РУКОВОДСТВО ДЖОЭЛА СПОЛЬСКИ

по подбору программистов и управлению ими

SMART AND GETS THINGS DONE

Joel Spolsky's Concise Guide to Finding the best Technical Talent

Apress®

РУКОВОДСТВО ДЖОЭЛА СПОЛЬСКИ

по подбору программистов и управлению ими

Джоэл Спольски

Издательский дом "Вильямс"
Москва • Санкт-Петербург • Киев
2008

ББК 65.245
С73
УДК 331.108.54

Издательский дом "Вильямс"
Главный редактор *С.Н. Тригуб*
Зав. редакцией *А.В. Назаренко*
Перевод с английского *И.В. Константинова*
Под редакцией *А.В. Назаренко*

По общим вопросам обращайтесь
в Издательский дом "Вильямс" по адресу:
info@williamspublishing.com, http://www.williamspublishing.com
115419, Москва, а/я 783; 03150, Киев, а/я 152

Спольски, Джоэл.

С73 Руководство Джоэла Спольски по подбору программистов и управлению ими. : Пер. с англ. — М. : ООО "И.Д. Вильямс", 2008. — 144 с. : ил. — Парал. тит. англ.

ISBN 978-5-8459-1354-8 (рус.)

Известный эксперт по вопросам организации работы компаний по разработке программного обеспечения Джоэл Спольски легко и понятно объясняет, как найти, отобрать, увлечь и нанять на работу лучших технических специалистов. Кроме того, много внимания уделяется созданию хороших условий труда для людей, которые этого действительно заслуживают. Книга будет полезна всем, кто нанимает на работу программистов или руководит ими в организации.

ББК 65.245

ISBN 978-5-8459-1354-8 (рус.)
ISBN 978-1-59059-838-2 (англ.)

Оглавление

Содержание

Посвящается Джареду

Об авторе

Джоэл Спольски — это всемирно признанный эксперт по разработке программного обеспечения. Его Web-сайт Joel on Software (`www.joelonsoftware.com`; на русском языке — `russian.joelonsoftware.com`) популярен среди разработчиков программ со всего мира и переведен более чем на 30 языков. Основав в Нью-Йорке компанию *Fog Creek Software*, он создал FogBugs — популярную систему управления проектами, предназначенную для команд программистов. Джоэл работал в компании *Microsoft*, где, входя в команду разработчиков Excel, он проектировал VBA, а работая в компании *Juno Online Services*, проектировал известный Интернет-клиент, используемый миллионами. Он написал две книги — *User Interface Design for Programmers* (Apress, 2001) и *Joel on Software* (Apress, 2004)[1], а также был редактором книги *The Best Software Writing I* (Apress, 2005)[2]. Степень бакалавра наук по программированию Джоэл получил в Йельском университете. Он служил парашютистом в Армии Израиля и был одним из основателей киббуца Ханатон.

[1] *Джоэл о программировании*, Символ-Плюс. — 2006.

[2] *Лучшие примеры разработки ПО*, Питер. — 2006.

Введение

Представьте себе маршрут с десятью препятствиями. В самом начале была целая группа тех, кому предстояло по нему бежать. Впереди их ждет двухэтажная стена, через которую надо перелезть, а также что-то вроде подвесного моста и равнины со змеями.

Предположим для простоты, что каждое препятствие останавливает 50% бегущих.

Итак, если бег начали 12 человек, то шесть из них после самого первого препятствия поймут, что проиграли. Можно будет увидеть, как они свалились в кучу у самого подножия двухэтажной стены.

Что касается оставшихся шести, то они будут двигаться по подвесному мосту, с которого трое из них уморительно вывалятся через веревочные перила. Через какое-то время эти вывалившиеся обнаружат, что висят на одной ноге, а из их карманов сыпется всякая ерунда: ключи, кошельки, монеты, а также резиновые утята, казу[1]. Ну, вы знаете, что еще может сыпаться, — ерунда, одним словом.

В действительности никто из них не преодолеет все десять препятствий.

На самом деле, чтобы это удалось хотя бы одному человеку, на старт необходимо вывести 2^{10}, т.е. 1024 бегуна.

Процесс найма прекрасного технического таланта — это отборочное соревнование по бегу. Многие слыхом не слыхивали о вашей компании. А многие пребывают в неведении, что вы сейчас заняты наймом. Другие вообще живут не в вашем городе. Есть и такие, у кого визы не в порядке. Другие же присылают свои резюме, но Джон, которые их все читает, обычно выбрасывает резюме от выпускников тех школ, чьи команды игроков в

[1] Крохотные музыкальные инструменты. — *Примеч. пер.*

американский футбол побеждали команду из округа Колумбия, в которой играл Джон. Тем не менее другие все же приходят на собеседование, но уходят ни с чем. Среди прочих бывают те, кто выше всяких похвал, но им уже предложили другую работу. Ну а те, кому другую работу не предложили, бывают так огорчены серой облупившейся краской на стенах и ужасным флуоресцентным освещением, что остаются на своей прежней работе. Небольшое количество людей, не возражающих против работы в каморках, решили, что на самом деле они не хотят зарабатывать на жизнь, программируя бомбы для уничтожения бункеров. Не то чтобы эти люди были против таких бомб, просто у них другое призвание. Препятствия, одним словом...

Такова вызывающая депрессию реальность найма, похожая на бег с препятствиями (и это действительно реальность). Следовательно, чтобы нанять троих программистов, надо заняться тремя тысячами кандидатов?

Во всей этой математической путанице есть рациональное зерно или, если хотите, светлая сторона. Она заключается в том, что если убрать *одно* препятствие — только одно! — то численность нанятых людей можно *удвоить*. Уберите два препятствия, и вы вчетверо увеличите количество нанятых, прошедших через остальные испытания. Ну и так далее, и тому подобное.

Однако это решение — не палочка-выручалочка. Нет такого волшебного предмета, с помощью которого вы решите все свои проблемы по найму и получите прекрасных специалистов, готовых работать на вас хоть завтра. Что вам нужно сделать, так это взглянуть на весь процесс как на маршрут с препятствиями и начать думать, как убрать их, чем побольше. Работайте над ними всеми, так как они одинаково важны.

Имея должность, называемую "технический рекрутер" или что-то вроде "менеджер по работе с персоналом", вы, возможно, начнете сталкиваться с небольшой проблемой. Действительно, если ваша должность не "Лорд-Суперверховный-Правитель, Командор-Империи и Матка-всех-пчел", то, вероятно, вы столкнетесь с тем, что иногда препятствия, связанные с наймом, не будут вашей виной, и повлиять на них вы не можете. Если люди не хотят работать на вашу компанию из-за того, что не желают сидеть в разделенной на каморки шумной,

темной комнате без окон, с мерцающим освещением, старыми протертыми коврами и явственным запахом плесени, то это проблема уже не найма, а офисного оборудования, не так ли? Да, я знаю, что вашей *вины* здесь нет, однако ваша проблема здесь *есть*. Я буду говорить о многом, что вам нужно делать для найма прекрасных разработчиков, но, тем не менее, это лежит за пределами возможностей нормального рекрутера. Да и со значительной частью этого "многого" не справится даже сам генеральный директор.

Впрочем, еще не все потеряно. Вам надо хотя бы иметь представление об этих проблемах, чтобы знать, чего добиваться. И если вам трудно нанимать специалистов из-за ужасного состояния офисной площади, то (хотя традиционно это не дело рекрутера) вам надо будет насильно пробиться на следующее заседание по планированию офиса. Если ваша компания расположена в месте, малопривлекательном для незаурядных выпускников колледжей, тогда об этом следует поговорить с вашим генеральным директором, как только он поинтересуется, почему еще не заполнены предназначенные для них вакансии. Обычно же вам придется работать в тех частях маршрута с препятствиями, которые вы *можете* контролировать, и в таких случаях вам гарантированы положительные результаты.

История, не относящаяся к этой книге

Давным-давно (а если точно, то в 2000 году) я написал для своего Web-сайта путанную статью, озаглавленную "Ты что, не можешь найти программистов?"[2].

В те времена первый бум "дот-комов" был в самом разгаре и спрос на Web-программистов был таким высоким, что многие большие компании вынуждены были обращаться к фирмам, за-

[2] Обычно я указываю URL-адрес статьи, по которому ее можно просмотреть. Но эта старая статья, написанная мною еще в юности, просто нелепая, поэтому, если она еще где-то есть во Всемирной паутине, ищите ее сами, но этого я делать не советую. В оригинале она называется *Whaddaya mean, you can't find programmers*?

нимающимся IT-консалтингом, и платить по 200–300 долл. в час за не слишком хороших HTML-кодеров. Возникали целые армии Интернет-консультантов, в которые брали 22-летних выпускников колледжей, — лишь бы эти выпускники знали, как пользоваться FrontPage. Их потенциальный почасовой заработок составлял примерно 40 долл., но по выставленным за их работу счетам надо было платить уже 250 долл. в час.

Смысл статьи был в том, что достаточно хорошо платить, предоставить для работы хорошо освещенные офисы, организовать бесплатный массаж, и тогда проблем с наймом не будет.

Однако вскоре после того, как статья была написана, первый "Интернет-пузырь" лопнул, и в технической индустрии наступило что-то вроде ядерной зимы. На улицу была бесцеремонно выброшена масса программистов, разработчиков и Web-дизайнеров. Многие из них даже не понимали, что на самом деле несправедливо, когда начальную зарплату в 60 тыс. долл. получали те выпускники колледжа, чьей основной специальностью был английский язык, а единственным техническим навыком — умение создавать в Geocites личную страницу. Программисты были рады хоть какой-то работе, даже (*зажмите нос*) на эту отвратительную *Microsoft* (*прямо мороз по коже*). Следующие три–пять лет я вообще стеснялся писать подобные бестактности, когда столько талантливых (и бесталанных тоже) программистов вернулись жить к своим родителям и устроились работать в магазинах канцтоваров.

Когда я недавно перечитывал эту статью, то понял, что написал ее о том, в чем совсем не разбирался. Имея несколько идей о том, как нанимать программистов, я совсем не обладал опытом работы в этой области.

Последние шесть лет мы с моим другом Майклом создаем компанию по разработке ПО. С самого начала, когда мы даже не знали, какими именно будут эти программы, нас не покидает *полная* уверенность, что приоритет *номер один* — это найм прекрасных людей. Все эти годы, даже до того как появилась хоть какая-то возможность кого-либо нанять, мы постоянно следили за тем, чтобы у нас хотели работать прекрасные люди. Мы допустили некоторые ошибки, а также многому научились, и теперь я чувствую, что привлечение прекрасных людей

для работы в *Fog Creek* — это практически единственная проблема, с которой мы *не сталкиваемся*. Теперь весь процесс привлечения замечательного таланта мне знаком намного больше, чем во время написания своей первой статьи, при виде которой меня передергивает.

Да и за последний год-другой кое-что изменилось — закончилось тяжелое время, снова пошли инвестиции в технологию, здесь и там появляются новые фирмочки, а многие относительно признанные технологические компании опять бросились нанимать сотрудников, как сумасшедшие. Почти в каждой компании, нанимающей разработчиков программ, имеется длинный список вакансий. А компании, где этого списка нет, являются по большей части проворными, созданными с нуля, фирмами, которые стараются прожить за счет денег, реально заработанных на продаже своей продукции заказчикам; это намного более здоровая ситуация, чем в 2000 году. В целом я замечаю, что наличие вакансий характерно для компаний с большим количеством денег и заказчиков, а эти компании заняты наймом просто потому, что снова хотят инвестировать в последующие прибыли и в дальнейший рост. А чего я *не вижу*, так это доминирования на рынке труда множества фирм, созданных сумасшедшими, которые, вполне естественно полагая, что в новой компании обязательно должно быть примерно 170 сотрудников, первое, что делают, — нанимают 170 человек.

Как и прежде, менеджеры, посредники по найму и рекрутинговые директора, которые долгое время были полностью убеждены, что самое важное — это нанять прекрасных разработчиков программ, начинают удивляться, почему так тяжело заполнить все эти имеющиеся у них вакансии. Видя такую ситуацию, я и написал эту книгу, надеясь, что она станет в некотором роде искуплением за мою первую неловкую попытку.

Еще я надеюсь, что книга поможет сделать вашу организацию прекрасным местом работы и, помогая вам, хоть немного увеличит количество счастья в мире.


50
LBS

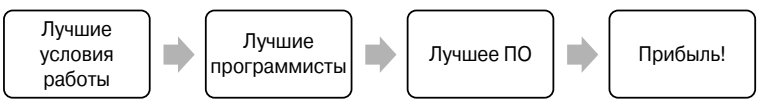
Глава 1

Достигая тех высот

В марте 2000 года я запустил Web-сайт Joel on Software[1], сделав довольно громкое заявление: большинство людей не правы, думая, что для создания успешной компании по написанию программ нужна идея. Вот мои слова.

Довольно распространенно убеждение, что вы создаете компанию по написанию программ, ставя цель найти разумную идею, которая справляется с некоторой еще нерешенной проблемой, затем реализовать эту идею и сколотить состояние. Мы называем это "верой в создание еще лучшей мышеловки". Но настоящей целью компании по написанию программ должен быть перевод капитала в работающие программы[2].

Последние пять лет я испытывал эту теорию на практике. Вот формула, по которой работает компания *Fog Creek Software*, основанная мною вместе с Майклом Прайором в сентябре 2000 года. Вкратце эту формулу можно показать в виде четырех этапов.



Это очень удобная формула, особенно потому, что настоящей целью основания *Fog Creek* была компании по созданию программ, где *нам хотелось бы работать*. Еще тогда я объявил, что хорошие условия труда (или более коряво, "создание компании, где хотелось бы работать лучшим в мире разработчикам программ") должны привести к прибылям, причем так

---

[1] `www.joelonsoftware.com`.

[2] Joel Spolsky, "Converting Capital Into Software That Works", опубликованная на сайте `www.joelonsoftware.com`, 21 марта 2000 года (поиск по словам "Converting Capital").

же естественно, как шоколад — к полноте и мультяшный секс в видеоиграх — к разгулу бандитизма.

Впрочем, сейчас мне хотелось бы ответить всего на один вопрос, ведь если эта часть теории неверная, тогда и вся теория разваливается. Этот вопрос состоит в следующем: есть ли какой-то смысл говорить, что надо заполучить "самых лучших программистов"? Не слишком ли много разновидностей программистов, чтобы предыдущий вопрос имел какой-то смысл?

Может, для нас это утверждение и очевидно, но для многих оно все равно нуждается в доказательстве.

Несколько лет назад сравнительно крупная компания собиралась купить *Fog Creek*, но я знал, что это невозможно, так как генеральный директор этой компании говорил, что в действительности он не согласен с моей теорией о найме самых лучших программистов. Аргументируя свою позицию, он использовал библейскую метафору: вам нужны только царь Давид вместе с армией солдат, которые просто должны выполнять его приказы. Впоследствии акции его компании заметно упали в цене — с 20 долл. до 5 долл. за штуку; поэтому хорошо, что мы не приняли предложение.

На самом же деле, если взять мир бизнес-журналистов (мастеров списывать друг у друга) и больших компаний (тех, в которых полагаются на менеджеров-консультантов со сверхоплатой, нанятых думать за компанию, жевать за нее и т.д.), то для здравого смысла этого мира, как мне кажется, самым важным является уменьшение *стоимости* программистов.

В некоторых других отраслях дешевое *важнее* хорошего. *Wal-Mart* выросла в самую большую корпорацию на земле, продавая не хорошие, а дешевые продукты. Если бы эта компания попыталась продавать высококачественные товары, то их цена была бы более высокой, и преимущество, достигаемое благодаря дешевизне, было бы потеряно. Например, если бы эта компания пыталась продавать носки, которые могут выдерживать, например, очень интенсивную стирку в стиральной машине, то в них пришлось бы использовать всевозможные дорогие компоненты, например хлопок, и тогда стоимость каждого носка должна бы была возрасти.

Итак, почему в отрасли по созданию программного обеспечения нет места поставщикам дешевизны — тем, кто старается использовать самых дешевых программистов? (Напомните, чтобы я не забыл спросить у *Quark*, как работает план "Выжми из каждого все, затем найди ему дешевую замену".)

Ответ кроется в том, что дублирование программ обходится бесплатно. Это значит, что цена программистов распределяется на все копии продаваемой вами программы. А если говорить о качестве программ, то его можно поднять, не наращивая цену продаваемой единицы продукции.

В сущности, разработка скорее увеличивает ценность, а не цену.

Или, попросту говоря, проявив скупость к программистам, вы создадите дорогие программы, которые даже не окупят своего производства.

То же самое применимо и к индустрии развлечений. Вполне оправданно нанять (даже за большие деньги) для вашего последнего блокбастера Брэда Питта, поскольку затраченные средства можно будет разделить между всеми теми миллионами зрителей, которые придут смотреть фильм только потому, что Брэд такой *чертовски крутой*.

Или, с другой стороны, для вашего последнего блокбастера вполне оправданно нанять (даже за большие деньги) Анджелину Джоли, так как и в этом случае затраченные средства можно будет разделить между всеми теми миллионами зрителей, которые придут смотреть фильм только потому, что Джоли такая *чертовски крутая*.

Но я пока еще ничего не доказал. Что значит быть "самым лучшим программистом" и действительно ли программы, созданные разными программистами, так сильно отличаются по качеству?

Давайте начнем со старой доброй производительности. Измерить производительность программиста довольно трудно. Почти любая мера, какую можно применить (количество строк отлаженного кода, аргументов командной строки, количество точек входа функций), является слишком примитивной, да и очень трудно получить конкретные данные по большим проектам, ведь крайне редко двум программистам поручают сделать одно и то же.

Что касается меня, то я пользуюсь данными профессора Стэнли Айзенштата из Йельского университета. Каждый год он ведет интенсивный курс программирования (CS 323), по большей части состоящий из пяти заданий по программированию или что-то вроде этого. На выполнение каждого из них дается примерно две недели. Для вузовского курса обучения это очень серьезные задания: реализовать для системы Unix оболочку командной строки, реализовать программу ZLW-сжатия файлов и т.д.

Студенты были так недовольны слишком большой работой, требуемой на освоение курса, что профессор Айзенштат стал просить их сообщать, сколько времени они потратили на каждое задание. Эти данные он тщательно собирал в течение нескольких лет.

Я потратил некоторое время, обрабатывая его цифры; пожалуй, это единственный известный мне набор данных, который позволяет сравнить десятки студентов, одновременно работавших над одинаковыми заданиями с помощью одной и той же технологии.

Первое, что я сделал с этими данными, — подсчитал для часов, потраченных на каждое из заданий, среднее, минимум, максимум и стандартное отклонение. Результаты моих расчетов приведены ниже.

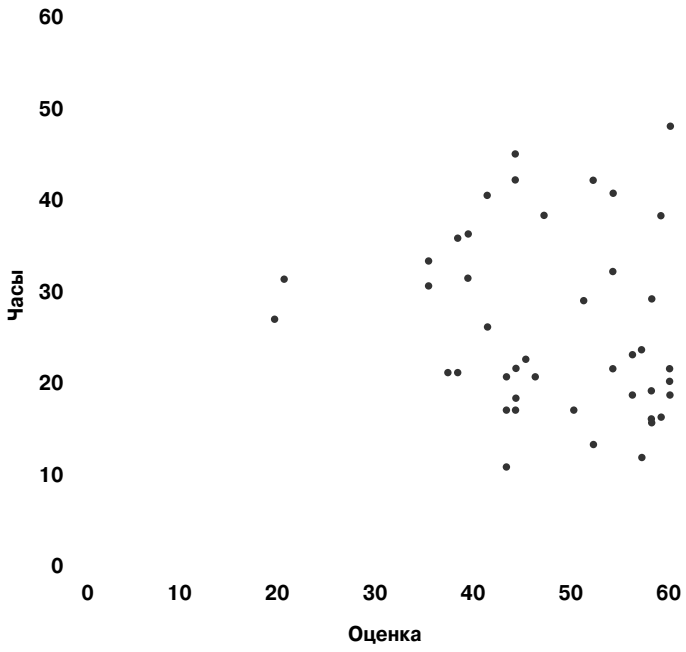
Сильнее всего здесь бросается в глаза громадный разброс значений. Самые быстрые студенты справлялись с работой в три-четыре раза быстрее, чем средние, и в десять раз быстрее, чем самые медленные студенты. Стандартное отклонение оказалось крайне большим. Поэтому я подумал, что, скорее всего, для некоторых студентов задание оказалось ужасно сложным. Мне не хотелось учитывать тех студентов, которые, потратив на задание четыре часа, не смогли создать работающей программы. Поэтому я рассматривал данные только по студентам, находящимся в верхнем квартиле успеваемости, ...т.е. среди лучших 25% по качеству кода. Должен сказать, что оценки на курсе профессора Айзенштата *полностью* объективны: они вычисляются по формулам, где используется только количество пройденных автоматических тестов кода. За плохой стиль никакие баллы не вычитались.

Проект	Среднее	Минимум	Максимум	Стандартное отклонение
CMDLINE99	14,84	4,67	29,25	5,82
COMPRESS00	33,83	11,58	77,00	14,51
COMPRESS01	25,78	10,00	48,00	9,96
COMPRESS99	27,47	6,67	69,50	13,62
LEXHIST01	17,39	5,50	39,25	7,39
MAKE01	22,03	8,25	51,50	8,91
MAKE99	22,12	6,77	52,75	10,72
SHELL00	22,98	10,00	38,68	7,17
SHELL01	17,95	6,00	45,00	7,66
SHELL99	20,38	4,50	41,77	7,03
TAR00	12,39	4,00	69,00	10,57
TEX00	21,22	6,00	75,00	12,11
ВСЕ ПРОЕКТЫ	**21,44**	**4,00**	**77,00**	**11,16**

Вот результаты для первого квартиля.

Проект	Среднее	Минимум	Максимум	Стандартное отклонение
CMDLINE99	13,89	8,68	29,25	6,55
COMPRESS00	37,40	23,25	77,00	16,14
COMPRESS01	23,76	15,00	48,00	11,14
COMPRESS99	20,95	6,67	39,17	9,70
LEXHIST01	14,32	7,75	22,00	4,39
MAKE01	22,02	14,50	36,00	6,87
MAKE99	22,54	8,00	50,75	14,80
SHELL00	23,13	18,00	30,50	4,27
SHELL01	16,20	6,00	34,00	8,67
SHELL99	20,98	13,15	32,00	5,77
TAR00	11,96	6,35	18,00	4,09
TEX00	16,58	6,92	30,50	7,32
ВСЕ ПРОЕКТЫ	**20,49**	**6,00**	**77,00**	**10,93**

Разница не слишком большая! Для верхнего квартиля стандартное отклонение почти одинаковое. И действительно, если внимательно просмотреть данные, то станет достаточно ясно: *заметной корреляции между временем и оценкой нет*. Вот типичный разброс данных по одному из заданий... Выбираю задание COMPRESS01, посвященное реализации алгоритма сжатия Зива–Лемпеля–Уэлча и предложенное студентам в 2001 году. Дело в том, что стандартное отклонение по этому заданию достаточно близко к общему стандартному отклонению.



Здесь вы ничего не видите, и в этом все дело. *Качество работы и количество потраченного времени просто не коррелируют между собой.*

Я спросил об этом профессора Айзенштата, и он обратил мое внимание на еще один фактор: время, выделенное для задания, истекает в определенный момент (обычно в полночь), **а штрафные** санкции за опоздание довольно значительны, по-

этому многие студенты не успевают закончить проект. Другими словами, максимальное время, потраченное на эти задания, такое низкое частично потому, что на выполнение задания (от его получения до сдачи результата) выделяется слишком мало часов. Если бы студенты, работая над проектами, были не ограничены во времени (что немного ближе к реальности), разброс был бы только *больше*.

Эти данные не совсем научные. Скорее всего, в них заложен определенный обман. Кое-кто из студентов может завысить время, потраченное на задание, надеясь добиться от преподавателя определенной симпатии и получить в следующий раз задание полегче. (Желаю удачи! В CS 323 задания сейчас такие же, как и во время моего обучения, т.е. в 80-х годах XX века.) Другие студенты, наоборот, занижают время, потому что сбились с темпа. Тем не менее, сделанный на основе этих данных вывод, что производительность разных программистов может отличаться как 5 : 1 или 10 : 1, я не считаю таким уж преувеличением.

Это еще не все!

Если бы программисты отличались друг от друга одной только производительностью, то следовал бы вывод, что пять заурядных программистов могут заменить одного по-настоящему хорошего. Но такой вывод явно неверен. Почему это так, следует из закона Брукса: “Подключение к запаздывающему программному проекту дополнительных работников делает его еще более запаздывающим”[3]. Когда над одной задачей работает один хороший программист, ему не нужно дополнительно заниматься координацией работы или поддержкой связи с коллегами. А если над задачей работает уже пять программистов, они вынуждены брать на себя эту дополнительную работу, что отнимает много времени. Таким образом,

[3] Frederick Brooks, *The mythical Man-Month: Essays on Software Engineering* (Reading, Mass.: Addison-Wesley, 1975); Фредерик Брукс, *Мифический человеко-месяц, или Как создаются программные комплексы*. — Пер. с англ. — СПб.: Символ-Плюс, 1999.

