейс: новые направления в проектировании компьютерных систем». – Пер. с англ. – СПб.: Символ-Плюс, 2003.

[Reimann, 2001] Reimann, Robert M. «So You Want to Be an Interaction Designer.» *Cooper Newsletter*, June 2001.

[Reimann, 2002] Reimann, Robert M. «Bridging the Gap from Research to Design.» Panel Presentation, IBM Make IT Easy Conference (2002).

[Reimann, 2002a] Reimann, Robert. «Perspectives: Learning Curves.» *Redesign Magazine*, Dec. 2002.

[Reimann, 2005] Reimann, Robert. «Personas, Scenarios, and Emotional Design». UXMatters.com (2005).

[Reimann and Forlizzi, 2001] Reimann, Robert M., and Forlizzi, Jodi. «Role: Interaction Designer.» Presentation to AIGA Experience Design 2001.

[Rheinfrank and Evenson, 1996] Rheinfrank, John, and Evenson, Shelley. «Design Languages.» Winograd, Terry, ed. *Bringing Design to Software.* Addison-Wesley, 1996.

[Rombaur and Becker, 1975] Rombaur, Irma S., and Becker, Marion Rombaur. *The Joy of Cooking.* Scribner, 1975.

[Rosenfeld and Morville, 1998] Rosenfeld, Louis, and Morville, Peter. *Information Architecture.* O'Reilly, 1998.[1]

[Rudolf, 1998] Rudolf, Frank. «Model-Based User Interface Design: Successive Transformations of a Task/Object Model.» Wood, Larry E., ed. *User Interface Design: Bridging the Gap from User Requirements to Design.* CRC Press, 1998.

[Saffer, 2006] Saffer, Dan. *Designing for Interaction.* Peachpit Press, 2006.

[Schön and Bennett, 1996] Schön, D., and Bennett, J. «Reflective Conversation with Materials.» Winograd, Terry, ed. *Bringing Design to Software.* Addison-Wesley, 1996.

[Schumann, et al., 1996] Schumann, J., Strothotte, T., Raab, A., and Laser, S. *Assessing the Effect of Non-Photorealistic Rendered Images in CAD.* CHI 1996 Papers, pp. 35–41 (1996).

[Shneiderman, 1998] Shneiderman, Ben. *Designing the User Interface.* Addison-Wesley, 1998.

[Simon, 1996] Simon, Herbert. *The Sciences of the Artificial.* The MIT Press, 1996.

[Snyder, 2003] Snyder, Carolyn. *Paper Prototyping.* Morgan Kaufmann Publishers, an Imprint of Elsevier, 2003.

[SRI Consulting Business Intelligence, 2002] SRI Consulting Business Intelligence. «Welcome to VALS.» SRI-BC.com (2002).

---

[1] Розенфельд Л., Морвиль П. «Информационная архитектура в Интернете», 2-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2005.

[Tidwell, 2006] Tidwell, Jennifer. *Designing Interfaces*. O'Reilly Media, 2006.[1]

[Tufte, 1983] Tufte, Edward. *The Visual Display of Quantitative Information*. Graphic Press, 1983.

[Van Duyne, Landay, and Hong, 2002] Van Duyne, Douglas K., Landay, James A., Hong, Jason I. *The Design of Sites*. Addison-Wesley, 2002.

[Veen, 2000] Veen, Jeffrey. *The Art and Science of Web Design*. New Riders Press, 2000.

[Verplank, et al., 1993] Verplank, B., Fulton, J., Black, A., and Moggridge, B. «Observation and Invention: Use of Scenarios in Interaction Design.» Tutorial Notes, InterCHI'93, Amsterdam, 1993.

[Weiss, 2000] Weiss, Michael J. *The Clustered World: How We Live, What We Buy, and What It All Means About Who We Are*. Little Brown & Company, 2000.

[Winograd, 1996] Winograd, Terry, ed. *Bringing Design to Software*. Reading, MA: Addison-Wesley, 1996.

[Wirfs-Brock, 1993] Wirfs-Brock, Rebecca. «Designing Scenarios: Making the Case of a Use Case Framework.» *SmallTalk Report*, November/December 1993.

[Wixon and Ramey, 1996] Wixon, Dennis, and Ramey, Judith, eds. *Field Methods Casebook for Software Design*. John Wiley and Sons, 1996.

[Wood, 1996] Wood, Larry E. «The Ethnographic Interview in User-Centered Task/Work Analysis.» (1996).

Руководства по стилю интерфейса

[Apple Computer, 1992] Apple Computer. *Macintosh Human Interface Guidelines*. Addison-Wesley, 1992.

[Apple Computer, 2002] Apple Computer. *Aqua Human Interface Guidelines*. Apple Developer Website (2002).

[Microsoft, 1999] Microsoft. Windows *User Experience Guidelines*. Microsoft Press, 1999.

[Sun MicroSystems, 1990] Sun MicroSystems. *Open Look Graphical User Interface* Application Style Guidelines. Addison-Wesley, 1990.

[1] Тидвелл Дж. «Разработка пользовательских интерфейсов». – Пер. с англ. – СПб.: Питер, 2008.

Словарь терминов

Адаптация [интерфейса]	Inflecting	Замусоривание окнами	Windows pollution
Анализ задач (рабочих заданий)	Task analysis	Идиома	Idiom
Аналитический уровень обработки	Reflective level of processing	Интерфейсный налог	Excise
Ассоциирование	Mapping	Информационная архитектура	Information architecture (IA)
Бизнес-цель	Business goals	Информационный иммунитет	Data immunity
Видимость	Findability	Инфраструктура взаимодействия	Interaction framework
Визуальная идиома	Visual idiom	Инфраструктура интерфейса	Design framework
Визуальная инфраструктура	Visual framework	Исключительные ситуации	Edge cases
Визуальная метафора	Visual metaphor	Исследование целевой аудитории	User research
Визуальная обратная связь	Visual feedback	Карточная сортировка	Card sorting
Визуальный шум	Visual noise	Качественные исследования	Qualitative research
Временный тип интерфейса	Transient posture	Ключевой персонаж	Primary persona
Второстепенный персонаж	Secondary persona	Ключевые сценарии	Key path scenarios
Графический дизайн	Graphic design	Когнитивное сопротивление	Cognitive friction
Деятельность	Activity	Количественные исследования	Quantitative research
Дополнительный персонаж	Supplemental persona	Командные элементы управления	Imperative controls
Жизненные цели	Life goals	Компоновочная схема	Layout

Конечные цели	End goals	Ожидаемое назначение	Affordance
Контекст использования	Context of use	Определение инфраструктуры	Framework definition
Контекстное исследование	Contextual inquiry	Опыт пользователя	User experience
Контекстный сценарий	Context scenario	Отвергаемый персонаж	Negative personas
Лексикон	Vocabulary	Отзывчивость	Pliancy
Ментальная модель пользователя	User mental model	Переменные	Variables
Модель артефактов	Artifact model	Перетаскивание	Dragging
Модель представления	Manifest model, Represented model	Персонаж	Persona
Модель рабочих процессов	Flow model	Персонаж покупателя	Customer persona
Модель реализации	Implementation model	Пластилиновый пользователь	Elastic user
Монопольный тип интерфейса	Sovereign posture	Поведенческий уровень обработки	Behavioral level of processing
Набор персонажей	Cast of personas	Поведенческий шаблон	Behavior pattern
Немодальная обратная связь	Modeless feedback	Порядок взаимодействия «подлежащее-сказуемое»	Object verb ordering
Непосредственное манипулирование	Direct manipulating	Порядок взаимодействия «сказуемое-подлежащее»	Verb object ordering
Нормализованный лексикон	Controlled vocabulary	Поток (состояние сознания)	Flow
Обогащенная обратная связь	Rich feedback	Представление	View
Обратная связь	Feedback	Проверка проектных решений	Design validation
Обслуживаемый персонаж	Served persona	Проверочные сценарии	Validation scenarios
Общая концепция взаимодействия	High-level interaction framework	Проектирование	Design
Ограничивающий элемент ввода	Bounded control	Проектирование взаимодействия	Interaction design (IxD)

Проектирование, ориентированное на деятельность	Activity-centered design (ACD)	Физическая модель	Physical model
Проектирование, ориентированное на пользователей, людей	User-centered design, Human-centered design	Финальное тестирование	Summative testing
Промежуточное тестирование	Formative testing	Фоновый тип интерфейса	Daemonic posture
Промышленный дизайн	Industrial Design	Форм-фактор	Form factor
Рабочая среда	Environment	Функциональные потребности	Functional needs
Рассказ, повествование	Narrative	Целеориентированное проектирование	Goal-directed design
Режим	Mode	Шаблон поведения	Pattern of behavior
Руководство по стилю	Style guide	Шаблон проектирования	Design pattern
Система извлечения [данных]	Retrieval system	Эвристическая оценка	Heuristic review
Спецификация формы и поведения	Form and behavior specification	Эксперт предметной области	Subject matter expert (SME)
Способы взаимодействия	Input vectors	Экспертная оценка	Expert review
Сценарий	Scenario	Элемент ввода	Entry control
Теория деятельности	Activity Theory	Элемент управления	Control
Терминальная команда	Terminating command	Эмоциональные цели	Experience goals
Тип интерфейса	Posture	Этап детализации	Refining/Refinement phase
Требование	Requirement	Этнографическое интервью	Ethnographic interview
Сговорчивость	Fudgable	Юзабилити	Usability
Физиологический уровень обработки	Visceral level of processing	Юзабилити-тестирование	Usability testing

Алфавитный указатель

Специальные символы

.NET (Microsoft), 222
37signals
  Basecamp, 219
  Writeboard, 397
3ds Max (Autodesk), 576

A

«A Pattern Language» (Александер), 196
Ableton Live, 301, 520, 531
ACD (Activity-Centered Design), 45
ActiveX (Microsoft), 214
Activity-Centered Design (ACD), 45
Adobe, 392, 528
  Fireworks, 576
    для создания эскизов интерфейсов, 171
    и симметрия в интерфейсе, 348
    палитры инструментов, 348
      припаркованные, 576
  Fireworks MX, 576
  Flash, 181, 214, 482
  Framework, 180
  Illustrator, 29
    вкладки палитр, 278
    дискретное выделение, 444
    для создания эскизов интерфейсов, 171
    модальные инструменты, 463
    направляющие, 466
    сообщения об ошибках в диалоговых окнах, 272
    точное позиционирование, 461
  InDesign, 211
  Lightroom, 485
    Develop (Проявка), представление, 485, 486
    боковые панели, 577
    выравнивание по композиционной сетке, 344
  Photoshop, 29, 486, 575, 576
    Brightness/Contrast (Яркость/Контрастность), диалоговое окно, 484
    Navigator, панель, 285, 529
    Варианты (Variations), интерфейс, 62
    визуальное сравнение, 362
    выделение областью, 449
    диалоговое окно подтверждения, 608
    и идиомы обзора, 285
    и пользовательская ментальная модель, 62
    и режимы, 227
    индикация сложного выделения (область), 449
    клавиатурные сокращения, 620
    комбо-кнопки, 504
    модальные инструменты, 462
    навигация между элементами палитр, 278
    немодальная обратная связь, 252
    палитры инструментов немодальные, 575
    панели инструментов контекстные, 565
    предварительный просмотр, функция, 362
    точное позиционирование, 461
    форматирование текста в окне, 528
  комбо-кнопки, 504
  стандарты интерфейса, 365
AirSet Desktop Sync, 604
AJAX, 214, 482
Alto (Xerox), система, 434, 476
Amazon.com, 218, 273
  «хлебные крошки», 285

AOL (America Online), 489
API (Application Programming Interface) Windows, 572
Apple, 248, 326, 371, 430, 450, 477, 480, 490, 558
iCal, представление календаря, 71
iDVD, 337
iLife, 577
Inspectors, 565, 577
iPhoto, 337, 401, 545
iPod, 127, 225, 248, 402
Click Wheel, колесо, 127
Shuffle, интерфейс, 248
iTunes, 222, 337, 401, 491, 587
вкладки в диалоговом окне, 587
как временное приложение, 209
iWeb, 337, 565
Logic, 565
Mac OS, 29, 461, 494, 513, 548
Automator, инструмент, 450
Dock, элемент интерфейса, 337, 480
Expose, элемент интерфейса, 480
Spotlight, инструмент индексации и поиска, 305, 371, 377
автоматизация, 450
всплывающая справка, 558
деревья, 514
и визуальные свойства, 342
и ожидаемые назначения, 330
клавиши быстрого доступа, 550
панель управления, 213
перемещение объектов, 465
пиктограммы, 342, 351
работа в фоновом режиме, 305
радиокнопки, 502
регулятор громкости звука, 614
система меню, реализация, 623
состояния окон, 490
файловая система, 401
флажки, 499
Mac OS 7, 514, 558
Mac OS 9, 367
Mac OS X, 305, 335, 342, 351, 367, 371, 377, 401, 450, 481, 496, 499, 541, 550, 559, 614
Macintosh, 320, 326, 367, 375, 426, 429, 434, 449, 477–479, 481, 532, 554
рабочий стол, метафора, 324
система меню, реализация, 623
MacPaint, 448, 479
выделение областью, 449
отрицательная звуковая обратная связь, 611
стандарты интерфейса (Macintosh), 365–367, 426, 432, 449, 541
Archie, протокол, 215
Art Directors Toolkit (Code Line Communications), 211
Autodesk 3ds Max, 576
Automator, инструмент Mac OS X, 450