если уменьшать команду, насколько это возможно, то появится дополнительное преимущество; “человеко-месяц” действительно выглядит мифом.

И ЭТО ЕЩЕ НЕ ВСЕ!

На самом же деле, когда вместо двух хороших программистов используют множество посредственных, то сколько бы они ни работали, их результаты не будут такими же хорошими, как у прекрасных программистов.

Пять Антонио Сальери, даже работая сто лет, все равно не создали бы *Реквием* Моцарта.

А пять Джимов Дэвисов... Это создатель несмешного мультипликационного кота Манди; из относящихся к коту шуток 20% — о том, как Манди лакает молоко, а остальные — как он любит лазанью (и в этом весь юмор!). Так вот, пять Джимов Дэвисов, даже потратив остаток жизни на написание комедий, все равно не создали бы такой эпизод, как “Суповый нацист” из сериала “Сейнфелд”.

Команда из *Creative Zen* может тратить годы, чтобы исправлять огрехи в своем уродливом iPod, и все равно не создаст такого прекрасного, доставляющего удовольствие и элегантного продукта, как iPod, выпущенный компанией *Apple*. Да и не собирается эта команда откалывать кусок от доли *Apple* на рынке продаж, так как *не имеет* волшебного таланта разработчика. Просто *нет его там*, и все.

Посредственность *никогда не возьмет высоких нот*, зато самый большой талант только это и делает. Ничтожно мало оперных див могут спеть партию Королевы ночи из моцартовской “Волшебной флейты”, которую просто нельзя исполнять без этой партии.

Итак, программное обеспечение — это действительно что-то вроде высоких музыкальных нот? “Может, для какой-то части программ это верно, — скажете вы. — Но я работаю над пользовательскими интерфейсами по приему счетов для утилизации медицинских отходов”. Что же, довольно справедливо. Основное внимание я уделяю компаниям, успех и неудача которых зависят от качества продукции. А если вы используе-

те свои программы только “для внутреннего пользования”, то вам, вероятно, только и нужно, чтобы они были достаточно хорошими.

Последние несколько лет нам встречалось много примеров замечательного программного обеспечения, по-настоящему высоких нот — того, что посредственные разработчики программ просто *не могли бы* создать.

К 2003 году компания *Nullsoft* выпустила новую версию Winamp, сопроводив выпуск вот такими комментариями на Web-сайте.

- Новый привлекательный вид!
- Новые превосходные функции!
- Большинство возможностей действительно работает!

Эта последняя фраза — “Большинство возможностей действительно работает!” — заставляет всех смеяться. Из того что люди действительно написали такое на своем сайте, напрашивается единственный вывод, что они... довольны, восхищены этой программой, используют ее, да еще рекомендуют своим друзьям, а Winamp внушает им благоговейный страх. Не правда ли, круто?

А если бросить горстку дополнительных программистов в команду Windows Media Player, смогут ли они достичь тех высот? Никогда в жизни. Причина здесь в том, что, чем больше людей будет в этой команде, тем больше вероятность, что среди них попадется настоящий вечно недовольный ворчун, у которого хватит непрофессионализма и незрелости написать на сайте: “Большинство возможностей действительно работает!”

Оставим без комментария слова: “Winamp 3: почти такая же новая, как и Winamp 2!”

Такие вещи и заставили нас полюбить Winamp.

К тому времени, когда *AOL Time Warner Corporate Weenieheads* положила глаз на эту программу, весь юмор с Web-сайта исчез. Вы можете видеть подобных людей просто кипящими от злости, мучающимися и лицемерно сочувствующими, как Сальери в фильме “Амадеус”, пытающимися сбить все признаки творчества, которые могли бы напугать какую-нибудь

старуху в Миннесоте, в результате этих попыток стирающими все, что могло бы заставить людей *любить* продукт.

Теперь посмотрите на iPod. *Вы не можете поменять батарейку*. Иными словами, *очень плохо*, если она "садится". *Покупайте новый iPod.* На самом же деле, компания *Apple* выполнит замену, если вы отправите iPod на фабрику, но это будет стоить 65,95 долл. Безобразие.

Так почему же нельзя поменять батарейку?

Моя теория состоит в следующем. *Apple* не хочет, чтобы совершенно гладкая, безупречная поверхность ее прекрасного, сексуального iPod была повреждена одной из тех ужасных батарейных крышек, которые можно видеть на дешевой потребительской ерунде и которые имеют маленькие, вечно ломающиеся задвижки, а также щели, забивающиеся нитками из карманов и прочей общеизвестной гадостью. iPod — это самый безупречный экземпляр изо всей той потребительской электроники, которая мне известна. Он прекрасен. Он прекрасен *на ощупь*, как гладкий морской камень. И весь эффект морского камня может быть сведен на нет одной-единственной батарейной крышкой.

Компания *Apple* приняла решение в пользу *стиля*; и действительно, в iPod полно таких *стильных* решений. Впрочем, стиль — это не то, чего способны достичь 100 программистов компании *Microsoft* или 200 промышленных дизайнеров компании, неуместно названной *Creative*[4], так как среди них нет Джонатана Айва, да и не слишком часто встречаются такие люди, как он.

Извините, но не могу закончить говорить об iPod. А прекрасное колесико, которое вращают большим пальцем и которое издает тихие щелкающие звуки... *Apple* потратила *дополнительные деньги*, чтобы вставить *непосредственно в iPod* динамик, дающий возможность слышать эти звуки. Если бы щелкающие звуки передавались через наушники, то были бы сэкономлены копейки (всего лишь *копейки!*). Зато благодаря щелкающему колесику вам кажется, что вы чем-то управляете. А людям нравится такое ощущение. *Люди счастливы, когда им*

[4] Творческая. — *Примеч. пер.*

кажется, что они чем-то управляют. То, что колесико реагирует на команды плавно, быстро и *слышно*, делает вас *счастливым*. Этого не скажешь об остальных 6000 изделий компьютерной электроники карманного размера, для загрузки которых нужно так много времени, что, нажав выключатель, приходится ждать целую минуту, пока не станет заметно, что что-то произошло. Управляете ли вы процессом? Кто знает? Когда последний раз вы имели мобильный, который был мгновенно готов уже в момент нажатия вами кнопки?

Стиль.

Удовольствие.

Чувства.

Вот то, что создает гигантские хиты среди программных продуктов, фильмов и потребительской электроники. Если вы это не поймете правильно, то хотя, возможно, и решите какую-то проблему, но ваш продукт не станет хитом номер один, который сделает богатым каждого сотрудника компании, чтобы вы *все* гоняли на стильных, веселых и привлекательных машинах, таких, например, как Ferrari Spider F1, и у вас бы еще осталось достаточно денег, чтобы построить у себя на заднем дворе ашрам.

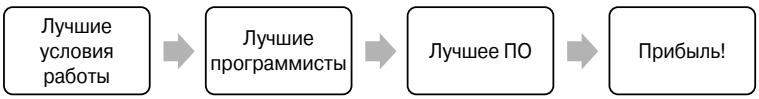
Дело не в том, чтобы быть "в 10 раз производительнее", а в том, что разработчик "средней производительности" никогда не возьмет высоких нот, создающих прекрасные программы.

Печально, что это на самом деле не относится к разработке программ, не являющихся продуктом. Домашние, предназначенные для внутреннего пользования программы редко настолько важны, чтобы из-за них приглашать рок-звезд. Ведь никто не приглашает Долли Партон петь на свадьбах. Вот почему разработчики программ в основном делают успешную карьеру в нынешних программных компаниях, а "не клепая софт" для какого-нибудь банка.

На современном рынке программного обеспечения действует принцип "победитель получит все". Больше никто не зарабатывает деньги на MP3-плейерах, кроме *Apple*. Больше никто не зарабатывает деньги на электронных таблицах и текстовых процессорах, кроме *Microsoft*. Конечно, я знаю — они очень жестко боролись с конкурентами, чтобы занять это ме-

сто. Тем не менее это не опровергает того, что победитель получит все.

Вы не можете себе позволить быть вторым номером или выпускать “достаточно хороший” продукт. Он должен быть необыкновенно хороший, то есть хороший настолько, чтобы люди это заметили. А “чаевые”, которые вы получите от действительно талантливых разработчиков программ, и служат вашей единственной надеждой на “замечательность”. Обо всем этом говорится в схеме:




FLOSS
LINUX
RULES
COMPU
TECH
HR

Глава 2

Поиск прекрасных разработчиков

Где все эти прекрасные разработчики?

Если вы похожи на большинство людей, то первый раз стараясь заполнить какую-либо вакансию, вы дадите объявления, будете, возможно, просматривать большие Интернет-ресурсы и получите "тонну" резюме.

Изучая их, вы будете думать: "Хм-м, возможно, подходящий вариант", или "Ни в коем случае!" либо "Интересно, он согласится переехать в Буффало?" А вот то, чего *точно не будет*. Вы, даю гарантию, *никогда* не скажете: "Какой замечательный специалист! Нам он очень нужен!" И действительно, изучив тысячи резюме (если, конечно, знаете, как их читать, а это нелегко, о чем мы и поговорим в главе 4), просмотрев тысячи заявок, вы, скажу честно, никогда не найдете прекрасного разработчика программ. Ни одного.

И вот почему.

Прекрасные разработчики программ будут в действительности лучшими людьми в своей отрасли, и их просто-напросто *никогда нет на рынке труда.*

В среднем великий разработчик программ за всю свою трудовую деятельность поменяет, *возможно*, четыре работы.

Прекрасных выпускников колледжей привлекает в интернатуру какой-либо профессор, имеющий связи в какой-либо компании, а затем им очень быстро предлагают в ней место, и о поступлении на другую работу они больше не беспокоятся. А если они все же уходят из компании, то это часто потому, чтобы работать вместе с другом в новой фирме или последо-

вать за своим прекрасным боссом в другую компанию либо потому, что они решили работать, скажем, на Eclipse, потому что она крутая, поэтому и ищут работу, связанную с Eclipse, в компании *BEA* или *IBM*, и, конечно, они ее находят, потому что они прекрасные специалисты.

Если вы *удачливы*, т.е. *очень удачливы*, тогда однажды прекрасные специалисты явятся на открытый рынок труда — например, вторая половина кого-то из них решит проходить медицинскую интернатуру в Анкоридже, и эта пара действительно отправит свои резюме в те несколько мест, в которых им хотелось бы работать, живя в Анкоридже.

Однако в большинстве случаев прекрасные разработчики (и это почти тавтология) являются, конечно же, прекрасными (ладно, это точно тавтология), так что опытные кадровики, как правило, быстро увидят их выдающиеся качества. Это значит, что они обычно поступают на любую работу, на которую хотели попасть. Так что, поверьте мне, им не нужно рассылать большое количество резюме.

Вам кажется, что именно такого человека вы хотите нанять? Скорее всего, так и есть.

Следствие этого правила (что прекрасных людей никогда нет на рынке труда) состоит в следующем: плохих людей, т.е. опасно неквалифицированных, на этом рынке *очень много*. Они всегда теряют работу, так как не могут ее делать. Их компании терпят крах — иногда потому, что туда, скорее всего, принимают еще много других неквалифицированных программистов, и это все вносит свой вклад в крах компании, а иногда плохие специалисты *на самом деле такие неквалифицированные, что сами компанию и разрушают*. Да, и такое бывает.

К счастью, эти ужасно неквалифицированные люди редко получают работу, но они настойчиво пишут заявления о приеме, а написав, заходят на сайт поиска работы и отмечают галочкой от 300 до 1000 вакансий, надеясь, что им повезет в лотерею.

Количественно прекрасных специалистов довольно мало, и их никогда нет на рынке труда, в то время как некомпетентные люди, даже если их *точно так же мало*, за время своей карьеры

тысячи раз обращаются с предложением своих услуг. А сейчас, Спарки, посмотри на ту большую стопку резюме, полученных тобой из *Craigslist*. Надо ли удивляться, что большинство из них отправили люди, которых тебе не хочется нанимать?

Надеюсь, проницательные читатели заметят, что я пропустил самую большую группу — солидных, компетентных людей. Их на рынке больше, чем прекрасных специалистов, но меньше, чем некомпетентных, и если брать в общем, то в пачке из 1000 резюме они представлены *мало*. Впрочем, сейчас по большому счету почти каждый менеджер по кадрам из Пало-Альто из 1000 резюме на своем столе извлекает 970, полученных от одного и того же меньшинства в 970 некомпетентных людей, которые обращаются по поводу почти каждой вакансии в Пало-Альто, и будут, возможно, это делать всю свою жизнь. И только на 30 резюме хотя бы стоит обратить внимание, причем прекрасный программист будет стоять лишь за одним из этих резюме (и то, если повезет). Ладно, хорошим может быть не только одно резюме. Вы еще увидите — найти эти иголки в стоге сена невероятно сложно, хотя и реально.

Могу я их получить в конце-то концов?

— Да!

— Ну, может быть!

— Возможно, это зависит от определенных причин!

Вместо того чтобы считать набор персонала "процедурой сбора и фильтровки резюме", вам надо начинать думать об этом наборе как о "процедуре отслеживания победителей и принуждения их к тому, чтобы они поговорили с вами".

У меня есть три главных метода добиться этого.

1. Забраться на вершину.

2. Интернатура.

3. Организация своего сообщества*.

После пункта "Организация своего сообщества" стоит маленькая звездочка, что означает "тяжело" — похоже на знаме-

нитую математическую проблему, решенную Джорджем Данцигом, который зашел в аудиторию слишком поздно и не услышал, что проблема считается неразрешимой[1].

Кроме того, у вас могут быть свои собственные идеи. Я просто собираюсь рассказать о трех идеях, которые помогают именно мне.

Забираемся на вершину!

Подумайте о том, где могут "тусоваться" люди, которых вы хотите нанять. Какие конференции они посещают? Где они живут? К каким организациям принадлежат? Какие Web-сайты читают? Вместо того чтобы бросать широкие сети с помощью поиска на универсальном сайте, используйте "кадровое агентство" *Joel on Software* (`jobs.joelonsoftware.com`) и ограничьте свой поиск умными разработчиками, которые читают мой Web-сайт. Ходите на действительно интересные технические конференции. Прекрасные Mac-разработчики бывают на конференции WWDC, организуемой компанией *Apple*. А прекрасные программисты, работающие в Windows, бывают на конференции PDC, организуемой компанией *Microsoft*. Кроме того, есть еще целая куча конференций, которые посещают любители открытого исходного кода.

Ищите крутую новую технологию. В прошлом году это был Python, а в нынешнем — Ruby. Ходите на конференции, где вы найдете тех, кто первым адаптирует такие технологии; эти люди удивляются всему новому и всегда заинтересованы в самосовершенствовании.

Крадитесь по коридорам, разговаривайте со всеми, кого встретите, ходите на технические сеансы и приглашайте докладчиков на пиво, а когда найдете кого-то интересного, то БА-БАХ! — бросайтесь в настоящие флирт и лесть.

— О-ох, как это *интересно*! — скажете вы. — Вау, не могу поверить, какой вы *умный*. Да к тому же красивый. Так где, вы

---

[1] Donald J. Albers, Constasnce Reid, "An interview of George B. Dantzig: The Father of Linear Programming", *College Mathematics Journal*, April, 1986, p. 293–314.

сказали, работаете? В самом деле? *Там?* Хм-м-м. Вам не кажется, что может быть и лучше? Думаю, моя компания могла бы вас нанять...

Из этого правила следует, что надо *избегать* рекламы в больших универсальных сайтах по поиску работы. Как-то летом я, не подумав, разрекламировал нашу летнюю интернатуру с помощью сайта MonsterTRAK, дающего за небольшую доплату возможность показать эту рекламу студентам всех учебных заведений США. Это привело к появлению буквально сотен резюме, ни одно из которых не было принято даже в первом туре. В конце концов мы, потратив кучу денег, получили кучу резюме, которые не оставили нам почти никакого шанса найти тех, кто нам нужен. Через несколько дней уже сам факт того, что источником резюме является MonsterTRAK, заставил меня думать, что, приславший его кандидат был, возможно, не для нас. Аналогично, когда появился такой ресурс, как *Craigslist*, и его действительно посещали первопроходцы Интернета, мы находили прекрасных специалистов по размещенной в нем рекламе. Однако сейчас его использует буквально каждый, кто обладает умеренной компьютерной грамотностью, благодаря чему появляется слишком много резюме, дающих очень малый шанс “найти иголку в стоге сена”.

Интернатура

Один прекрасный способ перехватить прекрасных специалистов, которых никогда нет на рынке труда, — это заполучить их, пока они еще не знают, где *находится* рынок труда, т.е. тогда, когда они еще учатся.

Некоторые менеджеры по приему ненавидят уже саму идею нанимать интернов. Для них они недисциплинированы и недостаточно квалифицированы. До некоторой степени это правда. Интерны не такие опытные, как опытные работники (разве нет?!). В них надо вкладывать чуть больше, и перед тем, как они преуспеют, должно пройти некоторое время. Для нашей деятельности хорошо то, что по-настоящему прекрасные программисты часто начинали программировать в десять лет. И в то время как любой другой человек их возраста бегал, иг-

рая в футбол, они сидели в папином домашнем офисе, пытаясь скомпилировать ядро Linux. (Футбол — это игра, популярная у многих детей, не умеющих программировать; он состоит в пинании ногами (знаю, что это ужасно) сферического объекта, который называется "мяч"). Вместо того чтобы гоняться за девочками по площадке для игр, они устраивали на Usenet ожесточенные войны по поводу крайней порочности программных языков, не реализующих интерфейс в стиле Гаскелла. Вместо того чтобы организовывать в гараже музыкальную группу, они делали крутой трюк, и когда сосед через их Wi-Fi-точку открытого доступа крал у них полосу пропускания, то в его Интернет-браузере все изображения отображались перевернутыми. БУ-ГА-ГА!

Итак, область разработки программ все же отличается, скажем, от юриспруденции или медицины, и к тому времени, когда эти дети будут учиться на втором или третьем курсе, они станут довольно хорошими программистами.

Почти каждый подает заявку только на *одну* работу — на самую первую, и большинство студентов думают, что до выпуска думать об этом не нужно. Действительно, тут большинство из них изобретательностью не отличаются и будут подавать это заявление только в пределах студенческого городка. В студенческих городках хороших колледжей приличных вакансий хватает, и студенты редко обращаются к работодателям, которые не появляются в этих городках.

Итак, либо вы будете участвовать в этом безумии, т.е. заниматься приемом прямо в студгородке, и — поймите меня правильно — это хорошо, либо совсем наоборот — будете стараться заполучить прекрасных детей за один-два года *до выпуска*.

Действуя таким способом, я добился в компании *Fog Creek* немалых успехов. Процесс начинается каждый год в сентябре, когда я принимаюсь использовать все свои ресурсы, чтобы найти лучших в стране студентов-компьютерщиков. Отправляю письма в пару сотен факультетов и кафедр компьютерных наук. Изучаю списки тех, кто специализируется в этих науках и получит диплом через два года (обычно, чтобы найти эти списки, надо на факультете или на кафедре иметь знакомого — профессора или студента). Затем я пишу письмо персонально каждому студенту из

списка. Не сообщение электронной почты, а настоящее послание на бумажном бланке фирмы *Fog Creek*, которое я подписываю собственноручно настоящей чернильной ручкой. Такое, очевидно, бывает настолько редко, что не может не привлечь *серьезного* внимания. В письме говорится, что у нас имеется интернатура и я лично приглашаю студента подать в нее заявление. Я отправляю электронные почтовые сообщения профессорам и выпускникам таких факультетов и кафедр, у кого обычно имеются своего рода почтовые списки рассылки студентов, специализирующихся по компьютерным наукам, и мои сообщения скорее всего перенаправляются этим студентам.

Через некоторое время у нас набирается много заявлений о приеме в интернатуру, и мы можем начинать сбор урожая. Последние пару лет я получал по 200 заявлений на каждое место. Обычно после нашего просеивания в этой стопке оставалось примерно 10 заявлений (на одну вакансию), и мы приглашали тех, кто их прислал, на телефонное собеседование. В результате этого собеседования мы, скорее всего, пригласим двух-трех человек лететь в Нью-Йорк уже для личного собеседования.

Вероятность, что человек, приглашенный на это собеседование, будет нанят, настолько велика, что теперь самое время начинать полномасштабную *вербовку*. Приглашенных встречает в аэропорту водитель лимузина, одетый в униформу. Погрузив их багаж, он быстро везет гостей в отель, вероятно, самый крутой из тех, что они когда-либо видели. Этот отель находится в районе модных магазинов. В него все время то входят, то выходят фотомодели, а сложные аксессуары ванной комнаты, вероятно, взяты из постоянной коллекции Музея современного искусства, но будет большой удачей, если вы угадаете, как ими чистить зубы. Дождавшись гостей в номере отеля, мы оставляем им подарочный пакет с тенниской, предлагаем пешеходную экскурсию по Нью-Йорку, подготовленную штатными сотрудниками *Fog Creek*, а также DVD с документальным фильмом об участниках летней интернатуры 2005 года[2]. В номере находится DVD-плейер, и мно-

[2] Копию этого фильма, созданного Лероном Д. Уилсоном, можно заказать по адресу `www.projectaardvark.com/movie`. Он называется *Aardvark'd: Twelve Weeks with Geeks*.

гие из гостей могут посмотреть, как “прикольно” было предыдущим интернам.

В конце дня собеседований мы приглашаем студентов, если они хотят посмотреть Нью-Йорк, остаться в нем на пару дней за наш счет. Затем лимузин отвозит их в отель; он же отвозит их в аэропорт, откуда гости летят домой.

Пусть где-то только один из трех кандидатов, дошедших до этапа личного собеседования, пройдет все наши собеседования, но по-настоящему важно то, что те, у кого это *получится*, приобретают положительный опыт. Даже те, кому не повезло, возвращаются назад в студенческий городок, думая, что компания *Fog Creek* — классный работодатель, и рассказывая всем своим друзьям, как было прикольно остановиться в роскошном отеле в городе “Большого яблока”; это заставит их следующим летом написать заявление в интернатуру — хотя бы для того, чтобы иметь шанс на путешествие.

Летом, во время самой интернатуры, студенты обычно начинают думать: “Ладно, эта летняя работа замечательная, накапливается хороший опыт, и, возможно, пока только *возможно*, здесь будет моя основная работа”. Ну а мы немного их опережаем. Лето мы собираемся использовать, чтобы решить, брать их на полную занятость или нет, а они — чтобы решить, хочется им работать у нас или нет.

Итак, мы даем им настоящую работу, причем тяжелую. Участники интернатуры всегда работают над созданием кода. Иногда они работают с самым крутым, какой только есть в компании, новым материалом, чему могут немного завидовать постоянные работники, но такова жизнь. Однажды летом группа из четырех интернов создала с нуля новый продукт Fog Creek Copilot (познакомьтесь с ним по адресу `copilot.com`). Та интернатура окупила себя за считанные месяцы. Впрочем, даже если они не создают новый продукт, они работают с настоящим, продаваемым кодом, за некоторые большие участки которого интерны отвечают целиком, лично (конечно, вместе с помогающими им опытными наставниками).

Еще мы стараемся обеспечить им прекрасное времяпрепровождение. Устраиваем вечеринки и дни открытых дверей. Даем им бесплатное жилье в довольно хорошем фирменном об-

щежитии, где они могут принимать друзей (подруг) из других компаний и учебных заведений. Каждую неделю у нас проводится что-то вроде внеплановых мероприятий или полевых вылазок: это бродвейские мюзиклы (в этом году студенты сходили с ума от *Avenue Q*), кинопремьеры, экскурсии в музеи, лодочная прогулка вокруг Манхэттена, игра с участием Yankees и, хотите — верьте, хотите — нет, одним из фаворитов этого лета было путешествие в Top of the Rock — высотное здание в центре Манхэттена, на крышу которого вы можете подняться. Вам, возможно, не кажется, что это так уж страшно, но это правда. Рядом со студентами на каждом мероприятии также находится несколько сотрудников *Fog Creek*.

К концу лета всегда появляется несколько интернов, убеждавших нас, что они по-настоящему прекрасные программисты и что мы просто обязаны их нанять. Заметьте, не все из них, но некоторые будут просто замечательными программистами, с которыми мы хотим продолжать сотрудничество, а другие, возможно, будут таковыми, но не в *Fog Creek*, а где-то в другом месте. Например, в нашей компании принята достаточная степень автономии, у нас нет многих руководителей среднего звена, и мы ожидаем от людей полной самостоятельности. Пару раз у нас работало по одному участнику летней интернатуры, которые прекрасно себя показали, если ими кто-то управлял, но в *Fog Creek* некому было этим заниматься в достаточной степени, и эти студенты попадали в затруднительные положения.

Как бы то ни было для тех, кого мы действительно хотим нанять, нет никакой пользы в ожидании. Поэтому им предлагается полная занятость как можно раньше, при условии, конечно, что они получили диплом. И это прекрасное предложение. Хочется, чтобы они могли снова приехать в свое учебное заведение, поделиться впечатлениями с друзьями и осознать, что у них первоначальная зарплата больше, чем у кого-либо другого.