B

Backspace, клавиша, 394, 395
Basecamp (37signals), 219
BlackBerry (RIM), 201, 228, 229
«Blink» (Гладуэлл), 124
Bluetooth, 600
BMW, iDrive, система, 240
Borland International, 431
Delphi, нагруженный курсор, 465
«Brazil», фильм, 64
Brightness/Contrast (Яркость/Контрастность), диалоговое окно (Photoshop), 484
«BusinessWeek», журнал, 23

C

«Civilization» (Сид Мейер), 609
Code Line Communications Art Directors Toolkit, 211
ColorDoctor (Fujitsu), 360
CompuServe Navigator, 488, 492
«Computers as Theater» (Лорел), 147
«Contextual Design» (Бейер, Хольцблат), 91, 119, 143, 155
cookie, 217
Cooper, 18, 28, 577
Cross Country TravCorp, портал, 170, 172, 180
GettyGuide, система информационных киосков, 233
Softek Storage Manager, 382
дизайн интеллектуального телефона, 225
использование боковых панелей, 577
обогащенная визуальная немодальная обратная связь, 610
приложение для управления маркетинговыми кампаниями, 364
система управления отношениямя с клиентами, 578
Corel Painter, 487

функция замораживания, 398
CP/M, система, 533
Cross Country TravCorp, портал, 170, 172, 180

D

del.icio.us, 378
Delphi (Borland), нагруженный курсор, 465
Design Within Reach, сайт, 283
«Designing Interfaces» (Тидвелл), 197
«Designing Visual Interfaces» (Маллет, Сано), 248, 334, 240
Disney, студия, 147
Disney.com, сайт, 270
Dock, элемент интерфейса в Mac OS X, 337, 480
«Don't Make Me Think!» (Круг), 216

E

«Emotional Design» (Норман), 124– 127
Excel (Microsoft), 29, 367, 440, 616
вкладки, 277, 483
курсорные подсказки, 440
перетаскивание выделенного фрагмента, 447
разделители экрана, 277, 440
Experience Music Project, музей, 234
Explorer (Microsoft Windows)
см. Internet Explorer
Explorer Bars, боковые панели (Explorer, Проводник Windows), 577
Expose, элемент интерфейса Mac OS, 480

F

File Manager (Microsoft Windows 3.x), 253
Finance (Google)
визуальное сравнение, 362
идиома обзора и, 285
Finder (Mac OS), 402, 452
Firefox, броузер, 577, 583
боковая панель (Slide Bar), 577
запоминание ранее введенных данных, 311
комбо-кнопки, 504
тактичность, 295
Fireworks (Adobe/Macromedia), 576
создание эскизов интерфейсов, 171
симметрия в интерфейсе и, 348
палитры инструментов, 348
припаркованные, 576
Flickr, 221, 378
«Flow: The Psychology of Optimal Experience» (Чиксентмихайи), 243
Framework (Adobe), 180
FTP (File Transfer rotocol), 215
Fujitsu ColorDoctor, 360

G

General Magic, Magic Cap, интерфейс, 323, 628
GettyGuide, система информационных киосков, 233
Google, 218, 248, 371
Finance, 285, 362
SketchUp, 466, 472– 474, 525
Spreadsheets, 219
интерфейс поиска, 248
Gopher, протокол, 215
GRiD Compass, ноутбук, 22
GUI (graphical user interface), 36

H

«Hare Brain, Tortoise Mind» (Клэкстон), 124
Harmony, пульт дистанционного управления (Logitech), 238– 239
HCI (Human-Computer Interaction), сообщество, 148
HDTV (High-Definition Television), 237
Hog Bay Software, WriteRoom, 248
HTML, 181
Human-Computer Interaction (HCI), сообщество, 148

I

IBM, 491, 532
Lotus 1-2-3, видимые иерархические меню, 535
Lotus Notes, 256
iCal (Apple), представление календаря, 71
iDrive, система BMW, 240
iDVD (Apple), 337
iLife (Apple), 577
Illustrator (Adobe), 29
вкладки палитр, 278
дискретное выделение, 444
модальные инструменты, 463

направляющие, 466
создание эскизов интерфейсов, 171
сообщения об ошибках в диалоговых окнах, 272
точное позиционирование, 461
«Information Architecture» (Розенфельд, Морвиль), 216
Inspectors (Apple), 565, 577
Internet Explorer (Microsoft), 29, 254, 402, 432, 446, 452, 491, 577, 583
автоматическая прокрутка, 456
боковые панели (Explorer Bars), 577
групповое выделение, 446
деревья, 514
запомнание ранее введенных данных, 311
индикатор хода выполнения, 582
как временное приложение, 209
кумулятивное выделение, 445
отмена файловых операций, 583
переименование файла, 404
перемещение файлов, 427
сортировка файлов, 510
списки, 512
тактичность, 295
iPhoto (Apple), 337, 401, 545
iPod (Apple), 127, 225, 248, 402
Click Wheel, колесо, 127
Shuffle, интерфейс, 248
iTunes (Apple), 222, 337, 401, 491, 587
вкладки в диалоговом окне, 587
как временное приложение, 209
iWeb (Apple), 337, 565
IxD (Interaction Design), 25
IxDA (Interaction Design Association), 23

J

Java, 214, 222
JCL, язык управления заданиями, 532

L

LibraryThing, 378
LIFO (last in, first out), стек, 389, 390
Lightroom (Adobe), 485
Develop (Проявка), представление, 485, 486
боковые панели, 577
выравнивание по композиционной сетке, 344
Linux, 29
Lisa, система, 477
Logic (Apple), 565
Logitech Harmony, пульт дистанционного управления, 238, 239
Lotus 1-2-3 (IBM/Lotus), видимые иерархические меню, 535
Lotus Notes (IBM), 256

M

Mac OS (Apple), 29, 461, 494, 513, 548
Automator, 450
Dock, 337, 480
Expose, 480
Finder, 402, 452
Spotlight, индексация и поиск, 305, 371, 377
автоматизация, 450
всплывающая справка, 558
деревья, 514
и визуальные свойства в интерфейсе, 342
и ожидаемые назначения в интерфейсе, 330
клавиши быстрого доступа, 550
панель управления, 213
перемещение объектов, 465
пиктограммы, 342, 351
фотореалистичные, 351
работа в фоновом режиме, 305
радиокнопки, 502
регулятор громкости звука, 614
система меню, реализация, 623
состояния окон, 490
файловая система, 401
флажки, 499
Mac OS 7 (Apple), 514, 558
Mac OS 9 (Apple), 367
Mac OS X (Apple), 305, 335, 342, 351, 367, 371, 377, 401, 450, 481, 496, 499, 541, 550, 559, 614
Macintosh (Apple), 320, 326, 367, 375, 426, 429, 434, 449, 477–481, 532, 554
рабочий стол, метафора, 324, 557
система меню, реализация, 623
MacPaint (Apple), 448, 479
выделение области, 449
Macromedia
Fireworks, 576
создание эскизов интерфейсов, 171
и симметрия в интерфейсе, 348
палитры инструментов, 348
припаркованные, 576

Flash, 181, 214, 482
Magic Cap (General Magic), 628
интерфейс, 323
«Making Use» (Кэрролл), 148
McDonalds, 322
MDI (многодокументный интерфейс), 491–493, 543
Microsoft Windows, 29, 461, 480, 494, 496, 499, 513, 526, 548
ActiveX, 214
API, 572
Excel, *см.* Excel
File Manager, 253
Internet Explorer, *см.* Internet Explorer
MDI и SDI, 491, 543
.NET, 222
Office, *см.* Office
Outlook, *см.* Outlook
PowerPoint, *см.* PowerPoint
Visio, 171, 460
создание эскизов интерфейсов, 171
Visual Basic, *см.* Visual Basic
Windows, *см.* Windows
Word, *см.* Word
Mitsubishi, 322
Motif, оболочка, 490
и ожидаемые назначения, 330
радиокнопки, 502
стандарты интерфейса, 541
Motorola Razr, телефон, 247, 323
MoveableType (SixApart), 219
Mozilla Firefox, 577, 583
боковая панель (Slide Bar), 577
запоминание ранее введенных данных, 311
комбо-кнопки, 504
тактичность, 295
MS Bob (Microsoft), 628
Multiplan, 477

N

Native Instruments Reaktor, 519
Navigator (CompuServe), 488–489, 492
Navigator, панель в Adobe Photoshop, 285, 529
Netscape, 482
.NET, 222

O

«Observing the User Experience» (Кунявски), 103, 106
Office (Microsoft), 29, 309, 577, 629
адаптивное меню, 546
боковые панели (Task Pane), 577
кнопки-значки, 497, 500
комбо-кнопки, 503
лента (ribbon), 546–548, 564
персонализация, 620
пиктограммы в меню, 550
Office 2000 (Microsoft), 546
Office 2003 (Microsoft), 497, 500, 546
Office 2007 (Microsoft), 506, 546, 560, 564
Office для Mac OS X (Microsoft), 497
OmniGraffle (Omni Group)
создание эскизов интерфейсов, 171
интеллектуальные направляющие (Smart Guides), 466
Open Look (Sun), 468
Organize, инструмент Flickr, 221
Outlook (Microsoft), 29, 256, 367, 442
как многопанельное приложение, 483
как монопольное приложение, 204
представление календаря, 70, 530
сообщения об ошибках в диалоговых окнах, 272
структурный шаблон, 199
Outlook 2007 (Microsoft), 483
Outlook Express (Microsoft), 509

P

Paint (Windows), 463
Painter (Corel), 487
функция замораживания, 398
Palm Treo, смартфон, 37, 227–232
переключение режимов, 227
PalmOS, раскрывающиеся меню, 552
Palo Alto Research Center (PARC), 22, 326, 476, 482
«Paper Prototyping» (Снайдер), 163, 183
PARC (Palo Alto Research Center), 326, 476, 482
«Peopleware: Productive Projects and Teams» (Демарко, Листер), 243
Photoshop (Adobe), 29, 484, 575, 576
Brightness/Contract (Яркость/Контрастность), диалоговое окно, 484

Navigator, панель, 285, 529
Варианты (Variations), функция, 62
визуальное сравнение, 362
выделение области, 449
диалоговое окно подтверждения, 608
и идиома обзора, 285
и пользовательская ментальная модель, 62
и режимы, 227
индикация сложного выделения (область), 448
клавиатурные сокращения, 620
комбо-кнопки, 504
модальные инструменты, 462
навигация между элементами палитр, 279
немодальная обратная связь, 252
палитры инструментов немодальные, 575
панели инструментов контекстные, 565
предварительный просмотр, функция, 362
точное позиционирование, 461
форматирование текста в окне, 528
PowerPoint (Microsoft), 29, 368, 446
адаптивные меню, 546
глобальный стандарт, 368
дискретное выделение, 444
и мастера, 627
как монопольное приложение,203
лента (ribbon), 547
маркеры, 467
нагруженный курсор, 464
полилинии, 468
создание нового документа, 258
создание эскизов интерфейса, 171
сортировщик слайдов, 447, 454
точное позиционирование, 461
PowerPoint 2003 (Microsoft), 546
PowerPoint 2007 (Microsoft), 467, 547
PRIZM, геодемографические кластеры, 104

R

«Rapid Contextual Design» (Хольцблат, Уэнделл, Вуд), 91
Razr (Motorola), 247
Reaktor (Native Instruments), 519
REI.com, 218
RIM BlackBerry, 201, 228, 229
RSS, 222

S

Salesforce.com, 219
ans-serif, шрифты, 358, 361
SAP R/3, ERP-система, 219, 492
SDI (интерфейс одного документа), 491
serif, шрифты, 358, 361
Shuffle, интерфейс (iPod), 248
SixApart MoveableType, 219
SketchUp (Google), 466, 471– 474, 525
Slide Bar, боковая панель (Firefox), 577
Smart Folders, механизм сохранения поисковых запросов в Mac OS X, 377
Smart Guides, интеллектуальные направляющие в OmniGraffle, 466
Softek Storage Manager, 382
«Software for Use» (Константайн), 75
Sony, 323
Walkman, 127
пульты дистанционного управления, 393
«Sources of Power» (Клейн), 124
Spotlight, функция индексации и поиска в Mac OS X, 305, 371, 377
Spreadsheets (Google), 219
SQL (Structured Query Language), 383, 544
Star (Xerox), система, 434, 477
«Star Trek», фильм, 266, 321, 428
Storage Manager (Cooper, Softek), 382
Submit, кнопка, 296
Sun Microsystems, 430
Open Look, 468
Swing, 222

T

Task Pane, боковые панели (Microsoft Office), 577
«The Art and Science of Web Design» (Вин), 216
«The Design of Everyday Things» (Норман), 329– 330, 617
«The Icon Book» (Хортон), 350
«The Inmates Are Running the Asylum» (Купер), 30, 112
«The Joy of Cooking» (Ромбаур, Беккер), 626
«The Media Equation» (Насс, Ривз), 261, 293, 294
«The Semiology of Graphics» (Бертен), 338
«The Timeless Way of Building» (Александер), 196

«The Visual Display of Quantitative Information» (Тафти), 336, 361
TiVo, 225, 237, 238

U

«Universal Principles of Design» (Лидвел, Холден и Батлер), 355
UNIX
виртуальные рабочие столы, 481
графические оболочки, 490
деревья, 514
мнемоники, 552
система хранения файлов, 372, 374
удаление файла, 441
«Usability Engineering» (Нильсен), 106, 182
useit.com, веб-сайт (Нильсен), 216
UX (user experience), 24

V

VALS сегментация, 104
Visio (Microsoft), программа, 460
создание эскизов интерфейсов, 171
Visual Basic (Microsoft), 490
нагруженный курсор, 464–465
формы, 490

W

Walkman (Sony), 127
Web 2.0, 214
Widgets (Yahoo!), как временное приложение, 209
WiFi, 600
wiki, 219
Windows (Microsoft), 29, 65, 461, 480, 494, 496, 499, 513, 526, 548
API, 572
автоматическая прокрутка, 454–456
встроенный поиск, 305, 377
деревья, 514
диалоговые окна, заголовки, 569
дискретное выделение, 445
и визуальное выделение, 356–357
и визуальные свойства, 340–341
и ожидаемые назначения, 330
индексация и поиск, 305, 377, 406
индикация
ожидания, 441
свободного места на диске, 253
хода операции, 581–583
интерфейс в модели реализации, 65
каскадное меню Пуск (Start), 546
каскадные диалоговые окна, 590
клавиатурные ментальные векторы, 619–620
клавиатурные сокращения, 619–620
комбо-кнопки, 503
курсорная индикация, 452
Менеджер файлов (File Manager), 253– 255
мнемоники, 552, 619 –620
непрерывное выделение, 445
область уведомлений на панели задач, 213
панель инструментов, 538
панель управления, 213
переименование файла, 404
перемещение объектов, 465
персонализация, 620
пиктограмма заголовка окна, 357
подтверждение удаления файла, диалоговое окно, 605
ползунок, 514
работа в фоновом режиме, 305
радиокнопки, 502
регулятор громкости звука, 614
режим меню, 462
Свойства диска, диалоговое окно, 365
система меню, реализация, 623
состояния окон, 490
списки, 505
файловая система, 401
флажки, 499
Windows 1.0 (Microsoft), 480, 491
Windows 2.0 (Microsoft), 481
Windows 3.0 (Microsoft), 496, 512
Windows 3.x (Microsoft), 253
Windows 95 (Microsoft), 431, 490, 506, 512, 514, 612, 623
Windows 98 (Microsoft), 497
Windows 2000 (Microsoft), 254, 497
Windows Internet Explorer (Microsoft) *см.* Internet Explorer
Windows Mobile (Microsoft), 227, 231
раскрывающиеся меню, 552
Windows Vista (Microsoft), 254, 351, 401, 481, 590
пиктограммы, 351
Windows XP (Microsoft), 213, 254, 335, 340, 356, 401, 404, 436, 456, 481, 502, 506, 512, 516, 628

Word (Microsoft), 29, 367, 435, 453, 524, 619
Автозамена, функция, 423
Автоформат, функция, 424
аннотированная полоса прокрутки, 286
аудит и редактирование, 423
групповая множественная отмена, 391
групповое выделение, 445
диалоговое окно свойств, 579
диалоговые окна
как средство обучения, 568
заголовки, 566
информирующие, 584
уведомлений, 603
запоминание выбора пользователя, 309
и симметрия в интерфейсе, 348
изменение формата документа, 411
каскадное меню, 545
кнопки представления документа, 207
комбо-кнопки, 504
комбо-списки, 513
на панели инструментов, 561
компьютерный стиль мышления, 37
лента (ribbon), 564
меню на панели инструментов, 561
многострочные вкладки, 588
Найти и заменить, диалоговое окно, 589
немодальная обратная связь, 251, 423
ненужные диалоговые окна, 38, 252
неподобающее поведение, 38
отделение функции от настройки (печать), 259
панели инструментов, 206, 260, 557
настройка, 564
Параметры, диалоговое окно, 588
переименование документа, 37
перетаскивание фрагментов текста, 447, 466
полосы прокрутки, 529
подтверждение сохранения документа, 38, 252, 400
Предварительный просмотр, диалоговое окно, 352–353
представление структуры, 529
радиокнопки со значком, 503
разворачивающееся диалоговое окно, 589–590
редактирование и аудит, 423
создание нового документа, 258
Сохранить изменения, диалоговое окно, 252, 400
список недавних документов, 542
Список, диалоговое окно, 348
Статистика, диалоговое окно, 251
счетчики, 518
Табуляция, команда, 568
Файл, меню, 542
функция отмены, 391–392
Шрифт, диалоговое окно, 579
Word 2003 (Microsoft), 251, 545, 555, 559
Word 2007 (Microsoft), 251, 286, 564
WordStar (WordStar Corporation), 204
Writeboard (37signals), 397
Writely, 219
WriteRoom (Hog Bay Software), 248
WYSIWYG (what you see is what you get), 327, 476, 528

X

Xerox
Alto, система, 476
Palo Alto Research Center (PARC), 326, 476
Star, система, 477

Y

Yahoo!, 218
Widgets, как временное приложение, 209

А

Автозамена, функция Word, 423
автоматическая прокрутка при перетаскивании, 454
автоматическое сохранение документов, 408
автомобильные интерфейсы, проектирование, 239
Автоформат, функция Word, 424
агенты интеллектуальные, 628
адаптация интерфейса под нужды пользователя, 289
адаптивные меню, 546
аккордный щелчок, 435
аккуратная обработка сбоев, 300
активная проверка допустимости, 522

Александер, Кристофер
«A Pattern Language», 196
«The Timeless Way of Building», 196
архитектурные шаблоны проектирования, 55, 198
анализ
критический, 129
рабочих заданий, 107
формы, 179
аналитический уровень когнитивной обработки, 125
анимация, 364
аннотированная полоса прокрутки, 285
аппаратное обеспечение, проектирование совместно с программным обеспечением, 224
артефактов, модели, 144
архетипы пользователей, 52, 117
составной, 109, 116
архивирование документа, 405
архитектура информационная, 215
архитектурные шаблоны проектирования, 196
ассиметричный порог смещения, 459
ассоциативное извлечение данных, 375
ассоциирование
грамматическое, 383
естественное, 288
логическое, 288
символов с объектами, 350
физическое, 286, 287
элементов управления с функциями, 286
атрибуты, 373
как основа системы извлечения данных, 375, 380
аудит
и ввод данных, 421
и редактирование, 422
продукта, 90

Б

базы данных
запросы, 383
извлечение информации, 379
критерии полноты, 275
баланс и симметрия в дизайне, 347
Батлер, Джилл («Universal Principles of Design»), 355
Безье маркеры и кривые, 469
Бейер, Хью
«Contextual Design», 91, 119, 143, 155
контекстное исследование, 90
о ролях пользователей, 119
объединенные модели, 119–120
ремесленническая модель обучения, 91
Беккер, Мэрион Ромбаур («The Joy of Cooking»), 626
белая доска, 170
Бертен, Жак («The Semiology of Graphics»), 338
библиотека шаблонов, 197
библиотечный каталог, 373
бизнес-решения, 631
бизнес-требования, 161
бизнес-цели, 44, 131
бинарные кнопки-значки, 500
блокировка элементов управления на панели инструментов, 560
боковые панели, 576
бокс ограничивающий, 472
борьба
с беспорядком, 355
с визуальным шумом, 355
с диалоговыми окнами, перегруженными элементами управления, 494, 566
с дребезгом при перетаскивании, 456, 459, 460
с иерархиями в пользовательских интерфейсах, 291
с лишними вопросами, 260, 300
с ненужными сообщениями, 257
с режимами, 478
с чистым листом, 258
«Бразилия» (фильм), 64
бренд
принципы, 175
требования, 161
брендинг, 322, 354
и опыт потребителей, 354
и пользовательский интерфейс, 354
непротиворечивость, 354
согласованность, 354
булева логика, 66, 252
буфер удаленных данных, 396
бытовые устройства
и временный тип приложения, 241
проектирование, 240

В

Вандер Вал, Томас, фолксономия, 378
варианты использования и сценарии с участием персонажей, 150
варианты ключевых сценариев, 173
ввод
виртуальные клавиатуры, 235
графический, 255
данных
аудит и редактирование, 422
в списки, 511
и производительность, 420
и эффективность, 418, 424
не по порядку, 595
недостающих, 595
немодальная обратная связь, 420
ошибки, 422
сложность, 229, 235
совмещение с выводом, 274
вдохновляющие плакаты, 128–130, 140
«Веб-дизайн: книга Стива Круга», 216
веб-приложения, 218
проектирование, 220
веб-проектирование, 214
веб-сайт, 215
информационный, 215
сервисный, 217
векторы
ввода, 56
запоминания, 618
командные, 615
множественные, 615
ментальные, 618
незамедлительные, 616
обучающие, 616
окружения, 618
вербальный тип мышления, 173
Верман, Ричард Сол, информационная тревога, 355
верньер, 460
вероятное, отделение от возможного, 401
Верпланк, Билл, 22
версии
документа, 412
создание, 396
вертикальная осевая симметрия, 347
верхнего уровня окна, 490
вершинные маркеры, 468
вечные середняки, 75
вживание в роль, 116
взаимное исключение, 444
взаимодействие
инфраструктура, определение, 55
опыт, 23–25, 175
прагматичное, 194
проектирование, 43
и повествование, 146
и эффективность, 47
практика, 630
принципы, 27, 189
происхождение термина, 22, 25
процессы, 27
целенаправленное, 193
шаблоны, 27, 55, 189, 196
элегантность, 247
с пользователем и модель представления механической эры, 68
взаимоисключающие кнопки, 502
Вид, меню, 543
видение продукта, 85
видимость (findability), 214
видимые иерархические меню, 535
видоизменение интерфейса, степень, 290
визуализация
интерфейса, 169, 180
поведения, 351
визуальная инфраструктура, 56, 163
определение, 174
создание, 174
визуальная обратная связь, 241
выделение, 448
и монопольные интерфейсы, 206
и непосредственное манипулирование, 436
при перетаскивании, 451, 454
визуальные
идиомы, 322
метафоры, 478
налоги, 269
элементы в дизайне, обзор, 349
визуальный анализ, 174
визуальный дизайн
единство, 365
интерфейсов, 333, 339
борьба с визуальным шумом и беспорядком, 355
для портативных и прочих устройств, 360
и живопись, 334
и коммуникация, 334
и прочие дисциплины проектирования, 334

и текст, 357
использование изображений, 349
принципы, 339
простота, 356
свойства
иерархия, 340
ориентация, 339
оттенок, 338
размер, 337
расположение, 339
текстура, 339
форма, 337
цвет, 338, 340, 358
яркость, 338
стиль и функция, интеграция, 352
строительные блоки, 336
структура и логические маршруты, создание, 343
визуальный информационный дизайн, 336
визуальное сравнение, 362
демонстрация причинно-следственных связей, 363
объединение текста, графики и данных, 363
отображение
изменений во времени, 362
множественных величин, 363
числовых данных, 365
принципы, 361
содержание, 363
визуальный стиль монопольных приложений, 205
визуальный тип мышления, 173
визуальный шум, борьба, 355
визуальный язык, 349, 359
Вин, Джеффри («The Art and Science of Web Design»), 216
виртуальные клавиатуры и ввод, 235
виртуальные рабочие столы, 481
вкладки
в диалоговых окнах, 586
многострочные, 588
палитр, 278
панелей, 277, 482
внедренные объекты и функция отмены, 387
внешние воздействия на документы, запоминание, 311
возможное, отделение от вероятного, 401
вопросы
для этнографических интервью, 99
наводящие, 102
ориентированные на мировоззрение, 100
ориентированные на рабочий процесс, 100
системоориентированные, 100
целеориентированные, 100
лишние, 260
избегание, 300
борьба, 260, 300
восприятие пользователем времени ответа программы, 264
восстановление окна, 490
вращение
камеры, 474
орбитальное, 474
объектов, 474
вред, минимизация, 192
временное приложение
память, 212
понятность, 210
простота, 210
временный тип приложений, 208
и налоги, 269
временный тип приложения
и бытовые устройства, 241
и веб-приложения, 221
и информационные веб-сайты, 217
и киоски, 236
и налоги, 269
и портативные устройства, 232
время
ответа программы, восприятие пользователем, 264
простоя, использование, 297, 304
Всемирная паутина, проектирование, 214
всплывающая справка, 558
всплывающие окна, 232
всплывающие подсказки, 80, 524, 559
Вставка, меню, 543
вставки, режим, 525
вставки, точка, 446
встроенные системы, общие принципы проектирования, 223
второстепенный персонаж, 142
выбор
кандидатов для этнографических интервью, 93
места хранения файла, 403
выбора запоминание, 308