Значит ли это, что мы переплачиваем? Вовсе нет. Видите ли, в средней зарплате за первый год надо учитывать риск, что человек не будет ее оправдывать. А мы уже проверили этих ребят и уверены, что они будут прекрасными специалистами.

Мы знаем их возможности. Таким образом, нанимая этих ребят, мы располагаем о них большей информацией, чем любой другой работодатель, который провел бы с ними только собеседование. Это значит, что им мы можем платить больше. Мы располагаем сравнительно лучшей информацией, поэтому хотим платить больше, чем работодатели без этой информации.

Если мы правильно сделали свою работу, а обычно это так, то с этого времени интерн полностью сдается и принимает наше предложение. Иногда приходится убеждать чуть дольше. Иногда им хочется испробовать и другие возможности, но выдающееся предложение от *Fog Creek* гарантирует, что в первый же раз, когда для собеседования в компании *Oracle* им надо будет встать в 8:00 утра и надевать костюм, а будильник уже успокоится, то скорее всего они скажут: "С какой стати я встаю в 8:00 утра и надеваю костюм, чтобы быть на собеседовании в *Oracle*, когда у меня уже есть прекрасная работа в *Fog Creek*?" И, я надеюсь, у них даже мысли не возникнет ехать на это собеседование.

Кстати, перед тем как продолжить, я должен кое-что прояснить насчет интернатуры в сферы информатики и разработке программ. В этот день и этот век в этой стране абсолютно все знают, что здесь интернатура *платная*, а соответствующая цена обычно достаточно высокая. И хотя в других областях (от издательской деятельности до музыки) распространена бесплатная интернатура, мы все же платим 750 долл. в неделю, плюс бесплатные жилье, обеды и проездные на метро, не говоря уже о возмещении расходов на переезд и всех имеющихся преимуществах. Да, долларов мы платим чуть меньше среднего, но с учетом бесплатного жилья получается, наоборот, чуть больше средней оплаты. Мне кажется, об этом надо говорить, потому что каждый раз, когда на своем Web-сайте я рассказываю об интернатуре, то кто-то обязательно путается и думает, что я пользуюсь рабским трудом или чем-то подобным. Эй ты, молокосос! Ну-ка дай мне охлажденного апельсинового сока, выжатого вручную, да сделай его побыстрее!

Интернатура создает трубопровод для прекрасных работодателей, но этот трубопровод достаточно длинный, и множество людей по пути теряется. Как показывают простые расче-

ты, нам приходится нанимать двух интернов, чтобы один из них стал сотрудником с полной занятостью, а если каждому из них осталось учиться еще год, то получится трубопровод длиной в два года между началом найма и первым днем их пребывания в качестве работников с полной занятостью. Это значит, что нам приходится каждое лето нанимать в интернатуру столько студентов, сколько физически поместится в наших офисах. Первые три лета мы пытались брать в интернатуру только тех студентов, которым осталось учиться год, но этим летом мы наконец поняли, что теряем некоторых прекрасных более молодых ребят, поэтому принимаем студентов любого года обучения. Хотите верьте, хотите — нет, но я даже думаю, каким образом привлечь еще и старшеклассников, чтобы они на деньги колледжей после занятий настраивали компьютеры. Я хочу это организовать просто для того, чтобы начинать налаживать связь со следующим поколением прекрасных программистов, даже если конвейер получится длиной в шесть лет. Да, у меня действительно далекоидущие планы.

Организация сообщества* (тяжело)

Идея состоит в том, чтобы организовать большое сообщество умных разработчиков-единомышленников, концентрирующихся вокруг вашей компании. В крайнем случае, когда бы у вас ни открылась вакансия, кандидаты на нее появятся сразу.

Это тот способ, каким, по правде говоря, мы и нашли наших замечательных сотрудников *Fog Creek*. Речь идет о моем личном Web-сайте Joel on Software (`joelonsoftware.com`). Его статьи читают миллионы людей, большинство из которых в той или иной степени являются разработчиками программ. Когда бы я ни объявил, что ищу того или иного специалиста, то благодаря огромной самоорганизующейся аудитории я обычно получаю довольно большую стопку очень хороших резюме.

Эта категория отмечена звездочкой, что означает “задача повышенной сложности”. В таких случаях мне кажется будто я даю вам совет: “Чтобы победить на карнавале красоты, надо,

во-первых, стать красивым и, во-вторых, присоединиться к карнавальному шествию". Дело в том, что на самом деле я не знаю, почему или как мой сайт стал таким популярным; почему люди, которые его читают, являются лучшими разработчиками программ.

Я действительно хочу вам помочь. Дерек Повачек написал хорошую книгу на эту тему[3]. Во многих компаниях прибегали к разным стратегиям использования блогов, и, к сожалению, немало из этих компаний не смогли создать какую-либо аудиторию. Поэтому могу сказать лишь одно: что принесло пользу нам, не обязательно принесет пользу вам, и я не знаю, что вы здесь можете сделать.

"Справочная работодателя": на мокром можно и поскользнуться

Стандартный совет о том, как найти прекрасных разработчиков программ, состоит в следующем: спросите об этом у тех разработчиков, которые у вас уже работают. Теоретически (конечно же) одни умные разработчики должны знать других умных разработчиков.

Может быть, они и знают, однако у них тоже есть очень дорогие друзья, которые бывают не очень хорошими разработчиками, и в этом поле около миллиона противопехотных мин, поэтому, говоря по правде, "справочную работодателя" я считаю в целом одним из самых ненадежных источников найма.

Немалый риск, конечно же, создают "соглашения о неконкуренции". Если вы думаете, что это неважно, то вот вам история с компанией *Crossgain*, из которой пришлось уволиться четверти сотрудникам, работавшим раньше в *Microsoft*, после того, как *Microsoft* пригрозила им персональными судебными

[3] Derek Powazek, *Design for Community: The Art of Connecting Real People in Virtual Places* (Berkeley, CA: New Riders, 2001).

исками[4]. Ни один программист в здравом уме не должен подписывать такое соглашение, но большинство из них все равно это делают, так как никогда не могут себе представить, что их заставят его исполнять, или потому, что не имеют привычки читать контракты, или они уже приняли предложение работодателя и перевезли свои семьи на другой конец страны, а увидели это соглашение в свой первый рабочий день, когда пытаться торговаться на эту тему уже несколько поздно. Поэтому они его и подписывают, хотя это один из самых отвратительных приемов работодателей; подобные соглашения *очень часто* обязательны к исполнению, и их заставляют исполнять.

"Соглашения о неконкуренции" могут означать, что если вы, чересчур доверившись "справочной", наймете группу людей от одного и того же работодателя, у которого они, как известно вашим работникам, занимали *первое* место как звезды программирования, тогда вы очень сильно рискуете.

Другая проблема состоит в том, что если вы, используя "справочную работодателя", проводите выборочный прием на работу, то ваши сотрудники даже не захотят говорить вам о своих настоящих друзьях. Никто не захочет убеждать друзей подать заявление о приеме только затем, чтобы этих друзей отвергли. Получается что-то вроде препятствия для дружбы.

Если они не захотят говорить о своих друзьях, и вам, возможно, не удастся нанять людей, с которыми они обычно работали, то вы от "справочной работодателя" получите не слишком много.

Однако *настоящая* проблема с этой "справочной" возникнет тогда, когда менеджеры по найму с зачаточным пониманием экономики решат предложить денежные бонусы за такие сведения (кстати, достаточно распространенное явление). Обоснование его состоит в следующем: наем хорошего специалиста с помощью "охотника за головами" или постороннего менеджера по найму может стоить 30–50 тыс. долл. Если вы можете платить своим работникам, скажем, бонус в 5000 долл.

---

[4] Jay Greene, "Crossgain vs. Microsoft: 'Mooning the Giant'", *Business Week*, February 5, 2001. Архивная копия находится по адресу `www.businessweek.com/2001/01_06/b3718158.htm`.

за каждого нанятого, которого они привели, или, возможно, дарить дорогой спортивный автомобиль за каждые десять "справок работодателю", то подумайте, сколько денег будет сэкономлено? А ведь 5000 долл. — это для сотрудников, живущих на зарплату, как целое состояние. Вот так возникает что-то вроде нездорового азарта.

Проблема здесь в том, что вы вдруг увидите, как закрутятся маленькие колесики, и работники начнут тащить на собеседования всех, о ком они только вспомнят, и у них будет по-настоящему сильный импульс к тому, чтобы этих людей наняли. Поэтому сотрудники начинают тренировать этих людей для собеседования; с сотрудниками, которые проводят собеседования, начинаются тихие переговоры в комнатах для совещаний, и внезапно весь ваш персонал пытается заставить вас нанять чьего-то приятеля по студенческому общежитию, который вам совсем не нужен.

Рассмотрим еще один бесполезный вариант. Компания *ArsDigita*, возникшая в период "дот-ком"-бума и затем пережившая депрессию, получила известность в значительной мере потому, что купила Ferrari, который поставили на стоянку и обещали любому, кто приведет десять новых сотрудников. Никто и близко не приблизился к заветному автомобилю, качество нанимаемых сотрудников упало, и компания развалилась. Вероятно, это произошло не из-за Ferrari, который, как оказалось, был взят в аренду, а также не из-за трюка для получения известности (хотя руководители явно были не против, чтобы в кои-то веки прокатиться на нем с ветерком).

Когда сотрудник компании *Fog Creek* предлагает кого-либо, кто мог бы прекрасно нам подойти, мы, возможно, хотели бы пропустить первоначальное собеседование по телефону, но, тем не менее, оно проводится *в любом случае*. Нам нужно, чтобы кандидаты прошли через одни и те же собеседования; мы придерживаемся одного и того же высокого стандарта и не даем кандидатам никаких поблажек.


PRIVATE
Red Bull
VOLT

Глава 3

Практическое руководство для разработчиков

Вы можете размещать рекламу во всех подходящих местах, иметь фантастическую программу интернатуры и проводить собеседования со всеми, с кем хотите, но если прекрасные программисты не хотят на вас работать, то их у вас, к сожалению, не будет. Данная глава представляет собой что-то вроде полевого руководства по разработчикам: что они ищут, что им нравится или не нравится на рабочем месте и какие ваши действия приведут к тому, что прекрасные программисты будут выбирать прежде всего вас.

Отдельные кабинеты

В прошлом году я был в Йеле на конференции по компьютерной науке. Один из докладчиков, ветеран Силиконовой Долины, который основал или возглавлял довольно большое число "стартапов", финансируемых венчурным капиталом, показывал книгу *Peopleware*[1].

— Вам надо читать эту книгу, — говорил он. — Это руководство по управлению программистской компанией. Это самая важная книга о том, как надо управлять такими компаниями.

Должен с ним согласиться: *Peopleware* — это прекрасная книга. Одно из самых важных и наиболее спорных ее положений состоит в следующем: если хотите, чтобы программисты были производительными, дайте им побольше тихого места,

[1] Tom DeMarco, Timothy Lister, *Peopleware: Productive Projects and Teams*, Second Edition (New York: Dorset House, 1999); Том Демарко, Тимоти Листер, *Человеческий фактор: успешные проекты и команды*. — Символ-Плюс, 2005.

а возможно, и отдельные кабинеты. Авторы, Демарко и Листер, твердят об этом снова и снова.

После того как речь закончилась, я подошел к докладчику.

— По поводу *Peopleware* я с вами согласен, — говорю я. — Только скажите мне: были ли отдельные кабинеты у разработчиков во всех ваших "стартапах"?

— Конечно же, нет, — сказал он. — Венчурный капитал никогда на это не пойдет.

— Хм-м-м. Но, возможно, это самое важно, что есть в книге, — говорю я.

— Тем не менее вы все равно проиграете. Ведь для венчурного капитала отдельные кабинеты выглядят как разбазаривание вами его денег.

В Силиконовой Долине сложилась мощная культура, в соответствии с которой надо собирать множество программистов в большом открытом пространстве, хотя факты свидетельствуют, что программисты с отдельными кабинетами демонстрируют намного высшую производительность. Но программистам и им подобным *нравится* быть среди людей, даже в ущерб своей производительности, поэтому в этом вопросе со мной не соглашаются, и приходится вести тяжелую борьбу.

Я даже слышал, что программисты говорят примерно так:

— Конечно, мы все работаем в каморках, но так работает *каждый* — вплоть до генерального директора включительно!

— Генерального директора! Так что, он действительно работает в каморке?

— Да, каморка у него *есть*, но сейчас то, о чем ты говоришь, — это комната для совещаний, в ней он проводит все важные встречи...

М-м-да-а. Довольно распространенное в Силиконовой Долине явление — генеральный директор устраивает большое шоу, показывая, что он работает в каморке, "как все", хотя так или иначе есть еще комната для совещаний, которую он обычно делает своей собственной. (Он будет утверждать, что комната нужна "только для обсуждения чего-то секретного", но в половине случаев, когда вы проходите мимо, в ней нет никого, кроме генерального, беседующего по телефону со своим при-

ятелем по игре в гольф, а на столе у него лежит каталог модной фирмы *Cole Haan*.)

Как бы то ни было, мне не хочется снова дискутировать, почему отдельные кабинеты делают разработчиков более производительными[2], или почему программиста осеняет меньше полезных озарений, стоит ему лишь надеть наушники с музыкой, чтобы заглушить окружающий шум[3], или почему выделение разработчикам отдельных кабинетов фактически обойдется ненамного дороже, чем их размещение каким-то другим способом[4].

Нет, сейчас речь идет о найме и о той роли, которую в нем играют отдельные кабинеты.

Итак, что бы вы ни думали о производительности и о равенстве рабочих мест, но два момента бесспорны.

1. У отдельных кабинетов более высокий статус.
2. Каморки и другое общее пространство бывают неудобны в смысле размещения людей.

Из этих двух фактов следует, что программистам нравится получать работу там, где они будут сидеть в отдельных кабинетах, — особенно там, где есть дверь, которую можно закрывать, окно и прекрасный вид из этого окна.

Теперь, к сожалению, надо признать: кое-что из того, что облегчает наем, на самом деле находится за пределами вашей компетенции. Даже генеральные директора и основатели, зависящие от венчурного капитала, не могут предоставлять личные кабинеты. Многие компании переносят или реорга-

---

[2] Tom DeMarco, Tim Lister, "Programmer Performance and the Effects of the Workplsace", *Proceedings of the 8th International Conference on Software Engineering* (London: IEEE Computer Society Press, 1985); Capers Jones, "How Office Space Affects Programming Productivity", *IEEE Computer*, January 1995, p. 7676; Gerald M. McCue, "IBM's Santa Teresa Laboratory — Architectural design for program development", *IBM Systems Journal*, January 1978.

[3] Tom DeMarco, Tim Lister, *Peopleware*, Second Edition, p. 78ff.

[4] Joel Spolsky, "Bionic Office", опубликована в `www.joelonsoftware.com`, 23 сентября 2003 года (ищите по слову "Bionic").

низуют свое офисное пространство только через пять–десять лет. А для мелких “стартапов” отдельные кабинеты — это вообще непозволительная роскошь. Итак, по моему мнению, во всех компаниях, за исключением самых передовых, всегда найдут кучу отговорок, чтобы буквально не дать разработчикам отдельные кабинеты. Да и в самых передовых компаниях решение о том, куда переходить и где люди должны работать, часто принимается раз в десять лет комитетом, состоящим из секретаря, офис-менеджера и младшего сотрудника большой архитектурной фирмы, который склонен верить сказкам архитектурных школ, что “открытость пространства означает открытость компании” или чему-то там еще. В этом комитете почти совсем не участвуют разработчики или команда разработчиков.

Отдельные кабинеты *не являются* чем-то таким невозможным, хотя я веду нешуточную борьбу. Отдельные кабинеты мы старались дать в большинстве случаев всем нашим программистам, работающим полную неделю (даже в Нью-Йорке, где самая высокая в стране арендная плата), и нет сомнения, что это делает людей, работающих в *Fog Creek*, намного более счастливыми. Поэтому если вы все равно хотите этому сопротивляться, *делайте, как хотите*, — благодаря вам я буду сохранять конкурентное преимущество.

Физическое рабочее пространство

Теперь от отдельных кабинетов перейдем к физическому рабочему пространству. Когда кандидат приходит в вашу компанию на собеседование, он хочет посмотреть, где люди работают, и попытаться представить себя там же. Если офисное пространство чудесное, светлое, окружено прекрасными соседями, а все вокруг новое и чистое, то кандидата посетят счастливые мысли. Однако если офисное пространство загромождено, ковры протертые, стены не покрашены, развешаны плакаты с командами гребцов и словами РАБОТА В КОМАНДЕ, напечатанными крупными буквами, то мысли у кандидата бу-

дут, как у Дилберта (героя карикатур об офисной жизни. — *Примеч. пер.*).

Впрочем, многие технари на удивление безразличны к условиям, окружающим их в офисе. На самом деле люди, привыкнув к офису, могут не замечать его вопиющих недостатков, хотя в другом случае им требовались бы прекрасные условия работы.

А теперь поставьте себя на место ваших кандидатов и честно подумайте над следующим.

- Что они подумают о вашем месторасположении? Насколько лучше Буффало звучит по сравнению, скажем, с Остином? Так ли уж людям хочется переезжать в Детройт? Если вы находитесь в Буффало или Детройте, то можете ли вы хотя бы пытаться делать большую часть своих собеседований в сентябре?
- Какое они вынесут впечатление от пребывания в вашем офисе? Что они увидят? Чистое и восхитительное место? Гостевой холл в виде портика с фонтаном и растущими пальмами либо что-то напоминающее стоящую в грязном переулке государственную стоматологическую поликлинику с увядшими стеблями кукурузы и старыми номерами журнала *Newsweek*?
- На что похоже рабочее пространство? Все в нем новое и сияющее? Или вы все еще видите гигантский, пожелтевший призыв вступать в TEAM BANANA, распечатанный на бумаге, складывающейся гармошкой, с помощью точечно-матричного принтера, причем распечатанный еще тогда, когда были привычными такие принтеры и такая бумага?
- На что похож стол? Имеют ли программисты по несколько больших плоских экранов или по одному ЭЛТ-монитору? Какие у них кресла — Aeron или Staples Special?

Давайте на минуточку отвлечемся и поговорим о знаменитых креслах от компании *Aeron*, созданных Германом Миллером. Они стоят где-то по 900 долл. штука. Это на 800 долл. больше, чем за дешевые офисные кресла от *Office Depot* или *Staples*.

Кресла Aeron *намного* удобнее своих дешевых собратьев, и если эти кресла правильно подобраны по размеру и как положено подогнаны, то большинство людей могут, не чувствуя неудобства, сидеть на них целый день. Спинка и сидение сделаны из своего рода сетки, поддерживающей циркуляцию воздуха и не дающей вам потеть. Чудесна эргономика, особенно в новых моделях с опорой для поясницы.

Они служат дольше, чем дешевые кресла. Наша компания работает уже шесть лет, и в ней каждое кресло Aeron выглядит, честное слово, как новое. Пусть кто-нибудь найдет, чем отлича-

ются кресла, купленные нами в 2000 году и три месяца назад. Они легко могут служить десять лет, а дешевые кресла буквально начинают разваливаться через считанные месяцы. Вам потребуется как минимум четыре дешевых кресла по 100 долл., чтобы они протянули столько, сколько одно кресло Aeron.

Получается, что одно такое кресло, служащее больше десяти лет, в действительности стоит на 500 дол. больше, что составляет 50 долл. в год. Получается доллар в неделю на одного программиста.

Примерно доллар стоит рулон хорошей туалетной бумаги. Ваши программисты, вероятно, расходуют по одному рулону в неделю на человека.

Итак, пересадка программистов в кресла Aeron обойдется буквально во столько, сколько вы тратите на их *туалетную бумагу*. Я уверен, что если вы захотите поднять в бюджетном комитете вопрос о туалетной бумаге, то вас решительно попросят не мешать: есть что обсуждать и поважнее.

На кресло Aeron, к сожалению, бросает тень его репутация экстравагантности, особенно для "стартапов". Позор, что оно каким-то образом стало символом всех денег венчурного капитала, растранжиренных во время бума "дот-ком". Ведь кресло Aeron не очень дорогое, если учесть, как долго оно служит. И действительно, если подумать о восьми часах в день, которые вы проводите, сидя в нем, даже в самом дорогом из серии выпущенных кресел с поддержкой поясницы и *подлокотниками*, то оно оказывается таким дешевым, что, покупая его, вы фактически *зарабатываете* деньги.

"Игрушки"

Похожая логика относится и другим игрушкам разработчика. Просто нет причины, чтобы не снабдить ваших разработчиков новейшими компьютерами, имеющими как минимум по два больших (21 дюйм по диагонали) ЖК-монитора (или один монитор с 30 дюймами по диагонали), а также не разрешить, чтобы они могли заказывать на Amazon.com любую нужную им техническую книгу. Это очевидные средства усиления производительности, но для темы нашего обсуждения

более существенно то, что это очень важные инструменты найма, особенно для мира, где во многих компаниях считают программистов “мелкими сошками”, “машинистками”, и действительно зачем вам такой большой монитор, и что плохого в ЭЛТ-экранах с 15 дюймами по диагонали? Когда *я* был еще ребенком, ...

Общественная жизнь разработчиков

Разработчики программ ничуть не отличаются от обычных людей. Конечно, мне известно, что сейчас разработчиков принято считать типичными чудаками с синдромом Аспергера, целиком отключенных от межличностных отношений, но это не совсем так, и даже люди с этим нарушением развития заботятся об общественной стороне рабочего пространства, которая включает в себя следующее.

Как относятся к программистам в организации?

Они “шишки” или “машинистки”? Руководство компании состоит из инженеров или бывших программистов? Отправляясь на конференцию, разработчики летят первым классом? (Меня не заботит, что это выглядит расточительством. “Звезды” перемещаются первым классом. Как правило.) Когда они прилетают на собеседование, в аэропорту их подбирает лимузин, или им нужно добираться в офис самостоятельно? При прочих равных условиях разработчики предпочтут организацию, где к ним относятся как к “звездам”. Если ваш генеральный директор — это ворчливый бывший продавец, который “не понимает, чего эти разработчики-примадонны все требуют какие-то подпорки для кистей рук, большие мониторы и комфортабельные кресла, и за кого они себя держат?”, тогда вашей компании, скорее всего, нужно сменить род деятельности. Вам явно не нужны великие разработчики, если вы их не уважаете.

Кто их коллеги?

В день собеседования программисты особое внимание обращают на сотрудников, с которыми встречаются. Хорошие ли они? И, что еще важнее, способные ли они? Как-то я проводил летнюю интернатуру в компании *Bellcore*, филиале *Bell Labs*, и каждый, с кем я встречался, повторял снова и снова: "В *Bellcore* прекрасно работается благодаря ее сотрудникам".

Скажем так, если у вас работают ворчливые разработчики, от которых нельзя просто так избавиться, то хотя бы отстраните их от собеседований, которые должны обязательно проводиться веселыми, компанейскими людьми, похожими на руководителей круизов. Постоянно напоминайте себе, что кандидат, вернувшись домой и решая, где ему работать, будет вспоминать о вашей компании не слишком хорошо, если в ней ему попадались одни только мрачные личности.

Кстати, первоначальное правило найма в *Fog Creek*, "украденное" у *Microsoft*, состояло в следующем: "Будь способным и доводи дело до конца". Но до того как была основана компания, мы поняли, что требуется еще одно правило: "Не строй из себя крутого"[5]. Если вспомнить прошлое, то для получения работы в *Microsoft* следовать этому правилу было *необязательно*. И хотя я уверен, что в этой компании пусть неискренне, но заявляли, как важно людям хорошо относиться друг к другу, в конечном итоге там никого еще не отстранили от работы только потому, что человек строил из себя "крутого". В действительности это качество сотрудника, возможно, является *предварительным условием* того, чтобы он выбился наверх. Интересы производства от этого не страдают, но страдают интересы найма. Кому захочется работать в компании, где терпимы к таким людям?

---

[5] Robert I. Sutton, *The No Asshole Rule: Building a Civilized Workplace and Surviving One That Isn't* (New York: Warner Bisiness Books, 2007).

Независимость и автономность

Еще в 1999 году, до того как основать компанию *Fog Creek Software*, я увольнялся со своей работы в городе Джуно и прошел собеседование с кадровиком, которое обычно проводят со всеми увольняемыми. И тут я попал в ловушку, рассказав ему все, что было не так с управлением компанией. Мне было точно известно, что из-за этого я не получу ничего хорошего, а могу только пострадать, но, тем не менее, я это сделал, и главное, на что жаловался, — на стиль руководства "наскоками". Видите ли, большую часть времени начальники оставляли людей в покое, чтобы те спокойно делали свою работу, но время от времени они вмешивались по мелочам, требуя что-то сделать именно так, как они говорят, и не иначе, а затем, не ожидая результатов этого фарса, довольно быстро переключались на мелочное руководство какой-то другой задачей. Например, я помню особенно надоедливые два-три дня, когда все, от моего начальника и до генерального директора, занимались тем, что указывали мне, как именно вводятся данные в анкету при приеме на работу. У них не было опыта работы в качестве разработчиков пользовательских интерфейсов, и они уделили не слишком много времени на разговор со мной, чтобы понять, почему в том конкретном случае я все-таки был прав. Впрочем, дело здесь в том, что начальство просто "уперлось" в этом вопросе и даже не удосужилось выслушать мои доводы. Вывод, сделанный после разговора с генеральным директором, состоял в том, *что не надо было болтать*.