вывод
в текстовых полях ввода, 526
на естественном языке, 382
выводы, запоминание, 310
выдвижные панели, 530
выделение
визуальная обратная связь, 448
вставка и замещение, 446
групповое, 445
дискретное и непрерывное, 443
и порядок команд, 441
индикация визуальная, 448
кумулятивное, 445
маркеры, 448
область, 448
выключатель, 500
выравнивание графических элементов, 343
выявление
значимых шаблонов поведения, 135
ожиданий персонажей, 156
поведенческих переменных, 134
требований, 160
целей из качественных данных, 124

Г

галочки и пункты меню, 549
Гамма Эрих, шаблоны проектирования в программировании, 55
гармоничное взаимодействие, проектирование
следование ментальным моделям, 246
стратегии, 245
Гейтс, Билл, 477, 480
геодемографические кластеры PRIZM, 104
гибкость в отношении данных, 419
Гиллиам, Терри (режиссер), 64
гиперссылки, 280, 498
главные окна, 483
глагол – объект, порядок, 441
Гладуэлл, Малкольм («Blink», «Озарение. Сила мгновенных решений»), 124
глобализация, 624
глобальные метафоры, 323
Голдберг, Руб, 325
горячая точка, 437
Градиент, инструмент Adobe Photoshop, 279
грамматическое ассоциирование, 383
грамотность компьютерная, 59
графика, объединение с текстом и данными, 363
графический ввод, обеспечение, 255
графический обзор, 285
графического дизайна инфраструктура, 174
Гринвуд, Уэйн, сценарии, основанные на персонажах, 153
Грудин, Джонатан, 116
группировка элементов
в визуальном дизайне интерфейса, 340
в общей инфраструктуре взаимодействия, 168
пространственная, 342
групповая множественная отмена, 391
групповое выделение, 445
Гудвин, Ким
процесс разработки персонажей, 133
сценарии, основанные на персонажах, 153
ценности проектирования взаимодействия, 191
гуманный интерфейс, 22

Д

Дабберли, Хью, ценности проектирования взаимодействия, 191
данные
ввод
аудит и редактирование, 422
в списки, 511
и производительность, 420
и эффективность, 424
не по порядку, 595
недостающих, 595
немодальная обратная связь, 420
ошибки, 422
дискретные, 443–445
извлечение
ассоциативное, 375
в цифровом мире, 374
и вывод на естественном языке, 382
по атрибутам, 375
по местоположению, 372
по названию, 375
по указателю, 373
позиционное, 375
цифровые методы, 375

качественные, как источник целей, 124
недостающие, 419, 595
непрерывные, 443
обработка недопустимых, 524
объединение с текстом и графикой, 363
отображение числовых, 365
поиск, 371
удаленные, буфер, 396
хранение
в цифровом мире, 374
по местоположению, 372
целостность, 417
числовые, отображение, 365
движение камеры, 474
двойной с перетаскиванием, 435
двойной щелчок, 434
действия
и модальные инструменты, 462
модифицирующие, 387
отмена, 394
потенциально опасные, 544
процедурные, 388
Демарко, Том («Человеческий фактор: успешные проекты и команды»), 243
поток, 243
демографические переменные, 94, 134
демография рынка, 104
демонстрации и рассказы в этнографических интервью, 101
демонстрация
причинно-следственной связи, 363
решений проектирования заинтересованным лицам, 175
деревья, элементы управления, 514
десятичная классификация Дьюи, 373
детализация, стадия целеориентированного проектирования, 179
деятельность
и телевизионные интерфейсы, 239
и цели пользователей, 45
уровень опыта, 74
Джаррет, Джим, 23
Джобс, Стив, 477
диагональная осевая симметрия, 347
диаграммы последовательностей, 143
диалоговые окна, 79, 212
борьба с перегрузкой элементами управления, 566
вкладки, 586
заголовки, 569
замена обогащенной немодальной обратной связью, 608
и новички, 79
и обучение, 568
и поток, 603
и рабочий процесс, 567
информирующие, 584
блокирующие, 585
временные, 585
и подтверждения, 585
и сообщения об ошибках, 585
как временные приложения, 212
как модальный способ связи, 251
каскадные, 590
модальные, 570
для приложения, 570
для системы, 570
немодальные, проблемы, 571
отказ пользователя, 274
перегруженные элементами управления, 494
подтверждений, 605
и человеческое поведение, 606
процессов, 580
разворачивающиеся, 589
рекомендации, 568
свойств, 579, 586
содержимое, управление, 586
сообщения
о нормальном состоянии дел, 258
об ошибках, 592
сравнение с комнатами, 483
терминальная команда, 569, 573
уведомлений, 602
уместное применение, 566
функциональные, 579, 586
дизайн
визуальная эффективность, 356
визуальный, единство, 365
и пользовательский интерфейс, 335
интерфейсов визуальный, 333, 339
борьба с визуальным шумом и беспорядком, 355
для портативных и прочих устройств, 360
и живопись, 334
и коммуникация, 334
и прочие дисциплины проектирования, 334
и текст, 357
использование изображений, 349
принципы, 339

дизайн интерфейсов визуальный
простота, 356
свойства
иерархия, 340
ориентация, 339
оттенок, 338
размер, 337
расположение, 339
текстура, 339
форма, 337
цвет, 338, 340, 358
яркость, 338
стиль и функция, интеграция, 352
строительные блоки, 336
структура и логические маршруты, создание, 343
информационный визуальный, 336
визуальное сравнение, 362
демонстрация причинно-следственных связей, 363
объединение текста, графики и данных, 363
отображение
изменений во времени, 362
множественных величин, 363
числовых данных, 365
принципы, 361
содержание, 363
промышленный, 336
«Дизайн для реального мира» (Папанек), 34
«Дизайн привычных вещей», «Дизайн промышленных товаров» (Норман), 329, 617
динамические визуальные подсказки, 438
диски
назначение, 414
помощь, 309
файловые системы хранения, 374
дискретное выделение, 443
дискретные данные, 443
дистанционное управление, 238, 393
пульт Logitech Harmony, 238
дневниковые исследования, 106
документоориентированные приложения, 208
документы
архивирование, 405
закрытие, 402
именование, 403, 410
отказ от всех изменений, 412
отмена изменений, 411
переименование, 404, 410
перемещение, 410
размещение, 410
создание версий, 412
создание копий, 409
сохранение автоматическое, 408
указание формата, 411
Дональд Шён (проектировщик), 84
дополнительные цвета, 359
дополнительный персонаж, 142
дорожные указатели, создание, 282
дребезг, 457
компенсация, 457
при перетаскивании, борьба, 456–460
Дьюи десятичная классификация, 373

Е

единство
в визуальном дизайне, 365
и стандарты, 368
естественное ассоциирование, 288

Ж

желание, 195
жизненные цели, 129, 137

З

заголовки диалоговых окон, 569
задачи
и цели пользователей, 45
связность, 314
задачи-налоги, 266
заинтересованные лица, 97, 98, 175
закрытие документа, 402
Закрыть, кнопка, 574
Заливка, иструмент Adobe Photoshop, 279
замены, режим, 525
замещение, 446
замораживание, 398
замусоривание окнами, 488
запись в базе данных, 380
запоминание
векторы, 618
выбора, 308
действий третьих приложений, 311
ранее введенных данных, 311
расположения файлов, 310
шаблонов, 309

запросы к базе данных, 383
захвата проблема, 473
звуковая обратная связь, 235, 241, 611
звуковые интерфейсы
  и ментальные модели, 241
  и навигация, 242
  проектирование, 241
здравый смысл тактичных программных продуктов, 297
знак вставки, 447
золотое сечение, 345
зрение, минимизация работы, 191

И

Ивенсон, Шелли, 147
игра в волшебство при разработке сценариев, 159
идиоматическая модель построения интерфейсов, 320
идиоматическая парадигма проектирования пользовательского интерфейса, 320
идиоматические интерфейсы, 320
идиомы
  визуальные, 322
  меню, 545
  непосредственного манипулирования, 250, 255
  построение, 327
идиосинкратически модальное поведение пользователей, 623
иерархия
  в пользовательских интерфейсах, борьба, 291
  элементов
    в визуальном дизайне интерфейса, 340
    в общей инфраструктуре взаимодействия, 168
избавление
  от сообщений об ошибках, 598
  от диалоговых окон подтверждений, 607
извлечение данных
  ассоциативное, 375
  по атрибутам, 375
  по местоположению, 372
  по названию, 375
  по указателю, 373, 381
  позиционное, 375
  система, 372
изменение размера и формы объектов, 467
изменения по времени, отображение, 362
изображения в визуальном дизайне
  ассоциирование символов с объектами, 350
  визуализация поведения, 351
изображения в визуальном проектировании функций, пиктограммы, 350
именование
  документов, 410
  файлов, 403
иммунитет информационный, 418
индикация
  выделения визуальная, 448
    глубины, 470
    ожидания, посредством курсора, 441
  состояния, 414
    на панели инструментов, 561
  хода выполнения операции, 581
инициативность тактичных программных продуктов, 297
Ино, Брайан, композитор, 611
инстинкт и обучение, 319
инструменты
  выбор и манипулирование, 250
  закраски, отмена действия, 395
  модальные, 462
  навигация, 278
  палитры, 462
    модальные, 462
  с нагруженным курсором, 464
    в Delphi, 465
    в PowerPoint, 464
    в Visual Basic, 464–465
интеграция функций в портативных устройствах, 230
интеллектуальные агенты, 628
интеллектуальные направляющие в OmniGraffle, 466
интеллектуальные продукты
  и память, 306, 310
  связность задач, 308
  циклы простоя, использование, 304
интеллектуальный телефон, 225
интерактивная оперативная справка, 627
интерактивность, 41

интервью
с заинтересованными лицами, 84
с покупателями, 87
с пользователями и потенциальными пользователями, 88
с экспертами в предметной области (ЭПО), 86
интерес к людям со стороны тактичных программных продуктов, 295
интернет-приложения, 222
интерфейс
Lotus 1-2-3, 535
автомобильный, проектирование, 239
адаптация под нужды пользователя, 289
в парадигме реализации, 316
визуализация, 169
гуманный, 22
звуковой, проектирование, 241
и ключевой персонаж, 141
игра в волшебство, 159
идиоматический, 320
командной строки, 267
метафорический, 317
многодокументный (MDI), 491–493, 543
одного документа (SDI), 491
прозрачность, 244
телевизионный, проектирование, 237
тип приложения, 201
интерфейсные принципы, 190, 191
интранет, 222
интуиция, 317, 319
информационная архитектура, 24, 215
«Информационная архитектура в Интернете» (Розенфельд, Морвиль), 216
информационная тревога, 355
информационные требования, 160
информационные элементы
группировка, 168
определение, 165
информационный веб-сайт, 215
информационный иммунитет, 418
информация
в голове, 617
в окружении, 617
запоминание выводов, 310
навигация, 280
потенциально полезная, 296
представление в контексте, 253
информирующие диалоговые окна, 584
блокирующие, 585
временные, 585
и подтверждения, 585
и сообщения об ошибках, 585
инфраструктура
взаимодействия, 163
макетирование, 169
определение, 164
проверочные сценарии для верификации решений, 173
создание, 164
создание ключевых сценариев, 171
визуальная, 163
определение, 174
создание, 174
пользовательского интерфейса, 162
определение функциональных групп и создание иерархии, 168
форм-фактор, тип приложения, способы управления, 164
функциональные и информационные элементы, 165
физическая, 163
определение, 177
создание, 177
исключительные ситуации, 115
исследования, 48, 49
дневников, 106
и персонажи, 115
и проектирование, ликвидация разрыва, 49
сопутствующая деятельность, 146
сценарии, 146
требования, 151
выработка, 153
как основа проектирования, 50
количественные, 82
рыночные, 48
методология
анализ рабочих заданий, 107
демография и сегментация рынка, 104
карточная сортировка, 106
фокус-группы, 103
юзабилити-тестирование, 105
полевые, этнографические методы, 52
пользователей, 48, 49
исследователь как роль проектировщика, 49

исследовательские киоски, 234, 237
источник перетаскивания, 451

К

календарь, 70
Калькулятор (Windows), 491
камера
вращение, орбитальное, 474
движение, 474
наезд, 474
режим облета, 474
каркас, 472
карточная сортировка, 106
каскадные диалоговые окна, 590
каскадные меню, 538, 545
каталог библиотечный, 373
каталог шаблонов, 197
качественные данные как источник целей, 124
качественные исследования, 82
аудит продукта и решений конкурентов, 90
виды, 84
значение, 82, 102
и исследования рынка, 104
и количественные исследования, 82
интервью с
заинтересованными лицами, 84
покупателями, 87
пользователями, 88
экспертами в предметной области, 86
наблюдение за пользователями, 89
обзор литературы, 90
кегль, 358
Кили Ларри (проектировщик), 41
киоски
ввод текста, 230
и контекст среды, 226
исследовательские, 234, 237
образовательные, 237
проектирование, 233
развлекательные, 237
расположение, 234
сервисные, 234, 237
клавиатурные сокращения, 550, 619
раздел оперативной справки, 626
клавиши
быстрого доступа, 550
служебные, 432
классификация десятичная Дьюи, 373
Клейн, Гэри («Sources of Power»), 124
клиенты, интеграция в процесс проектирования, 40
Клэкстон, Гай («Hare Brain, Tortoise Mind»), 124
ключевого пути, сценарии, 150
ключевой персонаж, 141
ключевые (пошаговые) маршруты, 56
ключевые сценарии
варианты, 173
создание, 171
кнопка отпущена и кнопка нажата, события, 436
кнопки
в Mac OS X, 496
в Windows, 496
взаимоисключающие, 502
как элемент управления, 496
мыши, 430
левая, правая, средняя, 431
кнопки-значки, 556
бинарные, 500
в Microsoft Office, 497, 500
и динамические визуальные подсказки, 438
как векторы запоминания, 620
как командные элементы управления, 496
комбо-кнопки, 503
подписи, 557
радиокнопки, 502
фиксация состояния, 500
когнитивная обработка, 124
аналитический уровень, 125
поведенческий уровень, 125
физиологический уровень, 124
когнитивная работа, минимизация, 191
когнитивный диссонанс, 65
колесо прокрутки, 432
количественная информация, представление в контексте, 253
количественные исследования, 82
количественные рыночные исследования, 48
коллекция шаблонов, 625
колокола кривая, 74
команда
для этнографических интервью, 97, 102
проектировщиков, размер, 92
терминальная, 569, 573