Обычно, если вы собираетесь нанимать способных людей, позвольте им применять свою квалификацию в работе. Начальники могут советовать, и это правильно, но им следует быть крайне осторожными, чтобы их "совет" не воспринимался как команда, так как по любому конкретному техническому вопросу начальство, скорее всего, знает меньше, чем сотрудники "на передовой", особенно если вы, как я уже сказал, нанимаете способных людей.

Разработчикам хочется, чтобы их нанимали за их квалификацию, относились к ним как к экспертам и разрешали принимать самостоятельные решения в рамках их собственного опыта.

Никаких интриг

В действительности интриги есть везде, где находится не меньше трех человек, и это явление может быть абсолютно безвредным. На самом деле говоря "никаких интриг", я имел в виду "никаких злостных интриг". У программистов очень обостренное чувство справедливости. Программный код или работает или нет. Нет смысла спорить, есть программная ошибка или нет, ведь ее можно найти, протестировав код. Мир программирования очень справедливый и очень жестко упорядоченный. Очень много людей идут в программирование прежде всего потому, что предпочитают находиться в справедливом и упорядоченном месте, где жестко правит система отбора по заслугам и в любом споре победа будет за вами просто потому, что вы *правы*.

Именно такого рода среду вам следует создать, чтобы привлечь программистов. Когда программист жалуется на "интриги", то это означает (если быть очень точным) любую ситуацию, когда технические соображения перевешиваются личными. Ничто так не выводит из себя, как ситуация, когда разработчику дают указание применять тот или иной язык программирования не потому, что этот язык лучше всего подходит для решения конкретной задачи, а потому, что нравится начальнику. Ничто так не раздражает, как ситуация, когда людей продвигают не исключительно по заслугам, а за способность заводить связи. Ничто так не злит разработчика, как ситуация, когда он вынужден делать что-то более худшее в техническом отношении, так как на этом настаивает тот, кто имеет в организации более высокое положение, или тот, кто имеет больше связей.

Ничто так не удовлетворяет, как победа в споре благодаря техническим заслугам, даже если по интригам вы в нем проиграете. Когда я начинал работать в *Microsoft*, там был громадный, плохо управляемый проект с рабочим названием Macro-Man. Целью этого проекта было создание графического языка для программирования макросов. Этот язык очень бы разочаровал настоящих программистов, так как из-за своей графической природы фактически не давал бы использовать циклы

или условия. Кроме того, он был бы фактически бесполезен и для непрограммистов, которые, как мне кажется, не привыкли думать алгоритмами и вначале вряд ли бы его понимали. Когда я пожаловался насчет MacroMan, то мой начальник ответил: “Брось ты. *Этот поезд* уже не остановить”. Но я продолжал спорить, спорить и снова спорить. И хотя я был недавним выпускником колледжа, имевшим связей меньше, чем любой другой сотрудник *Microsoft*, но со временем люди прислушались к сути моих аргументов, и проект MacroMan был закрыт. Тут дело не в том, кем я был, а в том, что я был прав. Вот такого рода организация без интриг и восхищает программистов.

В общем, для создания здорового и приятного места для работы, которое будет привлекать и удерживать программистов, очень важно уделять особое внимание общественному климату в организации.

Над чем я работаю?

До некоторой степени один из самых лучших способов привлечь разработчиков — дать им работать над чем-то интересным.

Вот это изменить, возможно, тяжелее всего. Ведь если вы работаете над созданием программ для промышленности по производству гравия и песка, то над этим вы и работаете, т.е. не можете притворяться крутым “Web-стартапом” только лишь для привлечения разработчиков.

Кроме того, разработчикам нравится работать над чем-то достаточно простым или достаточно популярным, что можно было бы объяснить тете Ирме на День Благодарения. Конечно, тетя Ирма, будучи физиком-ядерщиком, на самом деле не слишком разбирается в написании программ на языке Ruby для промышленности по производству гравия и песка, зато симуляторы для гидродинамики не оставляют ее равнодушной.

Наконец, многие разработчики обращают внимание на общественную значимость компании, в которой работают. В связи с этим популярна работа в общественных железнодорожных компаниях и компаниях по созданию блогов, так как она помогает связать людей друг с другом и вроде бы не загрязня-

ет окружающую среду. Намного менее популярна работа в сомнительных с точки зрения этики компаниях, в которых заправляют финансовые махинаторы, или в производстве вооружений.

К сожалению, я вряд ли смогу придумать какой-то рецепт для среднего менеджера по найму, помогающий ему что-то сделать в этом отношении. Вы можете попробовать изменить компоновку ваших продуктов и сделать что-то "крутое", но это не приведет к чему-то уж очень радикальному. Впрочем, я видел, как некоторые компании используют в этой области сразу несколько приемов.

Разрешите лучшим кандидатам что-то выбирать самим

В течение многих лет корпорация *Oracle* вела программу MAP (Multiple Alternatives Program, Программа множества вариантов). Она предлагалась выпускникам колледжей из каждого класса, которые казались самыми вероятными кандидатами на работу в корпорации. Идея состояла в том, чтобы они приехали в *Oracle*, недельку-другую осматривались вокруг, посещали все группы с вакансиями, а затем выбрали любую понравившуюся им вакансию.

В компании *Google* также разрешается программистам очень сильно дрейфовать в том, над чем они работают. Там разработана концепция "20% времени", которая разрешает каждому сотруднику, не получая ничьего разрешения, уделять 20% своего времени любому интересующему его проекту.

Крутые новые технологии не обязательны

Считается, что для программистов большие инвестиционные банки Нью-Йорка — далеко не курорт. Условия работы ужасные плюс долгие часы работы, шум и начальники-тираны. Там программисты — это явно граждане третьего сорта по сравнению с помешанными от тестостерона кривляками, которые, в сущности продавая и покупая финансовые инструменты, являются в корпорации кронпринцами со своими бо-

нусами по 30 млн. долл. и всеми чизбургерами, какие только могут съесть (и за которыми они часто посылают проходящего мимо программиста). Впрочем, такое мнение — не более чем стереотип, ну а, чтобы удержать лучших программистов, в инвестиционных банках применяют две стратегии — платят море денег и разрешают программистам фактически свободно переписывать все снова и снова на любом из новых крутых языков программирования, какой им заблагорассудится изучать. А не переписать ли на Ruby то финансовое приложение? Да что угодно, только принеси мне этот чертов чизбургер.

Некоторые программисты не могут легкомысленно относиться к языку программирования, на котором работают, но большинству просто нравится возможность поработать с увлекательными новыми технологиями. Сегодня ими может быть Python или Ruby on Rails. Три года назад это был C#, а до него — Java.

Сейчас я не советую, чтобы вы использовали для своей работы лучший инструмент, и не советую переписывать программы каждые два года на какой-либо изысканной *языковой новинке*, но если вы можете сделать так, чтобы разработчики получили опыт работы с новыми языками, структурами и технологиями, то эти разработчики будут еще счастливее. Даже если вы не посмеете переписать свои самые главные приложения, то почему внутренние инструменты или менее важные новые приложения нельзя было бы создавать на увлекательном новом языке в качестве учебного проекта?

Могу я определиться с компанией?

Большинство программистов не хотят просто отбывать повинность. Им не нравится высиживать на работе, а, наоборот, хочется ощущать, что их труд что-то значит. Они хотят отождествлять себя со своей фирмой. Юных программистов особенно привлекают компании с идеологией. Множество компаний связаны с движением за открытый исходный код или за бесплатное программное обеспечение, и этим они привлекательны для разработчиков-идеалистов. Другие компании ассоциируются с общественными начинаниями или создают

продукт, который в некоторой степени может считаться или быть действительно полезным для общества.

Ваша работа как рекрутера состоит в том, чтобы узнать идеалистические аспекты вашей компании и проверить, интересуют ли они кандидатов.

Некоторые компании даже стремятся создать свои собственные идеологические движения. Расположенный в районе Чикаго "стартап" *37signals* крепко связал себя с идеей простоты: простые, легкие в использовании приложения, например, Backpack, и простая, легкая в использовании, структура Ruby on Rails.

Для *37signals* простота — это "-изм", практически международное политическое движение. Простота является не только простотой, нет, это летнее время, прекрасная музыка, а также мир, справедливость и счастье, ну и красивые девушки с цветами в волосах. Дэвид Хайнемайер Хэнссон, создатель Rails, говорит, что их история — это история "красоты, счастья и мотивации. Чувства гордости и удовольствия в работе и в инструментах. Эта история — не просто банальность, а тенденция. История, которая позволяет таким словам как "страсть" и "энтузиазм" входить в разрешенный словарь разработчиков без необходимости извиняться за себя. Или смущаться тому, на что действительно похоже сделанное тобой"[6]. Поднятие Web-программирования до уровня "красоты, счастья и мотивации", возможно, похоже на высокомерие, но это очень привлекательно и по-настоящему выделяет компанию *37signals* среди прочих. Распространяя рассказ о Ruby on Rails как о счастье, представители этой компании практически гарантируют, что хотя бы некоторые разработчики будут искать работу, связанную с этой структурой программирования.

Впрочем, в таком деле, как кампания по управлению идентичностью, фирма *37signals* пока еще новичок. Она не выдерживает *сравнения* с *Apple Computer*, которая с помощью единственной рекламы на Суперкубке 1984 года смогла *до нынешнего дня* удержать свою позицию как контркультурной силы

[6] David Heinemeier Hansson, "Rails steps into year three", `www.loudthinking.com/arc/000594.html`, Август 6, 2006.

свободы против диктатуры, свободы против подавления, цветовой палитры против черно-белого, хорошеньких женщин в ярко-красных шортах против мужчин в костюмах и с промытыми мозгами. Неважно, в сущности всего этого лежит пародия на Оруэлла: гигантская корпорация манипулирует своим общественным имиджем таким способом, который даже не имеет смысла (это вроде бы компьютерная компания — так при чем здесь противостояние диктаторским режимам?), а еще корпорация успешно насаждает такую культуру личности, которая заставляет покупателей компьютеров по всему миру чувствовать, что они не просто делают покупку, а *присоединяются к движению*. Покупая iPod, вы, конечно же, поддерживаете Ганди против британского колониализма. Каждый покупаемый вами MacBook противостоит диктатуре и голоду!

Как бы то ни было, глубоко вздохните... Настоящий смысл этой главы — заставить вас подумать, что символизирует ваша компания, как это воспринимается и как еще может быть воспринято. Манипулирование вашим корпоративным брендом так же важно для найма, как и для маркетинга.

О чем не надо беспокоиться программистам

Они не должны беспокоиться о деньгах, если только вы не пустили деньги на другие цели. А если вы начинаете слышать жалобы насчет зарплат там, где никогда их не слышали, то обычно это признак того, что на самом деле людям не нравится их работа. Когда потенциальные новые работники просто так не отступают от требований непомерных зарплат, то вы, возможно, имеете дело с людьми, которые думают: “Ну, если на работе придется выкладываться, то хотя бы за хорошую оплату”.

Это не значит, что вы можете недоплачивать людям, так как они очень ратуют за справедливость и приходят в бешенство, когда узнают, что за одинаковую работу разные люди получают разную зарплату или что в вашем офисе все зарабатывают на 20% меньше, чем в таком же офисе вдоль дороги к центру города. Тогда деньги вдруг становятся большой проблемой. Конечно, вам надо платить конкурентоспособные зарплаты. Но (и с этим согласится каждый) изо всего того, что

взвешивают программисты, решая, где работать, зарплаты (если только они в целом справедливые) займут в списке приоритетов на удивление скромное место. Предложение высоких зарплат будет на удивление неэффективным инструментом в решении таких проблем, как, например, то, что программистам выдают 15-дюймовые мониторы, что на них все время вопят продавцы и что работа ведет к созданию атомного ружья на детенышей тюленей.


IN
RESUMATIC 2000
OUT
BOB
BOB

Глава 4

Сортировка резюме

Стандартное заявление о приеме на работу, сопроводительное письмо и резюме — это удивительно слабый способ знакомства с кандидатом. Вы получаете только самое смутное представление о его качестве.

Впрочем, иногда резюме дает достаточно сильное *отрицательное* представление, которое позволяет сразу же отсеивать кандидатов. Однажды я получил резюме от того, кто называл себя экспертом по программированию в Microsoft Windows [*именно так*]. В другой раз единственным предыдущим местом работы, указанным в заявлении, была компания *Dunkin' Donuts* ("Пончики Данкина"). Это резюме довольно тщательно следовало всем рекомендациям, которые любят раздавать советчики-специалисты в том, как делать карьеру выпускникам средних школ (например, приславший резюме парень "управлял подносами с пончиками"), но там не было ни малейшего намека, что кандидат когда-либо видел компьютер.

Однако из других резюме крайне трудно что-либо подробно узнать о кандидате. Поэтому мы в *Fog Creek* следуем такой политике по отношению к резюме.

1. Стараемся быть избирательными в рекламе наших вакансий, чтобы ограничить количество "балласта" в пачке резюме.
2. На основе резюме, ясное дело, не нанимаем, а только *отсеиваем* кандидатов, чтобы уменьшить количество тех, с кем надо проводить собеседование.
3. Для сортировки оставшихся резюме и таким образом решения, в каком порядке проводить собеседование, используем строго объективную систему просмотра и оценки, чтобы по меньшей мере быть справедливыми и последовательными в понимании того очень слабого сигнала, который нам подают резюме.

Критерии сортировки резюме

Есть несколько достаточно объективных критериев, которых мы и придерживаемся. Опять же, они предназначены *исключительно для сортировки резюме*, поэтому первые, кому мы позвоним, — это, вероятно, те, чьи резюме оказались в самом начале пачки.

Страсть

Нам нужны свидетельства того, что кандидат "пылает страстью" к компьютерам и по-настоящему любит программирование. Вот типичные свидетельства этого.

- Работа с компьютерами или опыт программирования с очень раннего возраста. Прекрасные программисты, скорее всего, проводят лето в компьютерном лагере или создают для своего дяди-дантиста онлайн-органайзер, а не работают над упаковкой одежды в Banana Republic.
- Внеклассная работа. Те, кто любят программирование, в свободное время часто работают над своими программными проектами (или участвуют в каком-либо проекте по созданию открытых исходных кодов).
- Восторг в сопроводительном письме по поводу того, как их тронула до слез книга "Структура и интерпретация компьютерных программ"[1].
- Иногда некоторые языки программирования и технологии, перечисляемые в резюме, указывают, что кто-то любит программирование настолько, что занимается изучением новых технологий. Когда были написаны эти слова, упоминание в резюме о Rubie было хорошим признаком программиста, которого именно страсть к программированию побуждает изучать самые последние новинки и пробовать

[1] Hal Abelson, Gerald Jay Sussman, *Structure and Interpretation of Computer Programs* (Cambridge, MA: MIT Press, 1985), Second Edition, 1996; Х. Абельсон, *Структура и интерпретация компьютерных программ*. — Добросвет, 2006.

совершенствовать квалификацию, ведь Rubie пока что требуется не слишком большому числу работодателей. Правда, здесь надо быть осторожным; в 1996 году упоминание в резюме о Java было признаком такой же страсти, а сейчас оно почти не добавляет ничего нового.

Серьезность намерений

Мы внимательно ищем в сопроводительном письме свидетельство того, что кандидат по-настоящему хочет работать у *нас*. Нам не требуется стандартное сопроводительное письмо, рассказывающее о кандидате то, то и то. Нет, мы хотим найти связное обоснование того, почему кандидат серьезно подумал и решил, что *Fog Creek* — именно то место, где он хочет работать. Есть две серьезные причины использовать такой критерий. Во-первых, это знак того, что кандидат одновременно с нами не обратился еще и в несколько сотен организаций. То, что он нашел время узнать о *Fog Creek* и написать сопроводительное письмо специально для нас, означает, что кандидат, хорошо изучив свои способности, обратился всего лишь к нескольким работодателям, а не сделал массовую рассылку, отправив тысячу обращений. Отправка тысячи резюме может быть признаком безрассудства. Еще важнее следующее: сопроводительное письмо, отправленное именно вам, — признак того, что на ваше предложение, если *только* оно последует, кандидат, скорее всего, ответит согласием. Это увеличит ваш "урожай". Если у меня есть время на собеседование только с шестью кандидатами, то, при прочих равных условиях, я его скорее потрачу на собеседование именно с этими кандидатами, которые по-настоящему хотят работать в *Fog Creek*, а не с теми шустрыми личностями, что обратились еще и в сотню других мест. Это, конечно же, при прочих равных условиях.

Владение литературным языком

Решение сортировать резюме по умению их авторов владеть английским нам было трудно осуществить, так как программирование — это одна из областей, где иммигрант, не говорящий на английском, все равно может быть блестящими программистом.

Но благодаря, скажем так, годам работы с программистами я знаю, что те из них, которые ясно излагают свои идеи, обычно во много раз эффективнее программистов, которые по-настоящему хорошо общаются только с компилятором. Это очень важно для документирования кода, для написания спецификаций и технической проектной документации, просматриваемых другими людьми, и очень важно даже для тех встреч, где вокруг вас спорят, как сделать что-то лучше всего. И здесь от блестящих программистов, затрудняющихся объяснить свои идеи, будет просто мало пользы. Данная категория подразумевает сортировку резюме по аккуратности и упорядоченности. Сумбурное резюме, кишащее грамматическими ошибками, в котором нет никакого порядка, — это явственный признак неорганизованного мыслителя или просто общей неаккуратности. Для многих видов деятельности такое вполне подходит, но только не для разработки программ. В частности, мы обычно полностью "дисквалифицируем" резюме, полные языковых ошибок. Ведь даже тем, для кого английский не родной, не составит труда найти человека, который проверит их резюме. Отсутствие такой проверки — обычный признак того, что кому-то глубоко безразлично качество его работы. А мы, скажем так, стараемся идти навстречу людям, для кого английский не родной, лишь бы только они прекрасно на нем говорили, и для нас не причина для отказа ни очаровательная восточно-европейская манера пропускать артикли, ни (также очаровательная для США) тихоокеанско-северозападная манера начинать каждый пункт словом "Итак".

Мозги

Сортировка по данной категории — это поиск признаков того, что кандидат, в общем-то, соображает или хотя бы относится к тем "головоломам", которые бывали в математических лагерях. К признакам всего этого относятся высокие оценки за учебу, высокие результаты стандартных тестов, членство в таких уважаемых обществах как Фи–Бета–Каппа, участие в соревнованиях по спортивному программированию TopCoder, в шахматных турнирах высокого уровня или в соревнованиях по программированию, проводимых *Association for Computing Mashinery*.

Пройденные конкурсы

А еще мы ищем в резюме признаки того, что кто-то в прошлом уже выдержал высокий конкурс. Конечно, не всякого выпускника колледжей "Лиги плюща"[2] следует принять и не всякого выпускника окружного колледжа —отвергнуть, но сам факт приема в колледж с очень высоким конкурсом говорит хотя бы о том, что таким способом *кто-то где-то* уже вас оценивал и решил, что вы достаточно способны. В нашей компании обычно считается, что критерию конкурентоспособности соответствуют те, кто был отобран в колледж или допущен к программе, принимающих меньше 30% кандидатов (в США таких колледжей примерно 60), либо работал в компании, которая известна трудным отборочным процессом, например, собеседованиями в течение целого дня. А если кто-то на военной службе попал на курсы с высоким конкурсом (например, в летные школы, на офицерские курсы и т.д.) или был принят в подразделения с высоким конкурсом, например морской пехоты, то это указывает на то, что кто-то прошел через трудную процедуру отбора среди кандидатов, и в любом случае это хороший признак.

Самое тяжелое

Для опытных программистов некоторые технологии более тяжелые, чем все остальные, — просто потому, что с ними, скажем, труднее работать как положено. Опять же, это довольно слабый индикатор, но при прочих равных условиях человек, работавший на OCaml, произвел бы на меня большее впечатление, чем тот, кто работал на Java. Работа с ассемблером, драйверами устройств или ядром в общем-то производит

[2] *Ivy League* — объединение восьми старейших привилегированных учебных заведений на Северо-Востоке США: Корнельский университет в Итаке, Университет Брауна в Провиденсе, Колумбийский университет в Нью-Йорке, Дартмутский колледж в Ганновере, Гарвардский университет в Кембридже, Принстонский университет в Принстоне, Пенсильванский университет в Филадельфии, Йельский университет в Нью-Хейвене.

большее впечатление, чем работа на Visual Basic или PHP. Ну а C++ с ATL тяжелее, чем Perl. Те, кто работали над операционными системами или компиляторами, выглядят более "солидными", чем работавшие над простыми клиентскими программами баз данных.

Конечно, это покажется подстрекательством; к тому же последние пять лет мой опыт программирования в основном ограничивался VBScript. Это нечто вроде языка Visual Basic, лишенного основных качеств специально для людей с тяжелыми травмами мозга. Вспомните еще раз, как я сказал, что резюме — это очень слабое средство оценки программиста, и из него можно получить только очень слабый сигнал. Некоторые технологии просто, скажем так, *тяжелее* других, и если вам удалось успешно с ними поработать, то ваши шансы быть принятым немного возрастают. Таким образом, при сортировке резюме трудные технологии, скорее всего, вынесут вас наверх, а заявление, что вы компетентны, скажем, в Microsoft Word, скорее всего, приведет вас куда-то вниз.

Разнообразие

Перед тем как я разожгу массированную войну международного масштаба с использованием многозначного термина "разнообразие", разрешите подробно объяснить, что я под ним подразумеваю. Лично мне нужны люди, у которых подготовка во многом другая, чем у тех, кто уже работает в моей команде. Это делается для того, чтобы новички принесли в команду новые идеи и способы мышления, противостояли зарождению группового мышления, которое может превратить наши размышления в сплошное поддакивание. Говоря о другой подготовке, я подразумеваю подготовку культурную, общественную и профессиональную. Принести полезное разнообразие в команду Интернет-программистов может тот, кто имеет большой опыт создания программ для предприятий, а в "стартап", забитый питомцами частых школ, — тот, кто вырос в бедности. Мамаша-домохозяйка, поступив на работу в офис, может принести полезное разнообразие в команду, состоящую из вчерашних выпускников, как и инженер-электрик с опытом рабо-

ты на языке ассемблера — в команду хакеров, работающих на языке Lisp. А в команду опытных бизнес-консультантов со Среднего Запада полезное разнообразие может быть принесено недавним выпускником колледжа из Эстонии. Единственный теоретический вывод отсюда — чем разнообразнее ваша команда, тем вероятнее, что кто-то из ее членов воспользуется подготовкой своих товарищей по команде, и в результате получится какое-то другое решение имеющейся проблемы.

Очень и очень важно не забывать, что эти категории (страсть, серьезность намерений, литературный язык, мозги, конкурсность, самое тяжелое и разнообразие) *не являются* критериями найма. Для этого они слишком слабые. Слишком много прекрасных людей наберут мало очков по этим критериям, и наоборот, много очков по ним наберет слишком большое количество плохих программистов. Впрочем, перед тем как разражаться гневными тирадами о том, чего это Джоэл думает, что надо набирать только выпускников колледжей из "Лиги плюща" или чего это он сделал из оценок за учебу своего рода фетиш, ну и так далее, и тому подобное, важно понять, что перечень таких критериев — это не список причин, по которым следует нанимать или отвергать кого-то. Все это объективный и справедливый способ сортировки большой стопки резюме, чтобы найти наиболее подходящих для вас кандидатов и с ними первыми провести собеседование, а *лишь затем* решить, следует ли их нанимать.

Если резюме такие малоэффективные, можно ли их подстраховать другими средствами?

Как ни напрягай воображение, все равно не найдешь идеальных правил сортировки резюме. Было бы намного лучше сортировать резюме по способности кандидата к реализации алгоритма рекурсии, по тому, насколько быстро он найдет ошибку в коде, способен ли кандидат сохранять пять объектов

в оперативной памяти. Это более надежные индикаторы успешности программиста, чем, например, вопрос, смогли ли вы успешно пройти приемную комиссию элитного колледжа. Но, к сожалению, о таком в резюме не сообщают.

Рекрутерам хочется использовать в процессе отбора несколько дополнительных средств. Я часто слышал, как предлагали устраивать что-то вроде викторины по программированию. Она, конечно же, помогает уменьшить количество кандидатов, но качество отбора фактически не увеличивает. Прекрасные программисты найдут достаточно мест, где вначале требуют только обычные резюме и сопроводительное письмо. А если использовать надуманные средства и экзамены по программированию или что-то еще попроще, то этим вы скорее всего вместе с плохими программистами отпугнете и хороших. Но вероятнее всего вы отпугнете именно лучших программистов, у которых больше всего вариантов, и останетесь с кучкой довольно отчаянных кандидатов, готовых к дополнительным усилиям просто потому, что у них нет иного выбора.