командной строки интерфейс, 267, 532
командные векторы, 615
множественные, 615
командные элементы управления, 494–495
гиперссылки, 498
кнопки, 496
кнопки-значки, 496
комбо-кнопки, 503
комбо-списки, 526, 561
на панели инструментов, 561
коммуникация и визуальный дизайн интерфейсов, 334
компенсация дребезга, 457
компетентность в предметной области, 95
композиция, 328
компьютер
интерактивность, 41
как подчиненный человека, 418
как представление о продукте, 224
компьютерная грамотность, 59
конечные цели, 129, 137
конкурирующие продукты, аудит, 90
Константайн, Ларри
«Software for Use», 75
о проектировании для середняков, 75
о ролях пользователей, 119
контекст
как сила, направляющая проектирование, 226
определение, 160
проектирование под цели, 47
контекстное исследование, 91
контекстное представление информации, 253
контекстные
меню, 537
панели инструментов, 565
сценарии, 129, 150
на стадии определения инфраструктуры, 55
пример, 158
разработка, 157
контраст, 359
конфликт интересов в процессе разработки, 39
концептуальная модель, 60
концептуальные принципы, 190
координатная сетка, 470
копия документа, создание, 409
Корзина в Windows, 605, 607, 608
Корман, Джонатан, ценности проектирования взаимодействия, 191
кривые Безье, 469
критический анализ (cognitive walkthrough), 129
Кристофер, Александер, 367
Кронин, Дэйв, сценарии, основанные на персонажах, 153
Круг, Стив («Don't Make Me Think!», «Веб-дизайн: книга Стива Круга»), 216
круговой манипулятор, 519
кумулятивное выделение, 445
Кунявски, Майк («Observing the User Experience»), 103
Купер, Алан («The Inmates Are Running the Asylum», «Психбольница в руках пациентов»), 30, 112
GettyGuide, киоск, 233
интеллектуальный телефон для рабочего стола, проект, 225
курсор
горячая точка, 437
и выделение, 441
и указание, 437
курсорные подсказки, 432, 439, 452
Кэрролл, Джон
«Making Use», 148
сценарный подход к проектированию, 147–149

Л

левая кнопка мыши, 431
лексикон взаимодействия, 327, 328
в графическом интерфейсе пользователя, 327
неделимые элементы, 327
лента (ribbon), 546, 564
Лидвел, Уильям («Universal Principles of Design»), 355
ликвидация разрыва между исследованиями и проектированием, 49
выработка требований, 153
сопутствующая деятельность, 146
сценарии, 146
требования, 151
Листер, Тимоти («Peopleware: Productive Projects and Teams», «Человеческий фактор: успешные проекты и команды»), 243

поток, 243
логика булева, 66, 252
логические маршруты
графических элементов, 343
создание, 346
логическое ассоциирование, 288
логическое масштабирование, 280
локализация, 624
Локвуд, Люси («Разработка программного обеспечения»), 75
Лорел, Бренда («Computers as Theater»), 147
о контексте, 47
о метафорах, 325

М

макетирование общей инфраструктуры взаимодействия, 169
максимизация представления документа, 208
Маллет, Кевин («Designing Visual Interfaces»), 248, 334, 340
манипулирование непосредственное, 425
выделение, 441
и визуальная обратная связь, 436
и меню, 538
и панели инструментов, 538
инструменты палитры, 462
курсор, 437
перетаскивание, 449
связывание объектов, 474
устройства указания, 427
элементами управления, 461
манипулятор круговой, 519
маркеры, 448, 464, 467
Безье, 469
в Microsoft PowerPoint, 467
вершинные, 468
«марширующие муравьи», 448
маршрут логический, создание, 346
мастер–подмастерье, модель обучения, 91
мастера (wizards), 627
масштабирование, 280
логическое, 280
объектов, 474
пространственное, 280
математическое мышление и модель реализации, 66
межсеансная отмена, 311
Мейер, Сид, игра «Civilization», 609
Менеджер файлов (Microsoft Windows 3.x), 253–255
ментальная модель, 60
и гармоничное взаимодействие, 246
и звуковые интерфейсы, 241
и модель представления, 61
и модель реализации, 61, 405
и новички, 78
и ожидания персонажа, 156
и ошибки, 385
и пользовательский интерфейс, 62
и системы хранения, 291
файловой системы, 405, 406
ментальные векторы, 618
меню, 537
адаптивные, 546
Вид, 543
Вставка, 543
и навигация, 278, 284
и непосредственное манипулирование, 538
идиомы, 545
иерархические
видимые, 535
последовательные, 533
история, 532
как средство обучения, 538
каскадные, 538, 545
моментального действия, 548
на других платформах, 552
на панели инструментов, 555, 561
недоступные элементы, 549
необязательные, 543
Окно, 543
Правка, 541, 543
раскрывающиеся, 537
режим, 461
Сервис, 544
Справка, 541, 543, 619
стандартные, 541
строка, 537
Файл, 408, 412, 541, 542
Формат, 544
щелчок и перетаскивание, 461
место проведения этнографических интервью, 99
метафорическая модель построения интерфейсов, 317
метафорическая парадигма проектирования пользовательского интерфейса, 317

метафорические интерфейсы, 317
метафора в пользовательском интерфейсе
визуальная, 478
глобальная, 323
для системы хранения, 291
и налоги, 269
ограничения, 318
подбор, 322
проблемы, 69, 315
расширение, 324
метки, 377
методология исследований
анализ рабочих заданий, 107
демография и сегментация рынка, 104
карточная сортировка, 106
фокус-группы, 103
юзабилити-тестирование, 105
методы проведения этнографических интервью, 98
минималистический подход к проектированию продуктов, 246
минимальные рабочие наборы, 616
минимизация
вреда, 192
когнитивной работы, 191
количества
окон, 281
панелей, 281
представлений, 281
элементов управления, 281
мнемонической работы, 191
прокрутки, 282
работы зрения, 191
сложности ввода, 229
трудозатрат посредством поведенческих и интерфейсных принципов, 191
физической работы, 191
мнемоники, 552, 619
мнемоническая работа, минимизация, 191
многодокументный интерфейс (MDI), 491–493, 543
многострочные вкладки, 588
множественная отмена, 388, 389
ограничения, 389
проблемы модели, 390
множественные
величины, 363
командные векторы, 615
ракурсы, 470
множество решений, сокращение, 312
модальные диалоговые окна, 570
модальные инструменты, 462
в Adobe Illustrator, 463
в Adobe Photoshop, 462, 463
моделирование, стадия целеориентированного проектирования, 110
модель, 109, 110
артефактов, 144
бизнес-процессов, 143
концептуальная, 60
ментальная и ошибки, 385
назначение, 110
обучения ремесленническая, 91
построения интерфейсов
в парадигме реализации, 316
идиоматическая, 320
метафорическая, 317
представления, 61
и ожидания персонажа, 156
информационной эры, 68
механической эры, 67
функции отмены, 390, 393
реализации, 49, 59, 62
демонстрация пользователям, 605
и математическое мышление, 66
и ментальная модель, 61, 405
и ошибки, 385
и пользовательский интерфейс, 64
и программное обеспечение, 64
и файловая система, 399, 402, 416
роль пользователя, 119
профили пользователей, 120
системная, 60
файловой системы унифицированная, 408
физическая, 144
модифицирующие действия, 387
отмена, 394
модульность сетки, 345
моментального действия, меню, 548
моноклинальная группировка, 291, 292
монопольный тип приложения, 203
визуальная обратная связь, 206
и ввод, 206
и веб-приложения, 220
и визуальный стиль, 205
и информационные веб-сайты, 216
и киоски, 236
и налоги, 269
и портативные устройства, 233

и развернутые окна, 491
и середняки, 203
и симметрия, 347
и экранное пространство, 205
Морвиль, Питер
«Information Architecture», «Информационная архитектура в Интернете», 216
«Ambient Findability», «Тотальная видимость. Как наши находки меняют нас», 214
мотивация, 118, 130
мышление
вербальный тип, 173
визуальный тип, 173
компьютерный стиль, 37
математическое, 66
мышь
использование, 428
кнопки, 430
левая, правая, средняя, 431
перетаскивание, 434
снижение чувствительности, 460
указание, 432, 433
щелчок, 432, 433, 435
аккордный, 435
двойной, 434

Н

наблюдение за пользователями, 89
наборы рабочие, 616
минимальные, 616
навигатор документа, 529
навигационная травма, 276
навигация, 281
в телевизионных интерфейсах, 237
гиперссылки, 280
и звуковые интерфейсы, 242
и интеграция функциональности, 230
и меню, 284
и панели инструментов, 284
и плотность информации, 228
как налог, 275
масштабирование, 280
между инструментами и меню, 278
между панелями, 276
между представлениями, 276
между страницами, 276
между экранами, 276
мини-карта, 280
мышь и клавиатура, 430
панорамирование, 280
по информации, 280
прокрутка, 280
улучшение
адаптация интерфейса под нужды пользователя, 289
ассоциирование элементов управления с функциями, 286
дорожные указатели, 282
отказ от иерархий, 291
при помощи обзоров, 284
сокращения количества пунктов назначения, 281
наводящие вопросы, 102
наглые налоги, 269
наезд камеры, 474
название как основа системы извлечения данных, 375
Найти и заменить, диалоговое окно (Word), 589
налоги, 266
в графическом пользовательском интерфейсе, 267
визуальные, 269
выявление, 271
и временный тип приложений, 269
и метафоры в пользовательском интерфейсе, 269
и монопольный тип приложений, 269
и навигация, 275
и неопытные пользователи, 268
и опытные пользователи, 268
наглые, 269
необходимость просить разрешения, 273
распространенные ловушки, 274
напольная сетка, 471
направляемое перетаскивание, 466
направляющие, 466, 471
в Illustrator, 466
интеллектуальные в OmniGraffle, 466
Нарди, Бонни (проектировщица), 46
нарушение стандартов, 367
наследование, 356
Насс, Клиффорд («The Media Equation»), 261, 293, 294
настольное программное обеспечение, проектирование, 202
настройка
панелей инструментов, 562

параметры, доступные пользователю, 620
Настройка, меню, 544
насыщенные цвета, 359
неделимые элементы лексикона взаимодействия, 327
недостающие данные, 419, 595
недоступные элементы меню, 549
незамедлительные векторы, 616
Нельсон, Тед, об идиоматическом проектировании, 320
немодальная обратная связь
в Microsoft Word, 251, 423
и ввод данных, 420
как замена диалоговых окон, 608
обеспечение, 250, 299
немодальная оперативная справка, 627
немодальные диалоговые окна, 571
проблемы, 571
решения, 572
необратимые действия, 398
неограничивающие элементы управления вводом, 515
непосредственное манипулирование, 425
выделение, 441
и визуальная обратная связь, 436
и меню, 538
и панели инструментов, 538
идиомы, 255
инструменты палитры, 462
курсор, 437
перетаскивание, 449
связывание объектов, 474
устройства указания, 427
элементами управления, 461
непрерывное выделение, 443
непрерывные данные, 443
неупорядоченный ввод данных, 595
Нильсен, Якоб
«Usability Engineering», 106, 182
useit.com, веб-сайт, 216
о стандартах, 365, 366
новички
и ментальные модели, 78
и налоги в интерфейсе, 268
и середняки, 73
перевод в середняки, 617
потребности, 78
нормали, 470
Норман, Дональд, 592
Activity-Centered Design (ACD), 45
«Emotional Design», 124, 126, 127
«The Design of Everyday Things», «Дизайн привычных вещей», «Дизайн промышленных товаров», 329, 330, 617
информация в окружении и информация в голове, 617
модель проектирования, 61
непосредственное манипулирование, 426
о естественном ассоциировании, 288
об ошибках в проектировании продукта, 592
ожидаемые назначения, 329, 330, 437
персонажи на данный случай, 122
проектирование, ориентированное на деятельность, 45
системная модель, 60
уровни когнитивной обработки, 124–127
«шкаф с бумагами», 292

О

обзор литературы, 90
обзор, функция оперативной справки, 627
обзоры
аннотированная полоса прокрутки, 285
графические, 285
создание, 284
текстовые, 285
хлебные крошки, 285
область уведомлений на панели задач Windows XP, 213
обогащенная визуальная немодальная обратная связь, 609
обогащенная немодальная обратная связь, 257
обогащенные средства ввода и монопольные интерфейсы, 206
обработка когнитивная, 124–125
аналитический уровень, 125
поведенческий уровень, 125
физиологический уровень, 124
обработка сбоев аккуратная, 300
образ продукта, определение, 154
образовательные киоски, 237

обратимость любых действий, 607
обратная связь
визуальная, 241
и непосредственное манипулирование, 436
при выделении, 448
при перетаскивании, 451, 454
звуковая, 235, 241, 611
отрицательная, 611
положительная, 613
немодальная, 250
визуальная обогащенная, 609
и ввод данных, 420
как замена диалоговых окон, 608
обогащенная, 257
отрицательная, 600, 612
звуковая, 611
положительная, 599
звуковая, 613
тактильная, 241
обслуживаемый персонаж, 119, 143
обучающие векторы, 616
обучение
и диалоговые окна, 568
и идиоматические интерфейсы, 320, 321
инстинкт, интуиция, 319
модель, 91
посредством меню, 538
общая структура
дизайна телевизионных интерфейсов, 237
проектирования визуальная, 237
общение и персонажи, 113
объединенное проектирование программной и аппаратной частей продукта, 224
объект – глагол, порядок, 441
объектные подсказки, 438
объекты
ассоциирование с визуальными символами, 350
вращение, 474
и требования, 160
изменение размеров и формы, 467
манипулирование
в трех измерениях, 469
изменение размеров и формы, 467
перемещение, 465
масштабирование, 474
отображение состояния, 256
перемещение, 465
размер, 467
связывание, 474
резиновая нить, 475
стабильные, 263, 282
трехмерные, манипулирование, 469
обязательные сценарии, 173
ограничения технические, 85
ограничивающие элементы управления вводом, 515, 520
ограничивающий бокс, 472
одиночная отмена, 388
ограничения, 389
ожидаемые назначения, 329
физические, 330
соответствие ожиданиям пользователя, 331
ожидание, индикация посредством курсора, 441
«Озарение. Сила мгновенных решений» (Гладуэлл), 124
окна
Alto, система, 476
MDI против SDI, 491
верхнего уровня, 490
восстановление, 490
всплывающие, 232
главные, 483
диалоговые, 79, 212
борьба с перегрузкой элементами управления, 566
вкладки, 586
заголовки, 569
замена обогащенной немодальной обратной связью, 608
и новички, 79
и обучение, 568
и поток, 603
и рабочий процесс, 567
информирующие, 584
блокирующие, 585
временные, 585
и подтверждения, 585
и сообщения об ошибках, 585
как временные приложения, 212
как модальный способ связи, 251
каскадные, 590
модальные, 570
для приложения, 570
для системы, 570
немодальные, проблемы, 571
отказ пользователя, 274
перегруженные элементами управления, 494

окна диалоговые
подтверждений, 605
и человеческое поведение, 606
процессов, 580
разворачивающиеся, 589
рекомендации, 568
свойств, 579, 586
содержимое, управление, 586
сообщения
о нормальном состоянии дел, 258
об ошибках, 592
сравнение с комнатами, 483
терминальная команда, 569, 573
уведомлений, 602
уместное применение, 566
функциональные, 579, 586
замусоривание, 488
и интерфейсы командной строки, 267
миниатюра, 491
минимизация количества, 281
многопанельные приложения, 482
перекрывающиеся, 479
плиткой, 480
подчиненные, 483
полномасштабные, 491
полноэкранные приложения, 481
проектирование, 483
развернутые, 490
расположение, 309
свернутые, 490
свободные, 491
смежные, 232
состояния, 490
управление, 276
Окно, меню, 543
окружение, информация, 617
окружения векторы, 618
оперативная справка, 79, 80
и новички, 79
и середняки, 80
интеллектуальные агенты, 628
интерактивная, 627
мастера, 627
немодальная, 627
перечень клавиатурных сокращений, 626
указатель, 626
функция обзора, 627
оперативные подсказки, 523
описание моделей, 109
определение инфраструктуры взаимодействия, 55
определение продукта, 49
опыт
взаимодействия, 175
пользователя, 178, 217
требования, 161
орбитальное вращение камеры, 474
организационная структура как основа интерфейсов, 317
органы управления, интеграция, 238
ориентация и визуальный дизайн интерфейсов, 339
оркестровка, 244
отвергаемый персонаж, 143
ответственность тактичных программных продуктов, 303
отзывчивость, 433
и подсказки, 437
при указании, 437
отказ от всех изменений в документе, 412
откат
команда, 397
после сбоя, 301
отклик активный, 439
визуальный, 439
отклонение от правил, 302
отмена
версии и откат, 396
вслепую, 388
действия инструментов закраски, 395
замораживание, 398
и буферы удаленных данных, 396
и внедренные объекты, 387
и повтор, 391
и пользователи, 384
и унифицированная файловая модель, 411
и файловая система, 402
изменений в документах, 411
как инструмент сравнения, 393
межсеансная, 311
множественная, 388
групповая, 391
ограничения, 389
проблемы модели, 390
модель представления, 390, 393
модифицирующие и процедурные действия, 387
необратимые действия, 398

одиночная, 388
ограничения, 389
поощрение экспериментирования, 385
поясняющая, 388
проектирование, 386
специфическая, 394
отметка, 507
отображение
изменений во времени, 362
множественных величин, 363
числовых данных, 365
отрицательная обратная связь, 600, 612
звуковая, 611
оттенок и визуальный дизайн интерфейсов, 338
ошибки
и пользователи, 385
ментальная модель, 385
модель реализации, 385
при вводе данных, 422

П

палитры плавающие, 575
память
временных приложений, 212
и диски, 414
и интеллектуальные продукты, 306
и тактичные продукты, 295
на действия, 311
наделение приложений, 311
панели
боковые, 576–578
выдвижные, 530
инструментов, 554
всплывающая справка, 558
и выбор, 260
и кнопки-значки, 496
и меню, 555, 561
и навигация в интерфейсе, 284
и непосредственное манипулирование, 538
и текст, 556
использование для индикации состояния, 561
как немодальная идиома, 575
контекстные, 565
лента (ribbon), 547, 564
настройка, 562
обучение применению элементов управления, 558
паркуемые, 562, 575
перекрывающиеся, 562
пиктограммы, 556
плавающие, 562
подвижные, 562
эволюция, 560
элементы управления, 558
минимизация количества, 281
навигация, 276
организованные как вкладки, 277, 482
смежные, 276, 277, 482
и разделители, 277, 482
стопка, 482
панель задач,480, 576, 577
панель управления (в Mac OS и Windows) 213
панорамирование, 280
Папанек, Виктор (промышленный дизайнер), 33
«Дизайн для реального мира», 34
папка с файлами, метафора, 325
парадигма проектирования пользовательского интерфейса, 316
идиоматическая, 320
метафорическая, 317
ориентированная на реализацию, 316
Параметры, диалоговое окно (Word), 588
паркуемые панели инструментов, 562, 575
партнеров требования, 161
пассивная проверка допустимости, 523
переименование
документов, 410
файлов, 404
перекрывающиеся окна, 479
перекрывающиеся панели инструментов, 562
переменные, 94, 134
демографические, 134
перемещение документов, 410
перемещение объектов, 465
пересмотр сделанного, 163
перетаскивание, 449
автоматическая прокрутка, 454
и визуальная обратная связь, 451, 454
и двойной щелчок, 435
и списки, 509
индикация целей, 453
источник, 451

перетаскивание
компенсация дребезга, 456, 459, 460
мышью, 434
на сенсорных экранах, 236
направляемое, 466
рамка выделения, 446
точная прокрутка, 460
цель, 452
потенциальная, 451, 453
персонажи, 109, 139, 149
в целеориентированном проектировании, 51
второстепенные, 142
деловые и социальные взаимоотношения, 137
дополнительные, 142
и исключительные ситуации, 115
и мотивация, 118
и пластилиновый пользователь, 113
и проектирование под себя, 114
и профили пользователей, 120
и роли пользователей, 120
и сегменты рынка, 121, 122
и стереотипы, 117
использование в сценариях, 149
как олицетворение личностей, 115
как представители групп людей, 116
как представители непользователей, 118
ключевые, 141
обслуживаемые, 119, 143
описание, 109, 139
основываются на исследованиях, 115
отвергаемые, 143
повторное использование, 117
покупателей, 118, 142
преимущества, 111, 112
рабочие наборы, 616
спектр вариантов поведения, 118
условные, 121
фотографии, 139
шаги разработки, 133
персонализация, 620
пиктограммы
в Mac OS X, 342, 350–352
в Windows, 352
Vista, 351
в меню, 550
во временных приложениях, 241
на панелях инструментов, 556
проектирование и визуализация, 351
проектирование и рисование, 349
фоновые, 213
фотореалистичные, 352
функций, 350
плавающая палитра, 575
плавающие панели инструментов, 562
плакаты вдохновляющие, 128–130, 140
планирование этнографических интервью, 96
пластилиновый пользователь, 113
платформа, 200
и меню, 552
принятие решений, 202
плотность информации во встроенных системах, 228
поведение
визуализация, 351
пользователей
идиосинкратически модальное, 622, 623
модальное, 623
проектирование, 41
радиокнопки, 502
спектр вариантов, 118
шаблоны
выявление значимых, 135
и персонажи, 110, 117
стадия исследований, 52
поведенческие переменные
выявление, 134
и гипотеза о персонажах, 93
сопоставление с респондентами, 135
поведенческие принципы, 189, 190
поведенческие шаблоны, 198
поведенческий уровень когнитивной обработки, 125
повествование, 147
и описания персонажей, 139
и сценарии, 146
повтор, функция, 391
повторное использование персонажей, 117
подбор метафоры в пользовательском интерфейсе, 322
подвешенное состояние объекта, 302
подвижные панели инструментов, 562
подготовка пользователей, параметр проектирования, 76
подписи кнопок-значков, 557
подсказки
визуальные
динамические, 438
статические, 438

всплывающие, 524, 559
и отзывчивость, 437
курсорные, 432, 439, 452
объектные, 438
оперативные, 523
Подтверждение удаления файла, диалоговое окно (Windows), 605
подтверждения
диалоговые окна, 605
и человеческое поведение, 606
избавление, 607
и информирующие диалоговые окна, 585
как налог, 272
подчиненные окна, 483
позиционирование, шаблоны, 198
позиционное извлечение, 375
познание, стимуляция, 196
покупатели
интервьюирование, 87
персонажи, 118, 142
требования, 161
цели, 131
поле в базе данных, 380
ползунок, 518
полилинии, 468
полное тестирование, 182
полномасштабные окна, 491
полноэкранные приложения, 481
положение человека, улучшение, 193
положительная обратная связь, 599
звуковая, 613
полоса прокрутки, 429, 434, 528
аннотированная, 285
в Mac OS, 429
в Windows, 434
пользователи, 86
архетип, 52, 117
составной, 109, 116
и заинтересованные лица, 86
и этнографические интервью, 101
интервьюирование, 88
исследования, 48, 49
наблюдение, 89
прогнозирование действий, 306, 314
профили, 120
роли, 119
цели
жизненные, 129, 137
и создание цифровых продуктов, 34
как мотивы, 130
конечные, 129, 137
типы, 127
эмоциональные, 128
чувствующие себя глупо, 133, 421, 594
пользователь пластилиновый, 113
пользовательский интерфейс, 64
адаптация, 289
и бренд, пользовательский опыт, 354
и дизайн, 335
и математическое мышление, 66
и ментальная модель, 62
и модель реализации, 64
и уровень подготовки, 73
как артефакт, 244
налоги, 267
парадигмы проектирования, 316
привлекательность, 127
происхождение, 326
хорошо сбалансированный, 75
пользовательский опыт, 178
и брендинг, 354
пользовательский отклик, 181
понятливость тактичных программных продуктов, 299
понятность временных приложений, 210
порог
предпочтений, 313
смещения, 456
асимметричный, 459
в трехмерной проекции, 473
портативные устройства
визуальный дизайн интерфейсов, 360
и временный тип приложения, 232
и контексты сред, 226
и монопольный тип приложения, 233
меню, 552
ограничение функциональности, 227
проектирование, 230
самостоятельные, 231
спутники, 231, 232
последовательные иерархические меню, 533
построение идиом, 327
потенциальная цель, 451
потенциально опасные действия, 544
поток, 243
и диалоговые окна, 603
и налоги, 271