Не предпочитайте опыт работы с конкретными технологиями

Однажды я был в группе специалистов, дававшей в Нью-йоркском университете советы студентам, как делать карьеру в информатике. Я советовал, чтобы перед выпуском студент обязательно знал, как красиво записать свои мысли, для чего ему, скорее всего, не помешал бы курс изящной словесности, а также студенту надо прослушать курс экономики, чтобы деловая сторона его работы не была для него тайной[3]. Еще я рекомендовал прослушать хотя бы один курс по низкоуровневому программированию на языке C или ассемблера — просто

[3] Joel Spolsky, “Advice for Computer Science College Students”, опубликована на сайте `www.joelonsoftware.com`, 2 января 2005 года (поиск по фразе “College Advice”). Также в Сети можно найти и перевод — “Совет студентам, изучающим вычислительную технику”. — *Примеч. пер.*

для того, чтобы студент понимал, как компьютер работает на низком уровне.

В этой группе со мной находился местный рекрутер — фактически один из лучших в городе рекрутеров по техническим специальностям. Его пятнадцатиминутная речь была напичкана скучными сокращениями. “Нам нужно много XML, немного C++, востребованными становятся SOAP и WSDL, а вот COM и даже ATL среди вас явно недостаточно”. И так далее, и тому подобное, пока мои глаза не стали квадратными. Это был парень, мысли которого были целиком в мире ключевых слов из резюме.

Самых сильных программистов раздражает в рекрутерах больше всего их почти болезненное восхищение ключевыми словами и “умными” терминами. Целая отрасль рекрутинга и вербовки специалистов имеет странную привязанность к простому алгоритму соответствия кандидатов и вакансий. Он состоит в поиске кандидатов, в чьих резюме находится полный список технологических аббревиатур, которые могут заинтересовать работодателя. Особенно раздражает то, что большинство таких рекрутеров представления не имеют, *что* из себя представляют все эти технологии. “Как, не имеете опыта работы с MSMQ? Не беда”. Даже когда агенты по недвижимости болтают о холодильниках Sub-Zero® и духовках Viking, то они как минимум знают, *что* это за предметы (хотя сейчас любой холодильник из нержавеющей стали считается “Sub-Zero”). Обнаружить рекрутера по техническим специальностям легче всего тогда, когда он неизменно требует пяти лет опыта работы с Ruby on Rails или отказывается говорить о специальности “Windows API” с тем, кто указал в своем резюме только “Win32”.

Рекрутеры пользуются этим алгоритмом потому, что он легкий, его можно компьютеризировать, да и знают они только его.

Впрочем, это наихудший способ найма почти на все должности по разработке программ.

Наша философия состоит в том, что мы нанимаем на долгий срок, и любая технология, которую вы, возможно, сейчас знаете, может запросто устареть через год. Более того, некоторые из этих технологий очень легко изучить. Если бы мне потребовалось нанять кого-то для разработки с помощью Ruby,

то им, скорее был бы человек, имеющий широкий опыт работы со Smalltalk и Python, хотя и не слышавший о Ruby, чем тот, кто однажды прочитал об этом языке книгу. Для того, кто в целом хороший разработчик программ, не составит особого труда выучить другой язык программирования. Через две недели такие специалисты будут достаточно продуктивны. Через два года вы, возможно, дадите им делать что-то на таком языке программирования, который еще даже не создан.

Да и не слишком надо доверять разделу резюме с ключевыми словами: каждый работающий программист знает о программах, фильтрующих резюме по таким словам, поэтому в такой раздел своего резюме он обычно заносит название любой технологии, с которой когда-либо соприкасался, — исключительно для того, чтобы пройти сквозь фильтры.

Но, как мне кажется, из этого правила есть одно исключение. Нанимая программного архитектора или главного разработчика, т.е. ведущего инженера-программиста, который будет отвечать за начальное планирование кода и продумывать совместную работу компонентов, вы, вероятно, захотите нанять того, кто имеет большой опыт работы в используемой вами технологии. В команде, которая разрабатывает с помощью C++ и MFC графические пользовательские интерфейсы для Windows, требуется хотя бы один эксперт по Windows/MFC, который прежде всего сумеет проверить, правильно ли организован код, а также обладает достаточным опытом, чтобы при возникновении по-настоящему сложных проблем знать, как их решить.

Не начинайте нового проекта, не имея хотя бы одного программного архитектора, обладающего несколькими годами солидного опыта работы с языком, классами, API и теми платформами, для которых вы создаете свой продукт. Если можете выбрать платформу, используйте ту, в которой ваша команда имеет наибольшую квалификацию, даже если такая платформа не самая модная или официально не самая производительная. Это может означать, что время от времени вам надо будет проводить собеседование, чтобы найти кандидата, имеющего по-настоящему большой опыт работы с конкретным набором технологий (обратите внимание — технологий, а не их ключе-

вых слов, которых полным-полно, например, LAMP или .NET или J2EE). Зато большинство ваших разработчиков должны быть наняты не по ключевым словам.


THE PHONE
INTERVIEW
GAME

Глава 5

Телефонный заслон

Это происходит все время. Мы получаем резюме, о котором каждый скажет, что оно действительно замечательное. Потрясающие звания. Все виды впечатляюще звучащих должностей. Огромный опыт. Говорит на семнадцати языках. И спас больше 10 тысяч котят!

Надо же! Котят!

А затем я звоню таким людям, и происходит невыносимый для меня разговор. Через десять минут я убеждаюсь, что они не собираются работать программистами. Мне попадались люди с прекрасными резюме, утверждавшие, что указатель должен помещаться в одном байте. Иногда они не могут ответить на простейшие вопросы, или вы чувствуете, что как будто сами должны вести борьбу с их вопросами.

Перед тем как переходить к полномасштабному персональному собеседованию, мы обычно используем телефонный заслон, чтобы не терять время и деньги на того, кто, если серьезно, не выглядит способным.

По сравнению с обычным персональным собеседованием телефонный заслон имеет явные преимущества. Во-первых, он дешевый. На него уходит от 45 минут до часа, и он фактически сокращает наполовину количество тех кандидатов, которые действительно прекрасно выглядели на бумаге.

Во-вторых, еще важнее то, что он более справедливый.

Благодаря телефонному собеседованию (ведь вы не видите человека) легче сосредоточиться на качестве того, что он говорит, а не на внешних факторах, не относящихся к его работе, к примеру, на внешнем виде или на нервозности. Даже когда Малкольм Гладуэлл написал книгу *Озарение: сила мгновенных решений*, меня все равно ужасала перспектива, что мы, возможно, слишком быстро оцениваем кандидатов на основе того, что

не относится к их способности делать свою работу[1]. Их внешний вид или уверенность в себе либо рост или вся манера напускать на себя вид знатока иногда заставляют нас видеть через розовые очки все остальное, что происходит на собеседовании.

В телефонном собеседовании прекрасно то, что намного труднее создать такого рода скоропалительное мнение; вам по-настоящему приходится слушать, что говорит собеседник, и решать, соответствует ли это тому, что может сказать способный человек. Конечно, это еще не вся правда: у вас может быть предвзятое мнение о некоторых акцентах или диалектах. Но вы хоть будете менее восприимчивы к предрассудкам, связанным с внешним видом.

Мои телефонные собеседования состоят из трех этапов. На первом из них я прошу кандидата описать его карьеру и коротко рассказать о себе. В основном это нужно для того, чтобы он расслабился и почувствовал себя свободно, чтобы снять всякую нервозность и дать кандидату возможность представить себя так, как ему хочется быть представленным. На данном этапе вам надо искать признаки того, что кандидат относится к числу людей, решающих проблемы, т.е. к тем людям, которые все доводят до конца. Надо также искать проявление страсти. Вам нужны люди, которые заботятся о том, что делают.

На этом этапе я подробно задаю вопросу, касающиеся двух тем — технологии и политики.

Технология. Если кто-то мне говорит, что реализовал такой-то и такой-то проект, я задам подробные вопросы об использованной при этом технологии и о том, как она использовалась. Я также специально спрошу, какова была роль кандидата. Разработчик-одиночка? Разработчик из команды? Я стараюсь подробнее останавливаться на таких вопросах, ведь благодаря этому вы распознаете людей, которые или не разбираются в том, что делали, или сочиняют либо преувеличивают собственную роль. Если, например, в чьем-то резюме подразумевается, что кандидат два года программировал на

[1] Malcolm Gladwell, *Blink: The Power of Thinking Without Thinking* (New York: Little, Brown and Company, 2005); Малкольм Гладуэлл, *Озарение: сила мгновенных решений*. — ООО ИД "Вильямс", 2006.

языке Python, то я обычно задаю подробные вопросы, пока не смогу достаточно хорошо убедиться, что у него вроде как точно двухлетний опыт работы с этим языком.

Политика. За имеющимся в резюме скучным перечнем прошлых работ всегда скрывается какая-то история. Лично я ищу историю о том, как кандидат справлялся в прошлом с трудностями. Мне нужны люди, которые все доводят до конца, даже если им противодействуют. Мне нужны люди, которые бросают вызов существующему положению, одолевают препятствия и все доводят до конца. Например, когда в резюме написано "управлял принятием .NET," то мне надо знать, что означает это "управление". В деталях. Когда в резюме говорится, что кандидат "основал компанию," то я хочу услышать все. Чья была идея? Кто кого убеждал? Кто что делал? Удалась ли работа? Почему не удалась?

Еще одна часть телефонного заслона — это техническая проблема. Один и тот же вопрос я обычно задаю годами, пока не заменю его другим: так легче сравнивать кандидатов. Этот объемный и нерешенный вопрос, относящийся к разработке, звучит примерно так: как спроектировать структуру данных или блок кода, чтобы можно было выполнять *x*? Буквой *x* обозначено нечто большое и сложное. Обычно у меня подготовлен ряд дополнительных вопросов, чтобы вести кандидата по пути проектирования структуры данных или блока кода, ведь основной вопрос довольно большой, и по тому, насколько продвинулся кандидат по этому пути за установленное время, я часто я могу сказать, насколько он умен.

Вот с каких идей можно начать.

- Как спроектировать программу, с помощью которой люди могут играть друг с другом в "монополию" через Интернет[2]?
- Какой должна быть хорошая структура данных для программы редактирования фотографий?

---

[2] Reg Braithwaite, "My favourite interview question", `weblog.raganwald.com/2006/06/my-favourite-interview-question.html`, 12 июня 2006 года.

- Как реализовать код для управления лифтами, предназначенными для подъема на большую высоту?
- Как реализовать машину визуализации Web-браузера?

Идеальный вопрос касается того, что кандидат очень хорошо знает, но вряд ли когда-то реализовывал самостоятельно. Вам нужно то, что можно сделать по телефону, без большой писанины. Поэтому вопрос: "Как вы написали бы код быстрой сортировки?" — не годится, ведь мы не надеемся, что программисты способны декламировать код по телефону. Вам захочется говорить об алгоритмах и структурах данных, которые фактически являются сутью программирования. Цель разговора не в том, чтобы обязательно найти как можно лучший ответ, а просто дать вам поговорить о коде, о компромиссах между временем и пространством, о характеристиках производительности кода. Все это даст вам очень хорошее представление, действительно ли ваш собеседник прекрасно разбирается в программировании, способный он или нет. Если вы заметите, что объясняете все по три раза, тогда или у вас очень плохо с умением объяснять или вы разговариваете с тем, кто не очень умен.

Суть моей техники собеседования в том, что способные люди, разговаривая с кем-то на трудную или сугубо специальную тему, обычно видят, способный их собеседник или нет. Ну а вопрос в собеседовании — это на самом деле только повод начать разговор на трудную тему, лишь бы у того, кто проводит собеседование, хватило здравого смысла определить, способный кандидат или нет.

Третья и заключительная часть собеседования состоит в том, чтобы кандидат вел собеседование со мной.

К этому времени мне уже ясно, достаточно ли умен кандидат, чтобы перехватить инициативу в персональном собеседовании. Итак, для меня третья часть телефонного интервью — это как бы собеседование, проводимое со мной, когда кандидат должен проявить себя перед *Fog Creek*. Что он и делает. Не забывайте — вся философия найма основана на том, что способные кандидаты имеют выбор, где работать, и если это так, то процесс собеседования во многом служит способом, позволяющим кандидату решить, хочет ли он работать на нас, а нам —

решить, хотим ли мы нанять этого кандидата. “Вопросы есть по нашей компании, по работе в ней или вы хотите спросить что-то другое?”

Иногда именно эта часть собеседования открывает угрожающий недостаток подготовки у кандидата. “Что именно делает *Fog Creek*? Кроме того, где она находится?” Если кандидат не может перед собеседованием выполнить даже простейшую домашнюю работу, проведя пять минут на нашем Web-сайте, то я сильно сомневаюсь в том, что он способный или доводит дело до конца.

Преодоление телефонного заслона никогда не означало, что кандидат уже нанят. Это всего лишь простой заслон, позволяющий экономить время и деньги, которые иначе пришлось бы тратить на персональные собеседования, а также позволяющий “отсечь” кандидатов, которые никогда такие собеседования не пройдут, причем сделать это еще до того, как вы их пригласите лететь через всю страну и поселите в изысканном отеле. Впрочем, даже среди кандидатов, отобранных телефонным собеседованием, персональное собеседование пройдет, вероятно, где-то одна треть.


$x^n+y^n=z^n$
$n>2$
$x,y,z,\neq 0$

Глава 6

Базовое руководство по собеседованиям

Разношерстная банда, состоящая из анархистов, сторонников свободной любви и агитаторов за права клоунов, захватила на выходе из Пуэрто-Вальярта круизное судно "Корабль любви" и угрожала затопить его через семь дней вместе со всеми 616 пассажирами и 327 членами экипажа, если их требования не будут удовлетворены. И в чем же состояли эти требования? Миллион долларов, разбросанный небольшими порциями по незаметным банковским счетам, а также бесплатную реализацию WATFIV, т.е. ожидаемый в скором времени компилятор языка Waterloo Fortran IV. (Удивительно, как мало может объединить сторонников свободной любви и борцов за права клоунов.)

Вам как главному руководителю команды программистов фирмы *Festival Cruise* надо решить, сможете ли вы буквально из ничего создать этот компилятор за семь дней. Вам в помощь дают двух программистов.

Так справитесь ли вы с этим делом?

— Думаю, что да, но при определенных условиях, — говорите вы.

Одно из преимуществ от написания этой книги — я могу вас заставить говорить что угодно, и ничего вы тут не сделаете.

Так при каких же условиях?

— Ну, будет ли у моей команды возможность пользоваться инструментами UML-генерации?

Это так важно? Три программиста, за семь дней, Waterloo Fortran IV. Разве успех или неудача в создании компилятора зависит от UML-инструментов?

— Думаю, что нет.

— Ладно, отчего же тогда зависит?

— У нас будут 19-дюймовые мониторы? И предоставят вам столько кофе, сколько захотим?

И опять, это так важно? Разве успех дела зависит от кофеина?

— Думаю, что нет. Ой, минуточку. Вы сказали, что мне в помощь дают двух программистов?

Именно так.

— Кто они?

Это так важно?

— Конечно! Если мы не поладим друг с другом, то никогда не сможем работать вместе. Кроме того, я знаю совсем мало суперзвезд программирования, которые *сами* "склепают" Fortran-компилятор за неделю, и *очень много* программистов, не способных меньше, чем за шесть месяцев, написать код даже баннера стартап-компании.

Вот это уже что-то!

На словах каждый согласен с тем, что самая важная часть программного проекта — это люди, но никто с уверенностью не скажет, что же вам надо *сделать*. Так вот, самое первое, что надо сделать, если требуются хорошие программисты, — это *нанять* нужных программистов. Следовательно, вам придется определять, кто *будет* нужным программистом, и такое обычно происходит на собеседованиях. Итак, эта глава целиком посвящена собеседованиям.

Всегда старайтесь, чтобы собеседования с каждым кандидатом проводили как минимум шесть человек, не меньше пяти из которых — это его возможные товарищи по работе (т.е. другие программисты, а не начальники). Известны ли вам компании, где собеседование с каждым кандидатом ведет только бывалый старый начальник, от решения которого все и зависит? Так вот, в них работают не очень хорошие специалисты. Очень легко впасть в заблуждение из-за одного собеседования, особенно тогда, когда с программистом беседует непрограммист.

Если даже двое из шести сотрудников, побеседовав с кандидатом, считают, что его не следует нанимать, то так и поступайте. Иными словами, "день собеседований" закончится после первых двух, если кандидата сочтут неподходящим для компании. Это неплохая идея, но, чтобы не казаться жестоким,

вы, возможно, не захотите говорить кандидату заранее, сколько людей будет с ним беседовать. Я слышал о компаниях, где кандидата отвергают, если он не понравится хотя бы одному сотруднику, проводившему с ним собеседование. Мне это кажется слишком жестоким. У меня такое право имеют разве что старшие сотрудники. Ну а чтобы отвергать кандидата лишь потому, что он не понравился одному из младших подчиненных, — такого я не допускаю.

Не пытайтесь проводить собеседование сразу с целой группой людей. Это просто нечестно. На каждом собеседовании должны быть только тот, кто беседует, и тот, с кем беседуют, а проводиться оно должно в комнате за закрытой дверью и с планшетом для рисования.

Кроме того, на основе своего обширного опыта должен вам сказать, что, потратив на собеседование меньше часа, вы не примете правильного решения.

Проводя собеседования, вы столкнетесь с тремя типами кандидатов. С одной стороны спектра находится темная масса тех, кто не знает даже самого основного. Чтобы их распознать и отсеять, обычно достаточно двух-трех легких вопросов. Другая сторона спектра занята блестящими суперзвездами, которые могут за выходные потехи ради написать на языке ассемблера для Nintendo DS компилятор языка Lisp. Ну а посредине расположено большое количество тех, кто “внушает надежды” и просто кажется способным дать хоть какую-то пользу. Фокус в том, чтобы отделить от них суперзвезд, ведь, говоря по секрету, вам никогда не хочется нанимать кого-то, кто просто “внушает надежды”.

В конце собеседования вы должны быть готовы принять четкое решение о кандидате. У этого решения может быть только два варианта — *Нанимать* или *Не нанимать*. Других ответов здесь нет. *Никогда* не говорите: “Нанимать, но не в мою команду”. Это грубо и подразумевает, что кандидат недостаточно разумен, чтобы работать с вами, но, возможно, вполне умен для аутсайдера, руководящего “не вашей” командой. Если возникнет искушение сказать “Нанимать, но не в мою команду”, то всего лишь механически переведите эту фразу в “Не нанимать”, и вы поступите правильно. Даже если кан-

дидат прекрасно выполняет именно ваше задание, но не может быть очень хорошим в другой команде, то это означает *Не нанимать*. В программировании изменения происходят так часто и так быстро, что вам требуются люди, которые могут успешно выполнить почти любое ваше задание по программированию. Если по какой-то причине вам встретится ученый идиот, который прямо-таки очень хорошо разбирается в SQL, но полностью неспособен даже изучить что-то другое, то ни в коем случае его *Не нанимать*. Решение кратковременной проблемы обернется для вас намного более масштабной хронической проблемой.

Никогда не говорите: "Ну, тут я не могу ничего сказать". Если не можете сказать, то это значит *Не нанимать*. На самом деле это легче, чем вы думаете. Не можете сказать? Тогда просто ничего не говорите! Если вы соблюдаете нейтралитет, такой "ответ" означает *Не нанимать*. Никогда не говорите: "Ладно, нанимать, но меня немного беспокоит, что...". Это также означает *Не нанимать*. Механически переведите всю эту болтовню в "нет" и будете полностью правы.

Почему здесь я такой упрямый? Потому, что отвергнуть хорошего кандидата *намного* лучше, чем принять плохого. Плохой кандидат будет стоить компании немало денег и усилий, а также времени, потраченного другими людьми на исправление всех его ошибок. Увольнение человека, нанятого вами по ошибке, может занять месяцы и оказаться кошмарно трудным, особенно если он не захочет уходить. Иногда уволить кого-то бывает невозможно. А ведь плохие сотрудники развращают хороших. Еще они могут оказать плохими программистами, но по-настоящему *хорошими людьми* или, возможно, *эта работа им действительно нужна*, поэтому вы не сможете взять на себя их увольнение или не сможете их уволить, не расстроив всех остальных либо не вызвав еще каких-то последствий. Получится действительно неприятная ситуация.

А с другой стороны, если вы отвергнете хорошего кандидата, тогда, как я подразумеваю, как я *догадываюсь* в некоем экзистенциальном смысле, случится несправедливость. Но если он такой способный, тогда не беспокойтесь: этому кандидату еще предложат *массу* хорошей работы. И не бойтесь, что вы

отвергнете слишком много людей и не сможете кого-то нанять. Во время собеседования это не должно вас беспокоить. Конечно, немаловажно найти хороших кандидатов. Но когда вы лично проводите собеседование, сделайте вид, что за дверью ждет еще как минимум 900 человек. Чтобы найти прекрасных кандидатов, не занижайте свои требования, какими бы тяжелыми они ни казались.

Да, я не сказал самого главного — как узнать, наймете ли вы кого-то?

В принципе это просто. Ищите людей, которые

- способны и
- доводят дело до конца.

Это (и только это) как раз то, что вам нужно. Помните об этом. Повторяйте это про себя каждый вечер, пока отходите ко сну. Во время короткого собеседования у вас не будет достаточно времени, чтобы узнать намного больше, поэтому не теряйте времени, пытаясь понять, будет ли вам с кандидатом приятно, если вы застрянете с ним в аэропорту, точно ли он знает программирование с помощью ATL и COM либо только притворяется.

Люди, которые *способны*, но не *доводят дела до конца*, часто имеют ученую степень и работают в больших компаниях, где их никто не слушает, потому что они совсем непрактичны. Вместо того чтобы решить проблему вовремя, они будут продумывать какой-то ее научный аспект. Этих людей нетрудно узнать: они любят подчеркивать теоретическое сходство двух очень разных понятий. Например, они скажут: “Электронные таблицы в сущности не что иное, как частный случай языка программирования”, — а затем исчезнут на недельку и напишут захватывающую, великолепную статью о теоретических вычислительных лингвистических атрибутах электронной таблицы как о языке программирования. Разумно, но бесполезно. Есть еще один способ узнать таких людей: обычно *в день, когда вы пытаетесь закончить бета-версию программы*, они появляются в вашем офисе с кружкой кофе в руках и пытаются начать долгий разговор о том, что же лучше — библиотеки COM-типов или самоанализ языка Java.

А люди, что *доводят дело до конца*, но не являются *способными*, будут делать глупости, причем явно бездумно, и потом кому-то другому придется за ними все вычищать. Это делает их общим *бременем* компании, ведь мало того, что у них не получается принести ей пользу, но они еще и забирают время у хороших специалистов. К такой разновидности людей относятся те, кто решает устроить рефакторинг вашим основным алгоритмам, чтобы использовать Visitor-шаблон, о котором они прочитали накануне вечером и ничего в нем не поняли. После чего вместо простых циклов, складывающих элементы массива, вы получаете класс `AdderVistior` (да, произносится неправильно), а также синглтон `VisitationArrangingOfficer`, и ни одна из частей вашего кода больше не работает.

Как во время собеседования обнаружить *способного* кандидата? Здесь первым хорошим признаком служит то, что вам не нужно что-либо повторно объяснять. Просто идет разговор. Часто кандидат говорит то, что показывает настоящую проницательность либо интеллект или острый ум. Поэтому важной частью собеседование является создание ситуации, в которой кто-то может показать, насколько он способен. Хуже всех проводят собеседования хвастуны. Они все время болтают, и у кандидата не остается времени что-либо сказать, кроме как "Да, *как* это верно. *Полностью с вами согласен*". Такие люди наймут каждого; они думают, что кандидат должен быть способный, ведь он "мыслит во многом так же, как и я!"

Второй наихудший тип ведущего собеседования — это ведущий телевикторины. Такой человек думает, что "способный" — это "тот, кто знает много фактов". Проводя собеседование, он всего лишь задает кучу тривиальных вопросов о программировании и дает зацепки для правильных ответов. Просто смеха ради приведу самый худший вопрос, заданный на собеседовании: "Какая разница в Oracle 8i между `varchar` и `varchar2`?" Надо же, какой ужасный вопрос. Невозможно представить, чтобы была связь между теми, кто знает определенный кусок тривиальных фактов, и теми, кого вы хотите нанять. Кому-то интересно, какая здесь разница? Ответ можно найти в Интернете примерно за 15 секунд! Итак, не забывайте, что "способный" *не означает* "тот, кто знает

ответы на тривиальные вопросы". Как бы то ни было, в команды разработчиков программ требуются люди *одаренные*, а не обладающие конкретным набором знаний. Ведь любой набор знаний, который люди могут с собой принести, за пару лет технологически устареет. Поэтому лучше нанимать тех, кто способен изучить любую новую технологию, а не тех, кто может *прямо сию минуту* сказать, как организовать общение JDBC с базой данных MySQL.

Но вообще-то больше всего вы узнаете о кандидате, когда дадите ему говорить. Задавайте этим людям вопросы, допускающие разные толкования, и ставьте перед ними проблемы.

Итак, о чем их спрашивать?

Мой личный список вопросов, задаваемых на собеседовании, берет начало с моей первой работы; ее я получил в *Microsoft*. Имеются сотни действительно знаменитых вопросов, задаваемых на собеседованиях в этой компании[1]. У каждого есть свой набор по-настоящему любимых вопросов. Вы также разработаете определенный набор вопросов и свой стиль ведения собеседования, которые помогут вам сделать выбор *Нанимать–Не нанимать*. Вот некоторые из моих приемов, которые оказались успешными.

Перед собеседованием я читаю резюме кандидата и вкратце пишу на листе план собеседования. Это просто список вопросов, которые я хочу задать. Вот обычный план собеседования с программистом.

1. Вводная часть.

2. Вопрос о недавнем проекте, в котором работал кандидат.

3. Легкие вопросы по программированию.

4. Вопрос об указателе или рекурсии.

5. Вы довольны?

6. Есть какие-то вопросы?

---

[1] Вот их целая книга: William Poundstone, *How Would You Move Mount Fuji: Microsoft's Cult of the Puzzle* (New York: Little, Brown and Company, 2003); Уильям Паундстоун, *Как сдвинуть гору Фудзи? Подходы ведущих мировых компаний к поиску талантов.* — Альпина Бизнес Букс, 2004.

Я тщательно стараюсь избегать всего, что может породить у меня предвзятое мнение о кандидате. Если вы считаете кого-то способным еще до того, как он зайдет в комнату и всего лишь из-за его ученой степени, полученной в Массачусетсском технологическом институте, тогда ничего из сказанного им в течение часа не поколеблет ваше первоначальное предубеждение. И наоборот, если вы считаете кандидата провинциалом из-за того, что он ходил в окружной колледж, то ничего из сказанного им не преодолеет это первоначальное впечатление. Собеседование похоже на очень нежные весы, и слишком трудно о ком-то судить на основе часового собеседования, которое может выглядеть как очень рискованное предприятие. Если вы заранее немного узнаете о кандидате, то это будет как большой груз на одной из чашек весов, и тогда собеседование бесполезно. Однажды, прямо перед собеседованием, ко мне в офис пришла женщина рекрутер. "Вы *полюбите* этого парня", — сказала она. *Мальчик* же, наоборот, позже сделал все, чтобы вывести меня из себя. На эти ее слова я должен был ответить: "Ну, если вы так уверены, что я его полюблю, почему бы вам просто не нанять его, чтобы мне не терять время на собеседование". Но я был молодой и наивный, поэтому провел с ним собеседование. Когда я услышал от него не совсем разумные вещи, то подумал: "Вот это да, но должно ведь быть исключение, подтверждающее правило". На все, что он говорил, я смотрел сквозь розовые очки. В конце концов я сказал *Нанимать*, хотя кандидат он был дрянной. И знаете что? Все остальные, проводившие с ним собеседование, сказали *Не нанимать*. Итак, не слушайте рекрутеров; не спрашивайте о человеке до собеседования с ним; никогда не говорите о кандидате с другим сотрудником, проводящим собеседование, пока вы оба самостоятельно не сделаете свои выводы. Вот это и есть научный метод.

Вводная часть собеседования предназначена для того, чтобы дать кандидату расслабиться. Я его спрашиваю, хорошо ли он долетел. Затем около 30 секунд говорю, кто я такой и как будет проходить собеседование. Я всегда убеждаю кандидатов, что нас интересует не сам ответ, а *как* они решают задачи.

Часть вторая — это вопрос об одном из недавних проектов, над которыми работал кандидат. Проводя собеседование с вы-

пускниками колледжей, спросите о дипломной работе, если они ее уже защитили, или о том учебном курсе, который им действительно понравился. К примеру, иногда я спрашиваю: "Какой из учебных курсов, пройденных вами на последнем семестре, понравился вам больше всего? Он может не обязательно относиться к компьютерам". А проводя собеседование с опытными кандидатами, можете поговорить о самом последнем задании, полученном ими на предыдущей работе.

И опять же, задавайте вопросы, допускающие разные толкования, после чего сидите и слушайте, только время от времени говоря: "Расскажите об этом поподробнее", — на тот случай, если кажется, что кандидат вот-вот застопорится.

Что надо искать с помощью вопросов, допускающих различные толкования?

Во-первых, страсть. Способные люди страстно относятся к проектам, в которых участвуют. Говоря на эту тему, они очень волнуются. Их речь становится быстрой, а сами они — оживленными. Даже страсть *в отрицательном смысле* может быть не более чем хорошим знаком. "Мой предыдущий начальник хотел, чтобы все делалось на компьютерах VAX, ведь только в них он и разбирался. Какой болван!" Вокруг слишком много людей, хотя и способных на чем-то работать, но которым, по сути, все равно, на чем именно. Трудно найти людей, таких как этот, чтобы они были так заинтересованы в чем-то.

Плохие кандидаты просто ни о чем не беспокоятся и на собеседовании вообще не проявляют энтузиазма. Если говоря о чем-то, кандидат на некоторое время забывает, что находится на собеседовании, то это действительно хороший признак его страстности. Иногда явившийся кандидат очень нервничает из-за того, что должен проходить собеседование. Такое состояние, конечно же, нормально, и я никогда не обращаю на это внимание. Если вы начнете говорить с ним о Вычислительном монохроматическом искусстве, он сильно оживится, и признаки нервозности пройдут. Хорошо. Мне нравятся страстные люди, которым не все равно. (Образец Вычислительного монохроматического искусства можно увидеть, если отсоединить монитор от компьютера.) Этому человеку иногда можно "бросить вызов" (например, дождаться от него чего-то

явно правдивого и сказать: “Такого быть не может”). И даже если пять минут назад он очень волновался, то все равно начнет защищаться, так как неравнодушен до такой степени, что даже забудет о том, что вы собираетесь вскоре принять Главное решение в его жизни.

Во-вторых, хорошие кандидаты, на каком бы уровне сложности ни беседовали, стараются все хорошо объяснять. Я часто отвергал кандидатов из-за того, что, говоря о своем предыдущем проекте, они не могли рассказать о нем так, чтобы их понимал обычный человек. Часто те, кто специализировался в компьютерной науке, просто считают, что все должны знать теорему Бейтса или что означает выражение $O(\log n)$. И если они начинают говорить подобным образом, остановите их на минутку и скажите: “Сделайте одолжение — хотя бы ради эксперимента говорите, пожалуйста, так, чтобы это могла понимать моя бабушка”. Тем не менее многие люди все равно продолжают использовать жаргон и совсем не могут понятно выражаться. *Отбой!* Их, по сути, нанимать не следует: они недостаточно способны, чтобы заставить других людей понимать свои идеи.

В-третьих, если проект, о котором рассказывают кандидаты, был коллективным, старайтесь узнать, брали ли они на себя руководство. Кандидат может сказать: “Мы работали над X, но начальник сказал Y, а клиент сказал Z”. Тут я спрашиваю: “И что же сделали *вы*?” Подходящий ответ на это может быть таким: “Я встретился с другими людьми, занятыми в проекте, и написал предложение...”. А вот один из неподходящих ответов: “Ну, сделать я ничего *не мог*. Такая была ситуация”. Помните: *Будь способным* и *Доводи дело до конца*. Единственный способ узнать, *доводит* ли кандидат *дело до конца*, — выяснить, было ли это характерно для него в прошлом. На самом деле можете даже прямо попросить, чтобы кандидат привел пример из недавнего прошлого, когда он взял на себя руководство и что-то довел до конца — например, преодолел установившуюся инерцию.

Впрочем, время собеседования по большей части надо потратить на то, чтобы кандидат мог доказать — он может писать код.

Заверьте кандидата, что вам понятно, как тяжело писать код без редактора, и вы будете снисходительны, если на планшете появится что-то запутанное. Кроме того, вам понятно,

как тяжело, не имея компилятора, писать код без ошибок, и вы это учтете.

Первое собеседование я начинаю по-настоящему легкой задачкой по программированию. Это я начал делать еще во время бума "дот-ком", когда на собеседования стало приходить много людей, думавших, что работа с HTML — это "программирование", и мне нужен был способ, позволяющий не тратить на них слишком много времени. Это такая задачка, которую любой сегодняшний программист должен решить где-то за минуту, например такие.

1. Напишите функцию, которая определяет, начинается ли строка с буквы A–Z в верхнем регистре.
2. Напишите функцию, которая определяет площадь круга с данным радиусом.
3. Сложите все члены массива.

Эти "блиц-вопросы" по программированию кажутся очень легкими, поэтому когда я впервые начал их задавать, то, надо признаться, считал, что с ними справится каждый. И вот что я увидел: задачка *поддавалась* каждому, но если посмотреть, *за какое время*, то здесь появлялось много вариантов.

Это напомнило мне, почему я не могу зарабатывать на жизнь, торгуя облигациями.

Дело в том, что мой партнер Джейрд торгует облигациями и всегда рассказывает мне о заключенных им интересных сделках. Есть такая вещь, как опцион, а также опционы на продажу, на покупку и рыночные тенденции. Короче, все это запутанно, но *мне известно, что все эти слова означают*, и точно известен опцион на продажу (право, но не обязанность что-то продать за определенную цену). Только за три минуты я могу вычислить, что же произойдет, если я приобрету опцион на продажу, а рынок пойдет вверх. Но на вычисление этого мне требуется *целых* три минуты, а когда мой партнер рассказывает что-то более сложное, где опционы на продажу — это только самая простая деталь в его рассказе, а там есть еще многое другое, я очень быстро "теряю нить", потому что теряю мысль (Видишь ли, рынок идет вверх, а это значит, что проценты идут вниз, и сейчас опцион на продажу — это право что-

то продать...). “Нить” будет потеряна до тех пор, пока он не возьмет бумагу с графиком и не начнет вести меня по нему, а мои глаза не станут тусклыми, и это очень печально. Даже поняв все самые мелкие детали, я не сделаю это *настолько быстро*, чтобы увидеть общую картину происходящего.

То же самое относится и к программированию. Если основные понятия для вас не настолько легкие, чтобы о них можно было и не думать, то сложные понятия вы не поймете.

Сердж Лэнг, профессор математики в Йеле, в первый же день занятий по математическому анализу давал студентам довольно простую задачку по алгебре, которую могли решить почти все, но некоторые ее решали *уже тогда, когда записывали*, а другим на решение требовалось довольно много времени. И тогда профессор Лэнг заявлял, что тот, кто решил задачу уже тогда, когда записывал, получит по анализу отличную оценку, а все остальные — не получат. *Скорость*, с какой студенты решали простую математическую задачку, была таким средством предсказания конечной оценки по анализу, как и целый семестр домашних работ, экзамены (как промежуточные, так и итоговый)[2].

Видите ли, если вы не можете мчаться в *легких* условиях даже со скоростью 160 км/час, то никогда не преодолеете сложные условия[3].

Но, как я уже сказал, хорошие программисты встают, пишут на планшете ответ, иногда добавляя остроумную реплику (Оп-ля! Unicode-совместимую! Прекрасно!), и все занимает 30 секунд. Теперь мне надо решить, действительно ли они хоро-

---

[2] Личное интервью Гэри Корнуэлла за 27 октября 2006 года. Типичная задача состоит в том, чтобы упростить выражение $\frac{\frac{1}{x+h}-\frac{1}{x}}{h}$.

[3] Или, как сказал Альберт Норт Уайтхед: “Есть глубоко ошибочный трюизм, повторяемый всеми учебниками и выдающимися людьми в своих речах, — нам надо вырабатывать в себе привычку думать о том, что мы делаем. Если быть точным, то верно как раз обратное. Цивилизация добивается прогресса благодаря увеличению количества важных операций, которые можно выполнять, не думая о них”. — Из *Введения в математику* (СПб., 1915).

шие, поэтому я задействую тяжелую артиллерию — рекурсию и указатели.

Пятнадцатилетний опыт собеседований убедил меня в том, что все лучшие программисты способны легко разбираться сразу с несколькими уровнями абстракции. По отношению к программированию это означает, что у них нет проблем с рекурсией (при обдумывании которой, среди прочего, надо держать в уме сразу несколько уровней стека вызовов) или со сложными алгоритмами, использующими указатель (в которых адрес объекта является чем-то вроде абстрактного представления самого объекта).

Я пришел к выводу, что понимание указателей языка C — это не квалификация, а способность. Когда начинается первый год изучения компьютерной науки, в начале семестра набирается около 200 детей, и все они, когда им было по четыре года, уже написали на языке BASIC для своих ПК сложные программы с приключениями. Они уже достаточно изучат в колледже язык C или Pascal, пока однажды профессор не познакомит их с указателями, и вдруг студентам становится ясно, что *указатели им непонятны*. Они просто ничего не понимают. 90% студентов уходят и специализируются на политических науках, а своим друзьям говорят, что среди студентов-компьютерщиков было мало хорошеньких представителей соответствующего пола, потому они и ушли. *По некоторой причине большинство людей, скорее всего, рождаются без той части мозга, которая понимает указатели.* Указатели требуют сложной формы двойного абстрактного мышления, которая некоторым людям просто недоступна, а она очень важна для хорошего программирования. Никогда не занимались изучением указателей многие “скрипт-жокеи”, которые начали программировать, копируя JavaScript-фрагменты в свои Web-страницы, а затем стали изучать Perl, и они никогда не сумеют создать код нужного вам качества.

Отсюда и берут начало все задаваемые на собеседованиях знаменитые вопросы, о которых вы слышали, например, о том, как “перестроить связанный список в обратном порядке” или “найти петли в древовидной структуре”.

К сожалению, несмотря на то, что, как мне кажется, все хорошие программисты должны работать с рекурсией и указателями, и это прекрасный способ узнать, является ли кто-то хорошим программистом, но правда заключается в том, что сейчас языки программирования почти полностью сделали это особое искусство излишним. Если десять лет назад было редкостью, когда студент-компьютерщик заканчивал колледж, не изучив на одном курсе рекурсию и функциональное программирование, а на следующем — C или Pascal со структурами данных, то сейчас вполне возможно, чтобы в уважаемых во всех других отношениях учебных заведениях проходили только язык Java[4].

Многие программисты, с кем вы проводили бы сейчас собеседования, склонны считать рекурсию, указатели и даже структуры данных глупыми деталями реализации, скрытыми во многих сегодняшних удачных языках программирования. "А вам когда последний раз пришлось писать алгоритм сортировки?" — хихикают они.

Тем не менее я, по правде говоря, не переживаю. Мне нужно, чтобы вызванная ко мне врач скорой помощи знала анатомию, даже если все, что она должна делать, — это поместить мне на грудь электроды компьютеризованного дефибриллятора и нажать большую красную кнопку. А еще мне нужны программисты, знающие программирование вплоть до уровня ЦПУ, даже если Ruby on Rails, после того как вы сделаете три щелчка мышью, *действительно* прочитает ваши мысли и полностью построит сетевой Web 2.0-сайт общественной организации.

Хотя с внешней стороны формат собеседования состоит в том, чтобы кандидат просто написал на планшете некий код, но моя настоящая цель состоит в том, чтобы поговорить об этом коде: "Почему вы сделали именно так?"; "Какова производительность вашего алгоритма?"; "Что вы забыли?"; "Где здесь ошибка?"

Это значит, что я в действительности не против, чтобы сообщить о слишком серьезных недостатках программирования,

---

[4] Joel Spolsky, "The Perils of JavaSchools", опубликованная на `www.joelonsoftware.com`, 29 декабря 2005 года (поиск по "JavaSchools").

а также дать кандидату шанс начать самому, а затем я с удовольствием по ходу дела даю небольшие подсказки — скажем так, небольшие точки опоры. Я могу попросить кого-то, скажем, спроецировать треугольник на плоскость, что является типичной графической задачей, и я не против того, чтобы помочь с тригонометрией, и когда я спрошу, как ускорить это, то могу бросать маленькие намеки о таблицах просмотра. Обратите внимание, что те подсказки, которые я рад предоставить, — это на самом деле просто ответы на тривиальные вопросы, т.е. нечто такое, что можно найти в Google.

Неизбежно в функции, написанной кандидатом, вы найдете ошибку. Таким образом мы приходим к вопросу номер пять из моего плана собеседования: "Вы довольны своим кодом?" Вам, возможно, захочется спросить: "Ладно, а где же ошибка?" Вот он самый основной вопрос из преисподней, который допускает разные толкования. Все программисты делают ошибки, и в этом нет ничего плохого, просто они должны иметь способность их находить. Что касается строковых функций из C, то большинство деток из колледжа забывают завершить новую строку символом null. Почти в любой их функции наверняка попадаются менее значительные ошибки. Они иногда забывают поставить точку с запятой. Их функция не будет работать правильно на строках нулевой длины или будет выдавать General protection fault ("Общий сбой защиты"), если `malloc` завершится аварийно... Очень и очень редко попадается кандидат, в коде которого с первого раза не остается ни одной ошибки. Тогда этот вопрос выглядит еще забавнее. Если вы скажете: "В этом коде ошибка", — он внимательно просмотрит свой код, и у вас будет возможность узнать, строг ли он в рассуждениях по поводу своего кода, а также проявляет ли он хоть и дипломатическую, но все же твердость, утверждая, что код безупречен.

В качестве последнего этапа собеседования спросите кандидата, есть ли у него какие-то вопросы. Не забывайте, что хотя собеседование проводите вы, но хорошие кандидаты имеют большой выбор мест работы, и они используют этот день, чтобы понять, хотят ли они у вас работать.

Кто-то из тех, кто проводит собеседования, стараются определить, задает ли кандидат "интеллектуальные" вопросы.

Лично меня не волнует, какие вопросы задают кандидаты, ведь я уже принял свое решение. Проблема в том, что кандидату в течение дня придется встретиться с пятью-шестью людьми, и для него будет тяжело задавать им разные, да еще и блестящие вопросы, поэтому если у него не будет никаких вопросов, то это тоже прекрасно.

Я всегда оставляю в конце собеседования примерно пять минут, чтобы разрекламировать перед кандидатом компанию и его возможную работу. Это по-настоящему важно, *даже если вы не собираетесь его нанимать*. В случае, когда вам крупно повезет и кандидат окажется по-настоящему хорошим, вы захотите сделать все возможное, чтобы у него обязательно было желание пойти к вам работать. Но даже тогда, когда кандидат плохой, вам захочется, чтобы ваша компания ему понравилась и он уходил бы с положительным впечатлением.

Плохие вопросы

Да, только сейчас вспомнил, что должен дать примеры действительно плохих вопросов.

Прежде всего не задавайте незаконных вопросов. В Соединенных Штатах Америки к таковым относятся все вопросы о расе, религии, поле, национальном происхождении, возрасте, годности к воинской службе, ветеранском статусе, сексуальной ориентации или физических недостатках. Если в резюме кандидата сказано, что он служил в морской пехоте, кандидата нельзя спрашивать, даже чтобы ему польстить, был ли он в Ираке. Это нарушение закона, запрещающего дискриминацию на основе ветеранского статуса. Если в резюме сказано, что кандидат учился в “Технионе”, расположенном в Хайфе, не спрашивайте кандидата, пусть и неофициально, израильтянин ли он, даже если ваша жена израильтянка или вам нравится такое блюдо, как фалафель. Это нарушение закона, запрещающего дискриминацию на основе национального происхождения.

Не задавайте вопросов, по которым может показаться, что вас какие-то темы беспокоят или вы на их основе проводите дискриминацию, хотя на самом деле нет ни того, ни другого. Лучший пример таких вопросов, какой я могу вспомнить, это

спросить, есть ли у кандидата дети, а также состоит ли он в браке. В результате может создаться впечатление, что, по вашему мнению, кандидаты с детьми не посвятят работе достаточно времени или хотят "сбежать" в декретный отпуск. В основном придерживайтесь вопросов, целиком относящихся к работе, по которой идет собеседование.

И наконец, не задавайте головоломных вопросов, например, как из шести спичек построить четыре одинаковых правильных треугольника. Или чего-то там еще о пиратах, стеклянных шариках и секретных кодах. Эти вопросы из разряда "Ага!", т.е. ответ на них уже вам известен или нет. Знание ответа на такой вопрос просто означает, что эта головоломка вам известна. А вы как ведущий собеседование не узнаете ничего о том, "способный ли кандидат, доводит ли он дело до конца", если будете выяснять, удастся ли ему сделать некоторое умственное усилие.

В прошлом я прибегал и к "невозможным вопросам", еще известным как "вопросы с обратной стороны конверта". Вот один из классических примеров таких вопросов: "Сколько в Сиэтле настройщиков роялей?" Кандидат не знает ответа, но если он способный, то не сдастся и с удовольствием попытается дать правдоподобную оценку. Посмотрим, скорее всего это... что, в Сиэтле миллион жителей? И, возможно, у 1% из них есть рояль? А настраивать его надо через каждые два года? На настройку уходит 35 минут? Конечно же, все неправда, но хотя бы кандидат идет на проблему в атаку. Единственная причина, чтобы задать подобный вопрос, — дать вам возможность поговорить с кандидатом. "Хорошо, 35 минут, а время на дорогу от одних роялей до других?"

— Хороший вопрос. Если с настройщиком договариваются заранее, то он может составлять расписание визитов, и таким образом время на дорогу сводится к минимуму. Знаете, ведь лучше настраивать в понедельник только рояли в Рэдмонде, чем три раза в день ездить туда-сюда по трассе SR-520.

Хороший "невозможный вопрос" позволяет поговорить с кандидатом и помогает составить мнение, является ли он способным. А результатом плохого "Ага!"-вопроса о пиратах обычно является то, что кандидат просто смотрит на вас некоторое время, а затем говорит, что "сдается".

Итак, если к концу собеседования у вас сложится убеждение, что этот человек *способный* и *доводит дело до конца*, а четыре или пять проводивших с ним собеседование с вами согласятся, то, нанимая его, вы не ошибетесь. Но если у вас будут какие-то сомнения, то лучше дождитесь кого-то получше.

Оптимальное время на принятие решения о кандидате — примерно три минуты после окончания собеседования. В слишком многих компаниях проводившие собеседование могут тянуть дни и недели, перед тем как они дадут заключение. К сожалению, чем больше пройдет времени, тем меньше вы будете помнить.

Проводящих собеседование я прошу писать свое заключение *сразу же* после собеседования, т.е. *Принять* или *Не принять*, а затем один-два абзаца заключения. На это после окончания собеседования надо потратить еще 15 минут.

Если затрудняетесь дать ответ, есть очень простое решение — *Не нанимать*. Просто не нанимайте людей, в которых не уверены. Первые несколько минут у вас из-за этого будет маленький нервный срыв: что если хорошие кандидаты вам *никогда* не попадутся? Здесь все в порядке. Если работает ваша система резюме и телефонного заслона, то будут приняты примерно 20% участников очного собеседования. А когда вам попадется способный кандидат, который доводит дело до конца, то *вы его узнаете*. Ну а если не будете потрясены встречей, тогда ищите дальше.

Заключение соглашения

Хорошо. Вы создали прекрасное рабочее пространство, нашли нужных кандидатов, провели с ними хорошее собеседование и готовы сделать им предложение. Но все можно испортить. Поэтому как только вы нашли нужного человека, самое время сосредоточиться на том, чтобы узнать все детали и заключить соглашение.

Решение проблем

Как только вы сделали предложение, обратите особое внимание на то, какие у кандидата могут быть проблемы и будьте готовы потратить деньги на их решение. Переезд? Будет опла-

чен. Юрист-специалист по вопросам иммиграции? Учтем и это. Помочь вашей половине найти работу? Нужен брокер по недвижимости? Хотите проехаться, чтобы посмотреть варианты своего будущего жилья? Заплатим и за это.

Относитесь к ним, как к самураям

В знаменитом фильме Акиры Куросавы *Семь самураев* крохотная, обедневшая деревня, страдающая от постоянных бандитских набегов, пытается нанять семь самураев, чтобы те помогли ее жителям защищаться. Деревня бедная и вряд ли сможет даже прокормить самураев, не говоря о том, чтобы заплатить им, но семь отважных самураев, имеющих в целом добрые сердца, соглашаются прийти защищать деревню.

Этой деревней является ваша команда. Самураи — это программисты, которые, как вы надеетесь, придут решить ваши проблемы, принеся свой талант и опыт в обмен, может быть, на миску риса. Возможно, вы бедны и безнадежны, но, конечно же, знаете, как продемонстрировать свое уважение самураям, с чьей помощью вы хотите остаться целыми и невредимыми.

Не забывайте: самый обычный для нашего общества рекрутинг исходит из того, что у компании имеется нечто драгоценное (работа), к чему стремятся люди. Но в рекрутинге, связанным с наймом главных разработчиков, ситуация совершенно противоположная. Уделяйте очень и очень большое внимание деталям всего процесса рекрутинга, проведения собеседований и найма, чтобы исключить из него какую-то случайную деталь, которая посылала бы вашим кандидатам-звездам единственное сообщение: “Вы не достойны!”

Кодировщик Боб Розельман рассказывал с сарказмом историю о том, как он ходил в *Microsoft* на собеседование[5]. Когда настало время ланча, его оставили в комнате совещаний одного, принеся ему ланч в коробке. Это абсолютно недопустимое обращение с *любым* гостем, не говоря уже о потенциальном

[5] Находится в Web по адресу `codingslave.blogspot.com/2005/03/q-when-is-last-time-you-behaved.html`.