потребности
бизнеса, 86
и требования, 152
почтительность тактичных программных продуктов, 295
поясняющая отмена, 388
правая кнопка мыши, 431
правила, отклонение от, 302
Правка, меню, 541, 543
прагматичное взаимодействие, 194
практика проектирования взаимодействия, 630
предварительный просмотр печати (Word), 352
предметная область
знания, 328
компетентность, 95
предпочтения пороги, 313
представления, 168
минимизация количества, 281
навигация, 276
представления модель
и модель реализации, 292
функции отмены, 390, 393
предупредительность тактичных программных продуктов, 297
предупреждение
о задержках, 264
потребностей человека, 297
приложение
временное, 208
документоориентированное, 208
единство и стандарты, 368
многопанельное, 482
модальные диалоговые окна, 570
монопольное, 203
и развернутые окна, 491
наделение памятью, 311
обнаружение внешних воздействий на документы, 311
ориентированное на работу с документами, 208
отображение состояния, 256
полноэкранное, 481
состояние, 414
тип, 201
веб-приложение, 220
временный, 208
для информационных веб-сайтов, 216
для киосков, 236
для портативных устройств, 232
для сервисных веб-сайтов, 218
монопольный, 203
настольное приложение, 202
фоновый, 212
фоновое, 212, 241
примитивы, 328
принципы
бренда, 175
взаимодействия, 167
проектирования взаимодействия, 27, 189
для встроенных систем, 223
и уровни детализации, 190
интерфейсные, 190
концептуальные, 190
минимизация трудозатрат, 191
поведенческие, 190
причинно-следственная связь, демонстрация, 363
проблема захвата, 473
проведение этнографических интервью, 97
командный анализ, 102
методы, 98
фазы, команды, график, 97
проверка допустимости в элементах управления вводом, 521
активная, 522
пассивная, 523
проверочные сценарии, 150
и стадия детализации, 56
Проводник (Windows), 254, 402, 432, 446, 452, 491, 577
автоматическая прокрутка, 456
боковые панели (Explorer Bars), 577
групповое выделение, 446
деревья, 512
индикатор хода выполнения, 582
как временное приложение, 209
кумулятивное выделение, 445
отмена файловых операций, 583
переименование файла, 404
перемещение файлов, 427
сортировка файлов, 510
списки, 512
прогнозирование действий пользователя, 306, 314
программисты
и диалоговые окна уведомлений, 604
перекладывание ответственности на пользователей, 606
сотрудничество, 631

программное обеспечение
взаимодействие, 244
и модель представления механической эры, 68
и модель реализации, 64
и уровень подготовки пользователей, 76
проектирование совместно с аппаратным обеспечением, 224
прозрачность интерфейса, 244
реакция как на разумное существо, 293
уровни навигации, 275
эволюция процесса разработки, 36
продукт, 104
аудит, 90
видение, 85
жизнеспособность, 104
как компьютер, 224
как человеческое существо, 167
определение, 49
проектирование, 111
тип интерфейса, 201
проектирование, 33, 41, 148
автомобильных интерфейсов, 239
бытовых устройств, 240
в среде Всемирной паутины, 214
веб-приложений, 220
взаимодействия, 43– 44
и повествование, 146
и эффективность, 47
практика, 630
принципы, 27, 189
происхождение термина, 22, 25
процессы, 27
целенаправленное, 193
шаблоны, 27, 55, 189, 196
элегантность, 247
встроенных систем, 223
для уровня
поведенческих реакций, 126
физиологических реакций, 125
звуковых интерфейсов, 241
интернет-приложений, 222
информационных веб-сайтов, 215
как определение продукта, 49
как продукт исследований, 50
киосков, 233
на основе контекста, 226
настольных приложений, 202
общая визуальная структура, 237
общая инфраструктура, 162
ориентирование на деятельность, 45
ориентированное на достижение целей пользователей в контексте, 47
отмены, 386
под себя, 114
портативных устройств, 230
принципы, 27, 189
для встроенных систем, 223
уровни детализации, 190
сервисных веб-сайтов, 217
сценарии, 148
телевизионных интерфейсов, 237
ценности, 190, 191
минимизация вреда, 192
прагматичное взаимодействие, 194
целенаправленность, 193
элегантность, 194
этичные решения, 192
эволюция в промышленности, 41
элегантные решения, 194
проектировщик
и диалоговые окна уведомлений, 604
и коммуникатор, 170
и юзабилити-тестирование, 184
как исследователь, 49
как роль пользователя, 101
с кем сотрудничать, 631
проектировщики, размер команды, 92
проецирование информации, 251
прозрачность интерфейса, 244
производительность
и ввод данных, 420
и поток, 243
прокрутка, 280
автоматическая при перетаскивании, 454
горизонтальная, 510
минимизация, 280, 282
точная, 460
промежуточное тестирование, 183
промышленная эволюция проектирования, 41
промышленный дизайн, 336
пропорции сетки, 345
простота
в визуальном дизайне интерфейса, 356
временных приложений, 210
пространственная группировка элементов, 342

пространственное масштабирование, 280
прототип, создание, 178, 181
профили пользователей, 120
процедурные действия, 388
процесс разработки
и цели пользователей, 39
конфликт интересов, 39
успешный, 42
эволюция, 36
процессы проектирования взаимодействия, 27
Прюит, Джон, 116
«Психбольница в руках пациентов» (Купер), 30, 112

Р

работа
в полный экран, монопольное приложение, 205
зрения, минимизация, 191
с портативными устройствами, 230
рабочая среда, переменные, 96
рабочие наборы, 79–80, 616
минимальные, 616
рабочий процесс и диалоговые окна, 567
рабочий стол
Macintosh, 324, 557
виртуальный, 481
радиокнопки, поведение, 502
радиокнопки-значки, 502
развернутые окна, 490
развлекательные киоски, 237
разворачивание на весь экран монопольного приложения, 203
разворачивающиеся диалоговые окна, 589
разделение труда в век компьютеров, 293, 418
разделители
и смежные панели, 277, 482
подвижные, 529
размер
и визуальный дизайн интерфейсов, 337
команды проектировщиков, 92
объектов, 467
шрифта, 358
размещение документов, 410
разработка
контекстных сценариев, 157
персонажей
выявление значимых шаблонов поведения, 135
выявление поведенческих переменных, 134
назначение типов, 141
проверка полноты и избыточности, 138
расширение описаний атрибутов и поведений, 139
синтез характеристик и соответствующих целей, 136
сопоставление респондентов с поведенческими переменными, 135
шаги, 133
процесс
и цели пользователей, 39
успешный, 42
эволюция, 36
«Разработка пользовательских интерфейсов» (Тидвелл), 197–198
разрешение экрана, 360
ракурсы множественные, 470
рамка выделения, 446
ранее введенные данные, запоминание, 311
раскадровка, 147, 163, 172
раскрывающееся поле со списком, 526
раскрывающиеся меню, 537
раскрывающийся список, элемент управления, 506
расписание этнографических интервью, 97
расположение
и визуальный дизайн интерфейсов, 339
киосков, 234
файлов, запоминание, 310
рассказывание историй, 101, 147
расширение метафор, 324
реакция, оптимизация скорости, 264
реализации модель, 49
демонстрация пользователям, 605
и математическое мышление, 66
и ментальная модель, 61
и ошибки, 385
и пользовательский интерфейс, 64
и программное обеспечение, 64
и файловая система, 399, 402, 416
режим
вставки, 525

замены, 525
меню, 461
облета, для камеры, 474
режимы
борьба, 478
и поведение, 227, 239
резиновая нить, 475
Рейман, Роберт (проектировщик)
процесс разработки персонажей, 133
сценарии, основанные на персонажах153
ценности проектирования взаимодействия, 191
Рейнфранк, Джон, 147
ремесленническая модель обучения, 91
респонденты, сопоставление с поведенческими переменными, 135
Ривз, Байрон («The Media Equation»), 261, 293, 294
риска, степень, 290
Робин, Джонатан, геодемографические кластеры PRIZM, 104
Розенфельд, Луис («Information Architecture», «Информационная архитектура в Интернете»), 216
роли
для корпоративных и потребительских рынков, 94
пользователей, 119
Ромбаур, Ирма («The Joy of Cooking»), 626
Рубенкинг, Нил, о функции отмены, 390
рукоятка, 518
рынка, сегментация, 48, 121, 122
рычажки, 531

С

самостоятельное устройство, 231
Сано, Дэррел (Designing Visual Interfaces), 248, 334, 340
свернутые окна, 490
свободные окна, 491
свойств, диалоговые окна, 579, 586
Свойства диска, диалоговое окно (Windows), 365
связность задач, 308, 314
связывание объектов, 474
резиновая нить, 475
сговорчивость системы, 302, 421
сговорчивость тактичных программных продуктов, 302
сегментация рынка, 48, 104
VALS, 104
сегменты рынка, 121, 122
сенсорные экраны, 229, 235
Сент-Экзюпери, Антуан де, 195, 356
Сервис, меню, 544
сервисные веб-сайты, 217
сервисные киоски, 234, 237
середняки, 73
вечные, 75
и метафоры, 318
и панели инструментов, 284
и приложения монопольного типа, 203
и соразмерность усилий, 289
и юзабилити-тестирование, 106
перевод из новичков, 617
потребности, 80
Сесил, Рик, 23
сетка, 344
и удобство применения, 346
и эстетика, 346
и эффективность, 346
координатная, 470
модульность, 345
напольная, 471
пропорции, 345
символы, ассоциирование с объектами, 350
симметрия
в приложениях монопольного типа, 347
вертикальная осевая, 347
диагональная осевая, 347
и визуальный баланс, 347
система
архивации и передачи изображений, 222
извлечения данных, 372
сговорчивая, 421
хранения, 291, 372
системоориентированные вопросы для этнографических интервью, 100
ситуации исключительные, 115
сценарии, 174
скорость реакции, оптимизация, 264
Скрепыш (Microsoft), 77, 296, 628
служебные клавиши, 432
смежные окна, 232
смежные панели, 276, 277, 482

смещения порог, 456
асимметричный, 459
в трехмерной проекции, 473
Смит, Джиллиан Крэмптон (проектировщик), 51
Снайдер, Кэролин («Paper Prototyping»), 163, 183
снижение чувствительности мыши, 460
содержимое диалоговых окон, управление, 586
создание
версий документа, 396, 412
копий документа, 409
сокращение множества решений, 312
сокращения клавиатурные, 550, 619
сокрытие функций, вызывающих
визуальные изменения, 263
необратимые действия, 263
сообщения
ненужные, борьба, 257
об ошибках
большое количество, 593
грубые, 36, 37
и информирующие диалоговые окна, 585
и связанные с ними проблемы, 593
избавление, 597
как налог, 272
ненужные диалоговые окна, 249
при переименовании файлов, 404
улучшение, 601
сопоставление респондентов с поведенческими переменными, 135
соразмерные усилия, 289
сортировщик слайдов (PowerPoint), 447, 454
составной архетип пользователей, 109, 116
состояние
индикация, 414
объекта, подвешенное, 302
отображение, 256
сотовые телефоны, 232
сотрудничество проектировщиков, 631
сохранение
автоматическое, 408
документов и отмена, 386
изменений файла, 400
Сохранение документа, диалоговое окно, 403
Сохранить изменения, диалоговое окно (Word), 400
спектр вариантов поведения пользователей, 118
спецификация формы и поведения, 56
специфическая отмена, 394
списки, элемент управления, 505
ввод данных, 511
горизонтальная прокрутка, 510
отметка, 507
перетаскивание элементов, 509
сортировка, 510
Список, диалоговое окно (Word), 348
способы управления
определение, 164
сотрудничество с проектировщиками взаимодействия, 177
справка
всплывающая, 558
оперативная
интеллектуальные агенты, 628
интерактивная, 627
мастера, 627
немодальная, 627
перечень клавиатурных сокращений, 626
функция обзора, 627
Справка, меню, 541, 543
информация о клавиатурных сокращениях, 619
спутник, устройство, 231, 232
сравнение
подчеркнутое визуальное, 362
функция, 393
среда
для этнографических интервью, 99
контекст, 226
средняя кнопка мыши, 431
средства ввода обогащенные, 206
стабильные объекты, 263, 282
стадия
выработки требований в целеориентированном проектировании, 54, 151
выявление ожиданий персонажей, 156
выявление требований, 160
персонажи и сценарии, 153
последовательность действий, 153
разработка контекстных сценариев, 157
детализации в целеориентированном проектировании, 56, 179

исследований в целеориентированном проектировании, 52
моделирования в целеориентированном проектировании, 52
определения инфраструктуры в целеориентированном проектировании, 55
сопровождения разработки в целеориентированном проектировании, 57
стандартные меню, 541
стандарты
Adobe, 365
Apple (Macintosh), 365, 426, 432, 449, 541
Microsoft (Windows), 365, 431, 432, 449, 452, 541, 619
Motif, 541
выгоды применения, 365
как рекомендации или общие правила, 366
нарушение, 367
распространяющиеся на несколько приложений, 368
риски, 366
Статистика, диалоговое окно (Word), 251
статические визуальные подсказки, 438
стек LIFO, 389, 390
степень риска, 290
стереотип пользователей, 117
стиль, интеграция с функциональностью, 352
стопки панелей, 482
страницы, навигация, 276
стратегия визуального языка, 56
строка меню, 537
структура дизайна телевизионных интерфейсов, общая, 237
структурирование визуальных элементов, 343
структурные шаблоны, 198, 199
пример Microsoft Outlook, 199
сценарии, 146
в проектировании, 148
и варианты использования, 150
исключительных ситуаций, 174
использование персонажей, 149
ключевого пути, 150
ключевые варианты, 173
создание, 171
контекстные, 129, 150
на стадии определения инфраструктуры, 55
пример, 158
разработка, 157
обязательные, 173
проверочные, 150
для верификации решений, 173
разновидности, 150
счетчики, 517