самурае — защитнике вашей деревни. Пусть вы что-то записали после утреннего собеседования, но ведь кандидаты испытали громадные трудности, сорвались с работы и, возможно, перелетели через всю страну, чтобы добровольно помочь решить ваши проблемы, и если даже они не совсем вам подходят, это не повод вести себя с ними иначе, чем с наивысшей степенью скромности и уважения.

И наконец, если вы кому-то должны сказать "нет", то делайте это быстро и с уважением. Сделать это вы должны, причем безотлагательно, хотя и не обязаны говорить кандидатам, *почему* они не совсем подходят. А приличия требуют, чтобы вы хотя бы разрешили им приехать еще раз при первой же возможности.

Снова о беге с препятствиями

Вернемся к самому первому вопросу: как вам нанять самых лучших людей? Как я уже говорил много страниц тому назад, для этого нет палочки-выручалочки: вы должны будете решить много проблем и проделать много работы, чтобы, переориентировав свою компанию, сделать ее местом, привлекательным для великих программистов. Вам придется уделить особое внимание всем препятствиям вместе и каждому в отдельности. Многое лежит за пределами компетенции директора по рекрутингу, управляющего по найму или даже генерального директора, но вы их обязаны делать.

В конце концов, у вас может появиться искушение сказать: "Да ну его. Зайду-ка я на один из этих Web-сайтов по найму программистов и заполучу какого-нибудь вундеркинда из Румынии, чтобы работал у меня за 20 баксов".

Только не сдавайтесь. Прекрасные люди намного ценнее обычных. В программировании они в 3–10 раз производительнее, а стоят всего лишь на 20–30% дороже. А еще они берут высокие ноты, которые никто другой взять не может.


MANAGER

Глава 7

Исправление неудачных команд

Получая в наследство уже существующую команду, вы становитесь чем-то вроде дантиста, принимающего пациента, который не встречался с дантистом около 20 лет, и его зубы начали портиться. Вы собираетесь делать три вещи.

1. Подробное исследование и рентген. После чего вы узнаете, где что не так: какие зубы целы, каким требуется "ремонт", а какие просто нужно удалить и заменить.
2. Много-много сверления, чтобы расчистить "дупла".
3. Заполнение пломбами и фарфоровыми протезами больших проемов, оставшихся после сверления.

Скорее всего будет довольно больно и вам придется засовывать пальцы в рот пациенту, но после нескольких недель болезненной работы у него снова появится прекрасная улыбка. И если вы исполните священный долг дантиста, то напугаете пациента до такой степени, что он будет избегать всех дантистов *следующие* 20 лет.

Да знаю я, вы всегда говорите пациенту, что ему *надо всерьез начинать чистить зубы вощеной ниткой*. Тогда скажите, как *это* получается у вас.

Как это не надо делать: измерения и стимулы

Крайне популярным и якобы научным считается исправление команды с помощью измерений и стимулов.

Например, вы решили начать считать строки кода, ежедневно создаваемого каждым разработчиком. Разработчики с максимальным числом строк получают премию. У кого сред-

нее количество строк, те сохраняют свое место. Попадите в низшие 20%, и вы уволены.

Это довольно распространенный прием управления, а он имеет *разрушительный* эффект.

Дело в том, что вы как программист можете тривиально увеличить количество строк, создаваемых вами каждый день. Просто иногда вставляйте в код пустые строки.

— Хорошо, — скажете вы. — Будем считать *непустые* строки кода.

— Ладно, — скажут программисты. — Будем писать *много* комментариев. Длинные пояснительные абзацы по три слова в каждой строке.

— Ну, комментарии я ценю, но эти — сплошной обман. Буду считать только *строки настоящего кода*.

— Ладно, не будете платить за комментарии, я их совсем писать не буду. А вызов каждой функции я раздвину так, что каждый ее аргумент будет на отдельной строке. Это будет разумно!

— Да-а! — думаете вы. — Может быть, есть то, что можно посчитать и с чем труднее делать махинации. Буду считать *выражения*; счет не изменится, даже если раздвигать их на несколько строк.

— Ладно. Вот у меня большой блок устаревшего кода, который больше не используется. Вообще-то его надо удалить, чтобы он не запутал другого программиста, читающего мой код. Но, удалив 500 строк кода, я на ближайшие два месяца стану *наихудшим программистом в команде*. Поэтому я этот блок просто "окружу" загадочным if-выражением, чтобы он никогда не выполнялся. Может быть, тогда не попаду под ваши санкции.

Этот диалог может продолжаться еще долго. Начальники явно верят, что есть *некая* единица измерения, с помощью которой можно мерить производительность, и надо просто выработать определенные правила, чтобы не было махинаций с этой единицей.

Программисты знают, в каких бы единицах вы ни измеряли, результаты измерений можно оптимизировать. Причем тривиально.

Как бы вы ни измеряли.

Поэтому измерять, откровенно говоря, бесполезно.

Измерять — это плохо?

Да, оказывается, что плохо. Профессор Роберт Д. Остин из Гарвардской школы бизнеса провел на эту тему большие исследования и написал классический труд *Измерение производительности и управление ею в организациях*[1]. Суть книги в том, что люди — не химические опыты, они сами себя осознают, и когда вы пытаетесь измерить что-то к ним относящееся, то они это ощущают. Кроме того, с помощью имеющихся у людей мозгов полученные значения измерений будут выглядеть так, как этим людям нужно.

Как показано в книге, какую бы новую единицу измерения вы ни устанавливали в "знающей" организации, т.е. в такой, работникам которой надо делать нечто более сложное, чем просто закручивать колпачки на тюбиках с зубной пастой, вначале будет заметно общее улучшение того, что вы хотите измерять. Программисты действительно будут стараться писать больше кода каждый день. Но очень скоро вы заметите, что работники придумывают ухищрения, позволяющие результатам измерений расти до небес, хотя реальная производительность будет на самом деле падать, ведь программисты начнут тратить больше времени на оптимизацию результатов измерений, которая будет делаться за счет качества выполняемой ими работы.

Более важно следующее: это происходит не просто из-за того, что вы не нашли совершенную единицу измерения, а *из-за самой природы "знающей" работы.*

Разные виды вносящих вклад

За свою карьеру я встречался и работал с сотнями талантливых программистов, но некоторые из них были выдающимися и не потому, что написали прекрасный код, а потому что в своей организации внесли поразительно ценный вклад, при-

[1] Robert D. Austin, *Measuring and Managing Performance in Organizations* (New York: Dorset House, 1996).

чем таким способом, который бы никак не учли с помощью традиционных средств, относящихся к производительности, — с помощью отчета или единицы измерения.

Мне известны разработчики, которые пишут собственный код не очень хорошо, зато считаются прекрасными отладчиками. Их часто можно найти в кабинете другого разработчика, ведь в команде быстро узнают, что их можно приглашать для решения по-настоящему сложных проблем.

А еще мне известны разработчики, которые, кажется, находятся в каком-то дремотном состоянии, тратя много времени на плавание по Интернету и на пробную загрузку новейших инструментов, в то время как их код остается ненаписанным. Некоторые из этих разработчиков просто непроизводительны, а некоторые предложили по-настоящему великие идеи, относящие к производительности, которые помогли всей организации добиться большего успеха.

Мне известна компания, откуда за недоведенную до конца обычную работу уволили разработчика, несмотря на то, что он выдвинул идею, которая спасла всю компанию. Мне также известна компания, где дали ужасную характеристику человеку, благодаря заботам которого (и больше ничьим!) все люди из его команды оставались счастливыми, веселыми и продуктивными. А все потому, что имевшиеся единицы измерения просто не давали распознать разные виды тех, кто вносит свой вклад.

Если бы не было такой плохой ситуации, что единицы измерения не измеряют, то они также бы укрепляли совершенно счастливые, производительные команды.

Действительно, некоторые производители просто не выдерживают своего веса

Даже хотя единицы измерения просто не походят к знающим работникам, все равно правда, что среди них имеются разработчики прекрасные, достойные и паршивые. Что интересно, все достаточно хорошо знают, кто есть кто. Но вы просто совсем не сумеете это измерить.

Тем не менее вам нужно разбить команду на три категории.

1. Прекрасных разработчиков.

2. Тех, кто нуждается в специальном усовершенствовании.

3. Безнадежных.

Если вы новый менеджер команды, то это разбиение быстрее всего сделать с помощью оценки коллег. Попросите каждого члена команды, чтобы он на условиях анонимности внес в какую-либо из этих категорий каждого из остальных членов команды. Увидев определенные тенденции (например, все считают, что Боб должен уйти), исследуйте их.

Увольнение балласта не всегда вредит боевому духу

Одной из причин, почему многие начальники боятся избавляться от балласта, является страх, что из-за этого в команде упадет боевой дух.

Однако часто это имеет обратный эффект.

Хорошие исполнители болезненно воспринимают то, что нужно доводить до ума неумело сделанную работу их сотрудников.

Прекрасные разработчики устают, занимаясь отладкой или переписыванием кода, который даже не может сносно работать. Их расстраивает явное равнодушие руководства к тому, что некомпетентность остается ненаказанной и что, следовательно, компетентность не ценится.

Итог такой "домашней уборки" в том, что если ее делать сразу, то боевой дух часто повышается.

"Резиновая комната"

Важно помнить, что вклад плохих исполнителей в создание кода не только *недостаточный*. Они не просто работают медленно, а вносят *отрицательный* вклад. В действительности они забирают время у хороших исполнителей, которые должны отлаживать их код, исправлять ошибки, сделанные плохи-

ми исполнителями, отвечать на их вопросы на уровне начинающих и разбираться с последствиями их неудачных решений. Так что плохих исполнителей надо фактически *оставлять за бортом команды*.

В некоторых организациях очень трудно увольнять балласт. В школьной системе Нью-Йорка настолько трудно справиться с учителем, нанятым на определенный срок, что начальство даже не пытается это делать. "В городских ведомостях на зарплату фигурируют сотни учителей, считающихся некомпетентными, дебоширами или виновными в сексуальных проступках. В школах боятся допускать их к преподаванию, поэтому помещают этих учителей в так называемые "резиновые комнаты", где они читают журналы, играют в карты и болтают за счет налогоплательщиков Нью-Йорка, которым обходятся по 20 млн. долл. ежегодно", — пишет Джон Стоссел в журнале *Reason*[2].

Конечная цель лидера команды состоит в том, чтобы улучшить производительность этой команды. Прекрасно, когда плохие исполнители уходят, но если из-за окружающей вас бюрократии, профсоюзных правил или национального законодательства добиться этого вы не можете, тогда используйте систему "резиновых комнат", т.е. что-либо, позволяющее не допускать этих людей к коду.

Улучшение производительности

Многие "проблемные" разработчики просто не имеют хоть какой-то способности быть прекрасными разработчиками. По моему глубокому убеждению, имеется класс людей, которые, вероятно, и очень способные, но, тем не менее, никогда не разберутся в указателях и рекурсии.

---

[2] John Stossel, "How to Fire an Incompetent Teacher", *Reason*, October 2006. Находится в Web по адресу `www.reason.com/news/show/36802.html`.

С другой стороны, для многих разработчиков просто нужен специальный подход, чтобы лучше управлять их производительностью. В таком случае руководителю надо устанавливать конкретные, реальные и достижимые цели, чтобы эти разработчики могли понимать, куда им двигаться. Это тренерская часть управления.

В других командах имеются способные разработчики, которые, однако, не могут работать вместе так, чтобы получались хорошие продукты. Эта тема в определенном смысле лежит за пределами данной книги, но ей посвящена другая моя книга, *Joel on Software*[3], в которой говорится обо всем, что может делать команда программистов, чтобы создавать еще лучший код, а также о принципах, применяемых буквально в любой высокотехнологичной команде. В основе той, другой книги лежит *тест Джоэла* — 12 важных моментов, которые, как мне кажется, все должны выполняться командами разработчиков. Этот *тест* дается в виде приложения к данной книге.

Методы управления

Если вы хотите управлять командой, компанией, армией или страной, то главная проблема, с которой вы столкнетесь, — это заставить всех двигаться в одном и том же направлении, что на самом деле является мягкой формулировкой выражения “заставить людей делать то, что вы хотите”.

Можете и так думать. Имея в своей команде больше одного человека, вы имеете разных людей с разными намерениями. Их желания отличаются от ваших. Если вы основатель “стартап”-компании, то, возможно, захотите быстро заработать много денег, чтобы пораньше отойти от дел и потратить следующие 20 лет на посещение конференций для женщин-блоггеров. Так что вы тратите свое время по большей части на

[3] Joel Spolsky, *Joel on Software: And on Diverse and Occasionally Related Matters That Will Prove of Interest to Software Developers, Designers, and Managers, and to those Who, Whether by Good Fortune or Ill Luck, Work with Them in Some Capacity* (Berkeley, CA: Apress, 2004); *Джоэл о программировании*, Символ-Плюс, 2006

поездки по Сэнд-Хилл-Роад и переговоры с представителями венчурного капитала, которые могут купить вашу компанию и слить ее с Yahoo!. Но программистку Дженис, которая работает у вас, совсем не волнует продажа компании и слияние ее с Yahoo!, так как Дженис не собирается зарабатывать деньги таким образом. Ее волнует написание кода на самом новом крутом языке программирования, ведь так весело изучать что-то новое. Кроме того, ваш финансовый директор целиком увлечен желанием выбраться из каморки, которую он делит с системным администратором Трекки Манстером, поэтому работает над новым бюджетным предложением, которое всего лишь показывает, сколько денег вы сэкономите, переехав в более просторный офис, расположенный (какое совпадение!) в двух минутах ходьбы от его дома.

Проблема того, как заставить людей двигаться в вашем направлении (или хотя бы *в одном и том же* направлении), характерна, конечно же, не только для "стартапов". С этой же самой фундаментальной проблемой сталкивается политический лидер, который избран после того, как дал обещание ликвидировать расточительство, коррупцию и мошенничество в правительстве. Мэру хочется быть уверенным в том, что он легко получит одобрение города на новый строительный проект. Городские строительные инспекторы хотят и дальше получать взятки, к которым привыкли.

С этой же проблемой сталкивается и военачальник. Ему, возможно, требуется команда солдат, чтобы атаковать врага, даже если каждому отдельному солдату на самом деле хочется съежившись спрятаться за скалой, чтобы врага атаковал вместо него кто-то другой.

Вот три самых распространенных подхода, которые вы могли бы применить:

- командования и контроля;
- экономической мотивации;
- отождествления.

Конечно, в природе можно найти и другие методы управления (например, такого экзотического, как *Дьявол носит*

"Прада", джихада, харизматического культа и перехода от одного метода к другому). Но о предыдущих трех методах стоит поговорить особо.

Метод командования и контроля

Фридрих Великий знаменит благодаря такому высказыванию: "Солдаты должны бояться офицеров больше всех опасностей, с которыми сталкиваются... Добрая воля никогда не заставит обычного солдата противостоять таким опасностям; это он будет делать только из-за страха[4]".

Способ командования и контроля основан на армейском управлении. В основу этого способа положена идея: человек делает то, что вы ему скажете, а если не делает, вы будете на него орать, пока он не сделает это, а когда такой способ не подействует, вы его отправите посидеть на гауптвахте, а если гауптвахта его не научит, вы отправите его в наряд чистить лук на подводную лодку, где он будет делить невероятно малое (точнее говоря, примерно 0,06 $м^3$) личное пространство с деревенским парнем, который и представления не имеет, что надо чистить зубы.

Можете использовать миллион прекрасных приемов. Некоторые идеи можете почерпнуть, взяв напрокат фильмы "Блокси Блюз" и "Офицер и джентльмен".

Некоторые руководители пользуются этим приемом, потому что научились ему во время военной службы. Другие выросли в авторитарных семьях или странах и считают его естественным средством добиться согласия. А третьи ничего лучшего просто не знают. Для армии же он подходит, подойдет и для Интернет-"стартапа"!

Когда этот метод применяют в высокотехнологичной команде, то в нем, как оказалось, проявляется три недостатка.

---

[4] Frederick, King of Prussia, *Frederick the Great and the Art of War*, edited and translated by Jay Luvaas (New York: Da Capo Press, 1999).

Прежде всего, люди по-настоящему очень его не любят, особенно остряки-самоучки разработчики программ, которые действительно очень способные и обычно думают, что знают больше, чем кто-то другой, причем по совершенно уважительным причинам, ведь иногда так оно и есть. Поэтому этот метод их по-настоящему раздражает, если им дают команду сделать что-то "потому, что надо". Но нет достаточно уважительной причины, чтобы совсем отбрасывать этот метод, ... здесь попробуем быть рациональными. У высокотехнологичных команд много целей, но сделать каждого счастливым вряд ли считается целью номер один.

Более практический недостаток метода командования и контроля в том, что у начальства буквально нет времени, чтобы управлять на микроуровне, так как на это просто не хватит начальников. На военной службе можно отдать приказ сразу большой команде людей, так как обычно в ней все делают одно и то же. Целой команде в 28 человек вы можете сказать: "Чистить стволы!" — после чего немножко вздремнуть, а затем на веранде офицерского клуба побаловаться охлажденным чаем со льдом. А в командах разработчиков программ каждый делает что-то свое, так что попытки руководить на микроуровне превращаются в *руководство с помощью набегов на микроуровне*. Вот таким образом в приступе активности вы руководите на микроуровне одним разработчиком, а затем на пару недель внезапно исчезаете из его жизни, чтобы управлять на микроуровне уже другим разработчиком. Проблема с управлением на микроуровне с помощью набегов состоит в том, что вы не находитесь на одном месте достаточно долго, чтобы увидеть, почему ваши решения не подходят, или чтобы корректировать курс. На самом деле все, что вы сумеете, — это в кои-то веки "пустить под откос поезд" бедных программистов, чтобы всю следующую неделю они искали "вагоны" своего "поезда", ставили их снова "на рельсы" и цепляли "вагоны" один за другим, слегка ошарашенные вашим "экспериментом".

Третий недостаток состоит в том, что в высокотехнологичной компании исполнители всегда более информированы, чем руководство, поэтому в действительности они находятся в самом лучшем положении для принятия решений. Когда на-

чальник забредает в офис, где два разработчика спорят, какой способ сжатия изображений самый лучший, то *наименее* информированным человеком будет начальник, т.е. *самым последним* из тех, кому бы вы доверили принятие технического решения. Вспоминаю, когда Майк Мейплс был ну очень большим начальником, руководившим Microsoft Applications, он всегда отказывался принимать чью-то сторону в технических спорах. Со временем люди поняли, что им незачем ходить к нему за приговором. Это их заставляло обсуждать тот или иной вопрос по существу, и вопросы всегда решались в пользу того человека, который был лучшим в споре, причем, мне кажется, всегда решались самым лучшим способом, какой только был возможен.

Если способ командования и контроля так плохо подходит для управления командой, почему же военные его используют?

Это мне стало понятно в унтер-офицерской школе. В 1986 году я служил парашютистом в Израиле, и, как сейчас думаю, вероятно, самым худшим изо всех парашютистов, которые когда-либо там были.

Есть несколько стандартных приказов, отдаваемых солдатам. Приказ номер один: если ты на минном поле, *замри*. Понятно? Ему вас постоянно обучают во время простых тренировок. Время от времени инструктор кричал: “Мины!” — и все должны замереть, так что со временем это входит в привычку.

Стандартный приказ номер два: если тебя атакуют, *беги навстречу атакующим и стреляй*. Стрельба заставляет их искать укрытия, и они не могут стрелять в вас. Бег навстречу атакующим заставляет вас приближаться к ним, и вам легче в них прицеливаться и, следовательно, легче их убить. В этом стандартном приказе также много смысла.

Ладно, а теперь вопрос для собеседования. Что делать, когда вы находитесь на минном поле и вас начинают обстреливать?

Это не такая уж гипотетическая ситуация, а скорее, довольно неприятное попадание в засаду.

Как оказалось, правильный ответ состоит в том, чтобы, игнорируя минное поле, бежать навстречу атакующим и стрелять.

Разумное объяснение этого состоит в том, что если вы замрете, вас всех застрелят по одному, но если будете атаковать,

то погибнут лишь некоторые из вас, наскочив на мины, поэтому лучше поступать именно так.

Но дело в том, что при таких обстоятельствах ни один здравомыслящий солдат атаковать не будет. У каждого солдата есть громадный стимул схитрить — замереть на месте, и пусть атакуют остальные, более мужественные солдаты. Получается что-то вроде дилеммы заключенных.

В ситуации между жизнью и смертью командирам нужна уверенность, что они могут выкрикивать приказы и солдаты будут исполнять их, даже если такое поведение будет для солдат равносильно самоубийству. Это значит, что солдат нужно программировать на подчинение таким способом, который, скажем, для программистской компании не настолько важен.

Другими словами, военные используют метод командования и контроля, потому что это единственный способ заставить 18-летних идти в атаку по минному полю, а не потому, что он является лучшим методом управления в любой ситуации.

В частности, в командах разработчиков программ, в которых хорошие разработчики могут работать там, где им хочется, игра в военщину становится *очень* неприятной, и в таком случае видно, что вы точно не собираетесь кого-то удерживать в своей команде.

Метод экономической мотивации

Анекдот. В еврейском местечке где-то в России XIX века жил бедный еврей. Как-то заезжает к нему верхом казак.

— Чем ты кормишь того цыпленка? — спрашивает казак.

— Только хлебными крошками, — отвечает еврей.

— Как ты смеешь давать такой мерзкий корм прекрасному русскому цыпленку! — сказал казак и ударил еврея палкой.

На следующий день казак приехал снова.

— А чем сегодня ты кормишь того цыпленка? — спросил он еврея.

— Ну, тремя разными блюдами. Это свежескошенная трава, осетровая икра, а на десерт — чашечка густых сливок, по-

сыпанных импортными французскими шоколадными трюфелями.

— Идиот! — говорит казак, ударяя еврея палкой. — Как ты смеешь скармливать хорошую пищу мерзкому цыпленку!

На третий день казак спрашивает снова:

— Чем ты кормишь того цыпленка?

— Ничем! — защищается еврей. — Я даю ему копейку, и он покупает все, что хочет.

(Пауза для смеха)

(Нет?)

(Там-да-да-дам!)

(Все равно не смешно?)

(Ну и ладно)

Я использую термин "Экономическая мотивация" (или "Экономика 101") в несколько ироническом смысле. Справка для моих читателей-неамериканцев: во многих отделениях американских колледжей по любому предмету читается простой вводный курс (с номером 101). Управление "Экономика 101" — это стиль, используемый людьми, которые знают ровно столько экономической теории, чтобы быть опасными.

Управленец в стиле "Экономика 101" предполагает, что деньги являются побудительным мотивом для каждого и что лучший способ заставить людей делать то, что вам нужно, — это раздавать им финансовые поощрения и наказания, создавая таким образом стимулы.

Например, компания *AOL* может платить сотрудникам своих call-центров премию за каждого клиента, которого они убедили *не отменять* свою подписку.

Программистская компания может давать премиальные программистам, сделавших меньше всего ошибок.

Этот способ такой же эффективный, как и раздача денег вашим цыплятам, чтобы они сами покупали себе пищу.

Серьезная проблема здесь в том, что внутренняя мотивация заменяется внешней.

Внутренняя мотивация — это ваше собственное естественное желание сделать работу хорошо. Сначала у людей имеется очень сильная внутренняя мотивация. Они хотят хорошо работать. Они *хотят* помочь людям осознать — самый большой

интерес этих людей в том, чтобы продолжать платить компании *AOL* 24 долл. в месяц. Они *хотят* писать код с меньшим количеством ошибок.

Внешней выглядит мотивация, идущая снаружи, как, например, тогда, когда вам платят за какое-то особое достижение.

Внутренняя мотивация намного сильнее внешней. Люди работают намного старательнее над тем, что они *действительно хотят делать*. И здесь нет большого противоречия.

Но когда вы предлагаете людям деньги именно за то, что они и так хотят делать, то эти люди страдают от так называемого *эффекта сверхоправдания*. "Я обязан писать код, свободный от ошибок, ведь мне нравится получать за это деньги", — думают они, и внешняя мотивация *заменяет* внутреннюю. Так как внешняя мотивация намного слабее внутренней, вы в конечном итоге фактически *уменьшаете* желание людей делать хорошую работу. Когда вы прекращаете платить премиальные или когда люди решат, что деньги их волнуют не очень сильно, то они больше не будут думать, как создавать код, свободный от ошибок.

Другая большая проблема, связанная с методом управления "Экономика 101", — это склонность людей довольствоваться локальными максимумами. Они найдут какой-то способ улучшения именно того, за что вы платите, но не достигнут при этом того, что вы действительно хотите.

Например, ваш специалист по удержанию клиентов, желая получить соответствующие премиальные за поддержку клиента, будет его преследовать с таким остервенением, что в газете *New York Times* появится большая передовица с рассказом о том, как ужасно вы "обслуживаете" клиентов. И хотя поведение специалиста доводит до максимума то, за что вы ему платите (удержание клиентов), тем не менее оно не доводит до максимума то, о чем вы действительно заботитесь (прибыль). А когда вы пытаетесь вознаградить специалиста за прибыль, полученную компанией, например, дав ему 13 акций, то понимаете, что это не то, чем он действительно может управлять, т.е. ваша мера поощрения ни к чему не приводит.

Используя метод управления "Экономика 101", вы подталкиваете разработчиков к тому, чтобы они "играли" с системой.