Т

Табуляция, команда (Word), 568
тактильная обратная связь, 241
тактичные продукты
аккуратная обработка сбоев, 300
борьба с лишними вопросами, 300
возможное отклонение от правил, 302
демонстрация здравомыслия, 297
инициативность, 297
информирование пользователя, 299
качества, 294
обзор, 294
ответственность, 303
понятливость, 299
почтительность, 295
предупредительность, 297
проявление интереса к человеку, 295
сговорчивость, 302
уверенность в себе, 300
умолчание о своих проблемах, 298
услужливость, 296
Тафти, Эдвард, 363
«The Visual Display of Quantitative Information», 336, 361
о визуальном дизайне, 361
о представлении количественных данных, 254
принципы визуального представления информации, 361–362
теги, 377
текст
в визуальных интерфейсах, 357, 363
на панелях инструментов, 556
чтение, 357
текстовые поля ввода
единицы измерения и размеры, 525
использование для вывода, 526
обработка недопустимых данных, 524

оперативные подсказки, 523
проверка допустимости, 521
режимы вставки и замены, 525
текстовые элементы управления, 527
текстовый обзор, 285
текстура и визуальный дизайн интерфейсов, 339
телевизионные интерфейсы
и цели, 239
проектирование, 237
тени, 470
теория деятельности, 45
терминальная команда, 569, 573
Теслер, Ларри, отказ от режимов, 478
тест с прищуриванием, 343
тестирование
полное, 182–183
промежуточное, 182–183
техническая компетентность, 95
технические требования, 161
технические цели, 132
Тидвелл, Дженифер («Designing Interfaces», «Разработка пользовательских интерфейсов»), 197
тип
интерфейса, 200
определение, 165
мышления
вербальный, 173
визуальный, 173
приложений, 165
веб-приложение, 220
временный, 208
для информационных веб-сайтов, 216
для киосков, 236
для портативных устройств, 232
для сервисных веб-сайтов, 218
монопольный, 203
настольные приложения, 202
фоновый, 212
типовые экраны, применение выбранного визуального стиля, 177
Тогнаццини, Брюс, 23
«Тотальная видимость. Как наши находки меняют нас» (Морвиль), 214
точка вставки, 447
точка курсора горячая, 437
точная прокрутка, 460
травма навигационная, 276
требования
бизнеса, 161
бренда, 161
информационные, 160
опыта, 161
партнеров, 161
покупателей, 161
технические, 161
функциональные, 160
трехмерные объекты, манипулирование, 469
триггеры в меню, 549
Тэйбор, Филип (проектировщик), 51

У

уведомления, диалоговые окна, 602
уведомлений, область (Windows XP), 213
уверенность в себе тактичных программных продуктов, 300
удобство применения и сетка, 346
указание
мышью, 433
при помощи курсора, 437
указатель
как средство извлечения данных, 373, 381
оперативной справки, 626
украшательство излишнее, 270
Улица Сезам, передача, 337
улучшение
положения человека, 193
сообщений об ошибках, 601
умолчания, запоминание, 308
унифицированная файловая модель, 408
автоматическое сохранение, 408
именование и переименование, 410
индикация состояния, 414
отказ от всех изменений, 412
отмена изменений, 411
размещение и перемещение, 410
создание версий, 412
создание копий, 409
указание формата хранения, 411
упорядоченный список, 510
управление дистанционное, 238, 393
управления панель (в Mac OS и Windows), 213
уровень подготовки и пользовательский интерфейс, 73
усиление, 356
условный персонаж, 121

услужливость тактичных программных продуктов, 296
устройства
самостоятельные, 231
спутники, 231, 232
указания, 427
участники разработки, сотрудничество, 632

Ф

Файл, меню, 541, 542
изменение названия и содержания, 408, 412
файловая система
и модель реализации, 402, 416
и отмена, 402
и сохранение изменений, 400
ментальная модель, 405, 406
проблемы, 399
унифицированная файловая модель, 408
файлы
выбор места хранения, 403
именование, 403, 410
переименование, 404, 410
перемещение, 410
помощь, 309
размещение, 410
создание копий, 409
сохранение автоматическое, 408
фигуры и модальные инструменты, 462
физиологический уровень когнитивной обработки, 124
физическая инфраструктура, 163
определение, 177
создание, 177
физическая работа, минимизация, 191
физические модели, 144
физические ожидаемые назначения, 330
соответствие ожиданиям пользователя, 331
физическое ассоциирование, 286, 287
фиксация состояния кнопки-значка, 500
фиксированный шаг сетки, 344
флажки, 499, 507
фокус-группы, 103
фолксономия, 378
фоновый тип приложения, 212
Фор, Дэвид, ценности проектирования взаимодействия, 191
форма
анализ, 179
и визуальный дизайн интерфейсов, 337
и функция, 353
формат документа, указание, 411
Формат, меню, 544
форм-фактор
определение, 164
сотрудничество с проектировщиками взаимодействия, 177
формы в Visual Basic, 490
фотография персонажа, 139
фотореалистичные пиктограммы, 352
фреймы, 482
функции
ассоциирование с элементами управления, 286
и требования, 152
пиктограммы, 350
функциональность
интеграция со стилем, 352
ограничение во встроенных системах, 227
функциональные
диалоговые окна, 579, 586
требования, 160
элементы, определение и группировка, 165
функция
и форма, 353
разница между настройкой и выполнением, 259

Х

характеристики персонажа, синтез, 136
Хеллер, Дэвид, 23
«хлебные крошки», 285
Ходж, Чаллис, 23
Холден, Критина («Universal Principles of Design»), 355
Хольцблат, Карен
«Contextual Design», 91, 119, 143, 155
«Rapid Contextual Design», 91
контекстное исследование, 90–92
о ролях пользователей, 119
объединенные модели, 119
ремесленническая модель обучения, 91
Хортон, Уильям («The Icon Book»), 350

хранение
данных по местоположению, 372
системы, 291, 372
электронной корреспонденции, 380
хромостереопсис, 359
Хэлли, Лейн (проектировщик)
процесс разработки персонажей, 133
сценарии, основанные на персонажах, 153

Ц

цвет
дополнительный, 359
и визуальный дизайн интерфейсов, 338, 340, 358
насыщенный, 359
цветовое восприятие, нарушения, 360
целенаправленное проектирование взаимодействия, 193
целеориентированное проектирование
выявление целей пользователей, 44
и успех продукта, 57
ликвидация разрыва между исследованиями и проектированием, 49
методы, 33
практика, 630
проектирование взаимодействия, 43
процесс, 48, 51, 53
стадия
выработки требований, 54, 151
детализации, 56, 179
исследований, 52
моделирования, 52, 110
определения инфраструктуры, 55
сопровождения разработки, 57
эволюция проектирования в промышленности, 41
целеориентированные вопросы для этнографических интервью, 100
цели, 132
бизнеса, 44, 131
выявление из качественных данных, 124
и персонажи, 118, 123
и телевизионные интерфейсы, 239
и шаблоны использования, 123
и этнографические интервью, 100
определение, 46
организации, 131
перетаскивания, 452
индикация, 453
потенциальная, 451, 4533
покупателей, 131
пользователей
выявление, 44
жизненные, 129, 137
и процесс разработки, 39
и создание цифровых продуктов, 34
и успех продукта, 57
как мотивы, 130
конечные, 129, 137
отсутствие понимания, 39
проектирование под цели в контексте, 47
сравнение с задачами и деятельностью, 45
типы, 127
эмоциональные, 128
синтез, 137
технические, 132
типы, 130
вставки, 453
целостность
внутренняя, 195
данных, 417
ценности проектирования, 190, 191, 193
минимизация вреда, 192
прагматичное взаимодействие, 194
целенаправленность, 193
элегантность, 194
этичные решения, 192
«цифровой бульон», 381
цифровые методы извлечения данных, 375
цифровые продукты
грубость, 36, 37
и конфликт интересов, 39
и процесс разработки, 33, 35, 36
компьютерный стиль мышления, 37
неподобающее поведение, 38
отсутствие процесса, 39
планирование и проектирование, 43
сосредоточенность на задачах, 46
успешные, 57
создание, 42

Ч

частота использования, 290
«Человеческий фактор: успешные проекты и команды» (Демарко, Листер), 243

Чиксентмихайи, Михай
«Flow: The Psychology of Optimal Experience», 243
поток, 243
чистый лист, принцип проектирования, 258
чтение текста, 357

Ш

шаблоны, 167, 625
библиотека, 197
взаимодействия, 167
запоминание, 309
каталог, 197
коллекция, 625
поведенческие, 198
позиционирования, 198
проектирования
архитектурные, 196
в программировании, 55
взаимодействия, 27, 55, 189, 196
запись и использование, 197
и архитектурные шаблоны, 196
типы, 198
структурные, 198, 199
пример Microsoft Outlook, 199
язык, 197
Шнейдерман, Бен, непосредственное манипулирование, 425
шрифт, 358, 361
без засечек (sans-serif), 358, 361
размер, 358
с засечками (serif), 358, 361
Шрифт, диалоговое окно (Word), 579
Шунар, Ивон, 195

Щ

щелчок мышью, 433
аккордный, 435

Э

эволюция
панелей инструментов, 560
проектирования в промышленности, 41
процесса разработки программного обеспечения, 36
эвристическая оценка, 90
Эйнштейн, Альберт, 356
экономия формы, 194
экран, разрешение, 360
экранное пространство, 205
экранные элементы управления, 360
экраны
навигация, 276
типовые, применение выбранного визуального стиля, 177
экспертная оценка, 90
эксперты
в предметной области (ЭПО), интервьюирование, 86
и налоги в интерфейсе, 268
и юзабилити-тестирование, 106
описание, 74
потребности, 79
экстранет, 222
элегантное проектирование взаимодействия, 247
элегантные решения проектирования, 194
электронная почта, хранение и доступ, 380
элементы
визуальные
в дизайне, 349
структурирование, 343
графические, выравнивание, 343
информационные
группировка, 168
определение, 165
управления
ассоциирование с функциями, 286
борьба с перегруженными диалоговыми окнами, 494, 566
вводом, 494, 514
круговой манипулятор, 519
неограничивающие, 515
ограничивающие, 515, 520
оперативные подсказки, 523
ползунки, 518
проверка допустимости, 521
рукоятки, 518
счетчики, 517
текстовые поля, 520
выбором, 494, 498
ввод данных в списках, 511
горизонтальная прокрутка, 510
деревья, 514
комбо-кнопки, 503
комбо-списки, 512
отметка, 507

элементы
управления
выбором
перетаскивание элементов, 509
радиокнопки, 502
раскрывающиеся списки, 506
списки, 505
упорядоченные списки, 510
флажки, 499, 507
категории, 494
командные, 494, 495
гиперссылки, 498
кнопки, 496
кнопки-значки, 496
комбо-списки, 526
манипулирование, 461
минимизация количества, 281
на панели инструментов, блокировка, 560
отображением, 494, 527
выдвижные панели, 530
полоса прокрутки, 528
разделители, 529
рычажки, 530
текстовые, 527
раскрывающееся поле со списком, 526
экранные, 360
эмоции, стимуляция, 195
эмоциональные цели, 128
эмпатия, 116
ЭПО (эксперты в предметной области), интервьюирование, 86
эстетика
и сетка, 346
как основа юзабилити, 355
этичные решения проектирования, 192
этнографические интервью, 91
вопросы, ориентированные
на мировоззрение, 100
на рабочий процесс, 100
выбор кандидатов, 93
демонстрации и рассказы, 101
и анализ рабочих заданий, 108
командный анализ, 102
методы проведения, 98
наводящие вопросы, 102
планирование, 96
подготовка, 93
проведение, 97
расписание, 97
системоориентированные вопросы, 100
целеориентированные вопросы, 100
этнографические методы полевых исследований, 52
эффективность
и ввод данных, 418, 424
и дизайн, 356
и персонажи, 113
и проектирование, 47
и сетка, 346

Ю

юзабилити, 28
юзабилити-тестирование, 105
вовлеченность проектировщика, 184
на стадии создания общей инфраструктуры пользовательского интерфейса, 164
полное, 182
проверка результатов проектирования, 181
промежуточное, 183

Я

язык
визуальный, 349, 359
стратегия, 56
шаблонов, 197
яркость и визуальный дизайн интерфейсов, 338

По договору между издательством «Символ-Плюс» и Интернет-магазином «Books.Ru – Книги России» единственный легальный способ получения данного файла с книгой ISBN 978-5-93286-132-5, название «Алан Купер об интерфейсе. Основы проектирования взаимодействия» – покупка в Интернет-магазине «Books.Ru – Книги России». Если Вы получили данный файл каким-либо другим образом, Вы нарушили международное законодательство и законодательство Российской Федерации об охране авторского права. Вам необходимо удалить данный файл, а также сообщить издательству «Символ-Плюс» (piracy@symbol.ru), где именно Вы получили данный файл.