Предположим, вы решили платить премиальные разработчику, допустившему наименьшее количество ошибок. Теперь при каждой попытке тестировщика сообщить об ошибке возникает серьезный спор, и обычно разработчик убеждает-таки тестировщика, что на самом деле это не ошибка. Или тестировщик соглашается "неофициально" сообщать разработчику об ошибке еще до того, как регистрировать ее в системе отслеживания ошибок. И теперь никто не будет пользоваться этой системой. "Учет ошибок" продолжается, но контроль над ними вы теряете.

Разработчики таким образом хитрят. Что бы вы ни пытались измерить, они найдут способ увеличить измеряемую величину до максимума, и вы никогда не добьетесь того, чего хотите.

А наибольшая проблема управления с помощью метода "Экономика 101" состоит в том, что это вовсе не управление, а *отказ от управления*. Умышленное нежелание понять, как можно улучшить положение дел. Это знак того, что руководство просто не знает, как учить людей работать лучше, поэтому оно заставляет каждого работника, задействованного в системе, заниматься этим самостоятельно.

Вместо того чтобы учить разработчиков приемам создания надежного кода, вы просто снимаете с себя ответственность, платя им за эту работу. Теперь каждый разработчик сам должен об этом думать.

Что касается более земных дел, например, работы за прилавком в сети кафетериев *Starbucks* или диспетчером на телефоне в *AOL*, то маловероятно, чтобы средний работник думал о том, как лучше всего выполнять свои обязанности. Вы можете зайти в любой другой кафетерий в стране, не входящий в сеть *Statbucks*, и заказать маленькую чашечку очень горячего кофе-латте с соевой карамелькой, и вам придется повторять свой заказ снова и снова: один раз — тому, кто готовит кофе, второй раз — тому же человеку, когда он забудет, что же вы сказали, и, наконец, кассиру, чтобы он посчитал, сколько вам надо платить. Это результат того, что работников никто не учит, как надо делать лучше. Нигде об этом не думают, кроме как в *Starbucks*, где на стандартных тренировках изучают полную систему наименований, где заказы пишут на чашках и ис-

пользуют такой вызов заказов, который гарантирует, что клиентам достаточно один раз указать номер их напитка. Эта система, изобретенная еще в штаб-квартире *Starbucks*, работает прекрасно, но работникам в других системах она и близко не знакома.

Сотрудники вашей службы работы с клиентами большую часть дня проводят в разговорах с этими клиентами и не имеют ни времени, ни склонности, ни навыков понять, как лучше делать свое дело. В службе удержания клиентов никто не собирается накапливать статистику и измерять, какие приемы удержания клиентов работают лучше всего, и посылает подальше немногочисленных блоггеров. Они недостаточно заботливы, недостаточно способны, не обладают достаточной информацией и слишком заняты повседневной работой.

Ваша задача как руководителя состоит в том, чтобы разобраться в системе. Именно-Так-Вы-Получите-Большие-Бабки.

Если вы еще ребенком зачитывались Эйном Рэндом или изучали экономику лишь один семестр и не дошли до положения, что польза не измеряется в долларах, то, возможно, думаете, что, установив упрощенную систему премиальных и Платя-За-Производительность, вы получите довольно разумную систему управления. Но это не так. Лучше займитесь управлением своей работы и прекратите кормить своих цыплят копейками.

Разработчики, являясь листьями в дереве, обладают наибольшей информацией; управление на микроуровне или выкрикивание команд по методу командования и контроля явно приводят к не самым лучшим результатам. Но вы, тем не менее, должны создать систему. Вам нельзя отделываться взятками от ответственности за обучение персонала. Руководство вообще-то должно организовать такую систему, чтобы люди могли доводить дело до конца, оно должно избегать замены внутренней мотивации на внешнюю и не должно заходить слишком далеко, т.е. наводить страх и рявкать на людей, отдавая конкретные приказания.

Теперь, когда я раскритиковал такие методы управления как командования и контроля, а также “Экономика 101”, расскажу еще об одном методе, которым могут пользоваться ру-

ководители, чтобы направить людей в нужном направлении. Я его называю методом отождествления.

Метод управления с помощью отождествления

В управлении требуется настоящее умение, чтобы заставить людей *отождествлять* себя с целями, которых вы пытаетесь достичь. Этот метод намного сложнее, чем все остальные, и чтобы он все-таки заработал, надо хорошо уметь общаться с людьми. Но если им пользоваться правильно, он будет эффективнее любого другого метода.

Проблема с управлением по методу "Экономика 101" заключается в том, что он вытесняет внутреннюю мотивацию. А метод отождествления — это способ *создания* такой мотивации.

Чтобы быть руководителем — последователем метода отождествления, вы должны, призвав все свои коммуникативные способности, заставить своих подчиненных, чтобы они отождествляли себя с целями организации. Тогда у них будет сильная мотивация, если вам потребуется дать подчиненным информацию, чтобы они следовали в правильном направлении.

Как сделать так, чтобы люди отождествляли себя с организацией?

Этому помогает ситуация, когда цели организации являются благородными или некоторым образом ощущаются как благородные. Компания *Apple* создала почти фанатичное отождествление почти целиком с помощью сюжетно-тематической картинки, которая берет начало с единственной рекламы на Суперкубке 1984 года: мы против тоталитаризма. Этот прием не слишком яркий, чтобы быть примером для подражания, тем не менее он сработал. Что касается моей компании, *Fog Creek Software*, то мы храбро боремся против убийства котят. Ура!

Ну а если серьезно, то мне очень удобно пользоваться таким методом, как совместные приемы пищи. Я всегда отличался тем, что устраивал ланч совместно с сотрудниками, он заказывается каждый день на всех сотрудников *Fog Creek* и совместно уничтожается нами всеми за одним большим столом. Трудно пред-

ставить, как сильно это влияет на восприятие компании как одной семьи — мне кажется, семьи в хорошем смысле. За шесть лет от совместного ланча еще никто не отказался.

Одна из целей нашей программы интернатуры (пусть для некоторых студентов, проходящих у нас летнюю интернатуру, эта цель покажется чудачеством) — добиться, чтобы люди отождествили себя с нью-йоркцами, и им было бы легче свыкнуться с мыслью о переезде сюда после окончания колледжа и о работе у нас полную рабочую неделю. Чтобы добиться этой цели, мы имеем довольно плотный список внеклассных летних мероприятий: две бродвейские постановки, путешествие на Top of the Rock, лодочное путешествие вокруг Манхэттена, поход на игру с участием Yankees, дни открытых дверей, чтобы можно было встречать больше нью-йоркцев, а также поход в музей. Майкл и я устраиваем вечеринки в своих квартирах, как для того, чтобы радушно принять студентов, проходящих интернатуру, так и затем, чтобы они своими глазами увидели жизнь в нью-йоркской квартире, а не только в студенческом общежитии, в котором мы их поселили.

В общем-то управление по методу отождествления требует, чтобы вы создали сплоченную, спаянную команду, члены которой чувствуют себя как одна семья, и у них было бы чувство преданности и взаимопомощи по отношению друг к другу.

Впрочем, у метода есть и вторая часть: дать людям нужную им информацию, чтобы организацию можно было вести в нужном направлении.

Сегодня в мой кабинет уже приходил Бретт, чтобы обсудить даты выпуска бета-версии FogBugz 6.0. Он склонялся к апрелю 2007 года, а я — к декабрю 2006 года. Конечно, если бы мы выбрали апрель, то имели бы время, чтобы навести намного больше глянца и усовершенствовать многие части продукта, а если бы запустили в декабре, то нам, вероятно, пришлось бы урезать целую кучу прекрасных новых возможностей.

Впрочем, как я объяснил Бретту, весной мы хотим нанять шесть новых сотрудников, а позволить себе это у нас будет намного меньше шансов, если к тому времени не будет выпущен продукт FogBugz 6.0. Итак, то, как я закончил встречу с Бреттом, было способом заставить его понять точные финан-

совые мотивы, какими я руководствовался при выборе более ранней даты выпуска. И поскольку он их знает, я уверен, что Бретт примет правильное решение... пусть даже не обязательно *мое* решение. А может быть мы получим хорошую выручку от продаж и без FogBugz 6.0, и поскольку Бретт разбирается в основных финансовых параметрах, он поймет, что тогда можно будет попридержать 6.0, чтобы добавить ему еще какие-то возможности. Главное то, что, делясь с Бреттом информацией, я помогаю ему, даже если обстоятельства изменятся, правильно вести себя по отношению к *Fog Creek*. Если бы я пытался давить на Бретта, предлагая ему вознаграждение за каждый из дней, оставшихся с момента выпуска продукта до начала апреля, то этим стимулировал бы его переносить работу имеющейся системы отлова ошибок на пресловутый *вечер*. А если бы я пытался давить на Бретта с помощью метода командования и контроля, приказывая ему подготовить в срок код, свободный от ошибок, *так его перетак*, он, возможно, сделал бы эту работу, но возненавидел бы ее и уволился.

Стилей управления имеется столько, сколько и самих управляющих. Я назвал три главных стиля, двое из которых легкие, но разрушительные, а один трудный, но функциональный. Однако истина состоит в том, что во многих организациях, где разрабатываются программы, управление осуществляется более временным способом (“лишь бы только он работал”), который может меняться день ото дня или от сотрудника к сотруднику.

Приложение

Тест Джоэла

Слышали ли вы о SEMA? Это довольно таинственная система для измерения того, насколько хороша команда программистов. Нет, *подождите! Не уходите читать о SEMA!* Чтобы только понять этот материал, может потребоваться примерно шесть лет. Поэтому я и пришел с моим собственным, довольно безответственным и неаккуратным тестом, предназначенным для оценки качества команды разработчиков[1]. Главное его преимущество в том, что на его проведение уходит примерно три минуты. А все остальное время, которое вы сэкономите, можете посвятить обучению в медицинском колледже.

Тест Джоэла

1. Пользуетесь ли вы системой контроля версий?
2. Можете ли вы собрать проект за один шаг?
3. Выполняете ли вы ежедневные билды?
4. Используете ли вы базу данных ошибок?
5. Исправляете ли вы ошибки, перед тем как писать новый код?

---

[1] Опубликованный первоначально на Web-сайте *Joel on Software* (`www.joelonsoftware.com`) еще в 2000 году, *Тест Джоэла* уже стал классикой. Он появляется в книге *Joel on Software* вместе с подробным обсуждением каждого из пунктов.

Joel Spolsky, *Joel on Software: And on Diverse and Occasionally Related Matters That Will Prove of Interest to Software Developers, Designers, and Managers, and to Those Who, Whether by Good Fortune or Ill Luck, Work with Them in Some Capacity* (Berkeley, CA: Apress, 2004); *Джоэл о программировании*, Символ-Плюс, 2006.

6. Есть ли у вас актуальный план работ?
7. Имеется ли у вас спецификация?
8. Работают ли программисты в тихих условиях?
9. Используете ли вы лучшие средства, какие только можно купить?
10. Имеются ли у вас тестеры?
11. Пишут ли кандидаты на работу код во время собеседования?
12. Проводите ли вы коридорное тестирование удобства использования программы?

В *Тесте Джоэла* прекрасно то, что на каждый вопрос можно быстро ответить **Да** или **Нет**. Не нужно вычислять количество строк кода в день или среднее количество ошибок на каждое изменение. На каждый ответ "Да" присваивайте команде единицу. Неприятно в этом тесте то, что его *уж точно нельзя* использовать для проверки безопасности программ, используемых на вашей атомной электростанции.

Счет, равный 12, — это прекрасно, 11 — удовлетворительно, а 10 или ниже означает, что вы столкнулись с серьезными проблемами. Правда заключается в том, что большинство программистских организаций наберет всего 2 или 3 балла, и им требуется серьезная помощь, ведь такие компании, как *Microsoft*, все время работают со счетом 12.

Конечно, это не единственные факторы, определяющие успех или неудачу; например, у вас прекрасная команда программистов работает над продуктом, который никому не нужен, соответственно он и не продается. Еще можно представить команду бездельников, которая ничего этого не делает, но все равно справляется с созданием невероятного продукта, способного изменить мир. Но, при прочих равных условиях, если у вас эти 12 пунктов в порядке, то вы будете иметь дисциплинированную команду, способную всегда обеспечивать успех.

1. Пользуетесь ли вы системой контроля версий?

Я пользовался коммерческими пакетами контроля версий, а также пользовался бесплатным CVS, и, скажу я вам, CVS *замечательный*[2]. Но если у вас нет никакой системы контроля версий, вам следует настоять, чтобы программисты работали сплоченно. Ведь без этого они не могут узнать, что сделали другие, и код не так легко вернуть в то состояние, которое было до появления ошибки. В системах контроля версий прекрасно еще и то, что сам этот код проверяется на жестких дисках каждого программиста, и я никогда не слышал, чтобы в проекте, где используется этот контроль, терялось много кода.

2. Можете ли вы собрать проект за один шаг?

Здесь подразумевается: сколько действий нужно, чтобы из кода с самыми последними изменениями создать готовое приложение? В хорошей команде можно запустить единственный скрипт, который выполнит полную проверку с самого начала, пересоберет все строки кода, создаст исполняемые файлы, причем во всевозможных версиях, для всех языков и комбинаций #ifdef, создаст инсталляционный пакет, а также окончательный носитель: компакт-диск, загрузочный Web-сайт, да что угодно.

Если на процесс требуется больше одного действия, то это чревато ошибками. И когда срок поставки приближается, то хочется иметь очень быстрый цикл, в котором исправляется "последняя" ошибка, создаются окончательные EXE-файлы и т.д. А если требуется 20 действий, чтобы выполнить компиляцию кода, запуск сборщик инсталляционных пакетов и т.д., то вы можете сойти с ума или допустить глупые ошибки.

---

[2] После того как я написал ту статью, появился такой продукт, как Subversion, сделавший CVS устаревшим. Subversion даже *более* замечательный, чем CVS.

Именно по этой причине последняя компания, в которой я работал, перешла с WISE на InstallShield: требовалось, чтобы процесс инсталляции мог запускаться из скрипта, автоматически, ночью, с помощью планировщика задач NT, а WISE не мог запускаться по расписанию ночью, поэтому мы от него избавились. (Добрые люди из WISE заверили меня, что самая последняя версия этого продукта все же поддерживает ночные билды.)

3. Выполняете ли вы ежедневные билды?

При использовании контроля исходного кода иногда какой-нибудь программист случайно делает вставку, которая нарушает нормальных ход сборки. Например, он добавляет новый исходный файл, и все прекрасно компилирует на своей машине, но при этом забывает добавить исходный файл в репозиторий кода. Заблокировав свою машину, он идет домой, забывчивый и счастливый. Однако после этого все остальные не могут работать, поэтому они также идут домой (правда в гораздо худшем настроении).

Нарушение хода сборки настолько — вещь настолько распространенная и неприятная, что стоит делать ежедневные билды, чтобы ни один сбой не прошел незамеченным. В больших командах хороший способ сразу исправить нарушения состоит в том, чтобы делать ежедневные билды каждый день после полудня, скажем, во время ланча. Каждый программист до ланча выполняет столько изменений, сколько ему заблагорассудится. Когда они все уходят на ланч, выполняется билд. Если все прошло хорошо, то это прекрасно! Каждый получает последнюю версию и продолжает работать. Если же произошел сбой, вы исправляете ошибки, но при этом каждый программист может продолжать работать с утренней, ненарушенной рабочей версией.

У нас, в команде разработчиков Excel, было правило, согласно которому каждый, кто нарушил компоновку, в качестве “наказания” должен был следить за компоновкой, пока ее не нарушит кто-то другой. Это был хороший стимул не нарушать

ее и хороший способ пропустить каждого программиста через процесс компоновки, чтобы все знали, как он выполняется.

4. Используете ли вы базу данных ошибок?

Меня не волнует, что вы скажете. Если вы разрабатываете код, даже в команде из одного человека, то без упорядоченной базы данных, в которой указаны все известные ошибки, обнаруженные в коде, поставляемый вами код будет низкого качества. Многие программисты думают, что смогут удержать список ошибок у себя в голове. Чушь. Я сразу не могу помнить больше двух-трех ошибок, а на следующее утро или в горячке поставки даже они забываются. Вам абсолютно необходимо официально регистрировать ошибки.

Базы данных ошибок могут быть сложными или простыми. В минимально полезной базе данных по каждой ошибке должны быть представлены следующие данные.

- Действия, выполняемые для воспроизведения ошибки.
- Ожидаемое поведение.
- Наблюдаемое (вызванное ошибкой) поведение.
- Кому поручено исправить.
- Исправлена ошибка или нет.

Если единственным препятствием к регистрации ошибок является сложность соответствующего программного обеспечения, то всего лишь заведите простую таблицу из пяти столбцов с таким нужными полями и *начинайте ее использовать.*

5. Исправляете ли вы ошибки, перед тем как писать новый код?

Работа над самой первой версией Microsoft Word for Windows тянулась похоронным маршем. Проект беспрестанно застревал и зависал. Целая команда проводила над ним долгие часы, проект откладывался снова и снова, а напряжение было

невероятным. И когда через годы эта штуковина наконец-то была отправлена в продажу, вся команда была за счет *Microsoft* отправлена в Канкун на отдых, а затем ее собрали для серьезного "разбора полетов".

Вскрытие показано, что менеджеры проекта очень настаивали на соблюдении расписания, что программисты просто работали второпях, создавая крайне плохой код, так как фаза исправления кода не входила в официальное расписание. Не было попытки уменьшать количество ошибок. Как раз наоборот. Рассказывают, что один программист, который должен был писать код для вычисления высоты текстовой строки, просто написал "return 12;", а затем ждал сообщения об ошибке, чтобы узнать, каким образом его функция работает не всегда корректно. Расписание было просто списком возможностей, ожидающих, пока их превратят в ошибки. Впоследствии это назвали "методологией бесконечных дефектов".

Чтобы исправить ситуацию, в *Microsoft* приняли так называемую "методологию отсутствия дефектов". Над ней посмеивались многие из программистов, работавших в компании: это выглядело так, будто руководство думало, что количество ошибок можно уменьшить, издав соответствующее распоряжение. На самом же деле "отсутствие дефектов" означало, что наивысший приоритет имеет устранение ошибок *перед* написанием любого нового кода. И вот почему.

В общем, чем дольше вы тянете с исправлением ошибки, тем дороже ее исправить (с учетом времени и денег).

Например, если вы сделаете опечатку или синтаксическую ошибку, замеченные компилятором, то исправить их обычно очень легко.

Когда в вашем коде завелась ошибка, которую вы первый раз увидели при его первом запуске, то сразу сумеете ее исправить, так как еще хорошо помните весь код.

Если вы найдете ошибку в коде, написанном вами несколько дней назад, то для ее отслеживания вам потребуется некоторое время. Но заново перечитав написанный вами код, вы все вспомните и сумеете исправить ошибку в разумные сроки.

Но если вы найдете ошибку в коде, написанном вами несколько месяцев назад, вы, вероятно, уже многое забудете, что

относится к этому коду, и его будет намного труднее исправить. В это же время вам, возможно, придется исправлять код, написанный кем-то *другим*, кто как раз находится в отпуске на Арубе. Тогда исправление ошибки будет чем-то вроде научного исследования: вам приходится быть неторопливым, методичным и педантичным, и вы не сумеете точно сказать, когда именно закончите работу.

И если вы найдете ошибку в коде, который *уже отправлен в продажу*, вам придется понести невероятные расходы, чтобы ее исправить.

Одна из причин, почему надо исправлять ошибки сразу, состоит в том, что на это уйдет меньше времени. Есть и другая причина, связанная с тем, что легче *сказать*, за какое время можно написать новый код, чем за какое время — исправить ошибку. Например, если бы я спросил вас, за какое время можно написать код для сортировки списка, то вы мне ответили бы с довольно хорошей точностью. Но если бы я вас спросил, за какое время можно исправить ошибку, из-за которой не работает ваш код при установленном Internet Explorer 5.5, то вы не смогли бы даже приблизительно подсчитать, так как не знаете (по определению) *из-за чего* возникла ошибка. Для ее отслеживания может потребоваться как три дня, так и две минуты.

Это значит, что если бы вы имели расписание по исправлению множества ошибок, то такое расписание было бы ненадежным. Но если бы вы исправили все *известные* ошибки, а все оставшееся было бы новым кодом, то ваше расписание оказалось бы потрясающе точным.

Другая прекрасная вещь, связанная с сохранением счетчика ошибок на нуле, состоит в том, что вы можете намного быстрее реагировать на вызовы конкурентов. Для некоторых программистов это означает сохранение продукта все время *готовым к продаже*. И тогда если в продукте конкурента появится потрясающая новая возможность, которая крадет ваших клиентов, вы можете в своем продукте реализовать только эту возможность, а затем поставить его на рынок, не испытывая нужды в исправлении большого количества накопившихся ошибок.

6. Есть ли у вас актуальный план работ?

Что заставляет нас составлять расписания? Если ваш код очень важен для бизнеса, то найдется много причин, почему бизнесу так важно знать, когда код будет сделан. Все знают, что программистов раздражает составление расписаний. “Когда будет сделан, тогда и будет сделан!” — кричат они на бизнесменов.

К сожалению, только этим дело не исчерпывается. Бизнесменам надо задолго до поступления кода в продажу принять слишком много решений, относящихся к планированию демонстрационных версий, коммерческих показов, рекламы и т.д. И единственный способ сделать это — иметь расписание и вовремя его обновлять.

Другая важная причина иметь расписание состоит в том, что оно заставляет вас решать, какие возможности надо делать, а какие (наименее важные) найти и *отбросить*, вместо того чтобы обрастать мелочью (так называемый “эффект снежного кома”).

7. Имеется ли у вас спецификация?

Написание спецификаций похоже на чистку зубов вощеной ниткой: все согласны, что это полезно, но никто такого не делает.

Не знаю, почему это так, но, скорее всего, потому, что большинство программистов ненавидят писать документы. В результате, когда команды состоят исключительно из программистов, атакующих проблему, то эти программисты предпочитают выразить свое решение в коде, а не в документах. Они скорее углубятся в написание кода, чем вначале создадут спецификацию.

Обнаружив неполадки на этапе проектирования, вы можете легко их исправить, отредактировав несколько строк текста. Если код уже написан, то стоимость исправления неполадок намного выше, как эмоционально (людям очень не нравится выбрасывать код), так и с точки зрения времени, отсюда и со-

противление настоящему исправлению проблем. Программное обеспечение, скомпонованное без спецификации, обычно оказывается плохо спроектированным, и расписание его исправления выходит из-под контроля. Эта проблема, возможно, была у *Netscape*, чьи первые четыре версии каждый раз по мере готовности превращались в такое месиво, что руководство приняло глупое решение выбросить весь имеющийся код и начинать все заново. А потом это руководство сделало такую же ошибку с проектом Mozilla, создав чудовище, вышедшее из-под контроля, и только спустя *несколько лет* этот проект добрался до стадии альфа-версии.

Моя излюбленная теория легко решит эту проблему, приучая программистов охотнее заниматься писательством (для этого их нужно отправить на интенсивные курсы). Другое решение состоит в том, чтобы нанимать способных менеджеров по разработке и сопровождению программ, которые будут писать спецификации. В обоих случаях надо обеспечивать соблюдение простого правила: "Никакого кода без спецификации".

8. Работают ли программисты в тихих условиях?

В литературе уже давно описано увеличение производительности работников умственного труда при предоставлении им достаточного места, причем тихого и уединенного. В книге *Человечески фактор*[3], классической книге по управлению программированием, довольно много говорится об этих способах повысить производительность.

Но есть одна проблема. Мы все знаем, что знающие работники работают лучше всего, попав в "струю", или, как еще говорят, "в колею", где они целиком концентрируются на своей работе и полностью отключены от окружения. Благодаря абсолютной концентрации они не замечают времени и создают прекрасный

[3] Tom DeMarco, Timothy Lister, *Peopleware: Productive Projects and Teams*, Second Edition (New York: Dorset House, 1999); Том Демарко, Тимоти Листер, *Человеческий фактор: успешные проекты и команды*. — (Символ-Плюс, 2005).

продукт. Это при условии, если они получают все нужное для продуктивной работы. Писатели, программисты, ученые и даже игроки-баскетболисты расскажут, что значит "попасть в колею".

Так вот, проблема в том, что "в колею" попасть нелегко. Если попробовать измерить этот процесс, то получится — в среднем требуется 15 минут, чтобы начать работать с максимальной производительностью. Иногда, если вы устали или уже сделали сегодня много творческой работы, то просто не можете попасть в колею и проводите остаток рабочего дня бездельничая, читая Web-страницы или играя в Tetris.

Другая проблема в том, что очень легко оказаться выбитым *из* колеи. Шум, телефонные звонки, перерывы на ланч, необходимость съездить на пять минут в *Starbucks* за кофе и вмешательство сотрудников — все это и *особенно* вмешательство сотрудников выбивает вас из колеи. Если сотрудник задает вам вопрос, он отвлекает вас на минуту, но в результате вы оказываетесь так сильно выбиты из колеи, что на восстановление работоспособности требуется уже полчаса; ваша общая производительность находится под серьезной угрозой. Если вы находитесь в шумном окружении, например, таком, какое любят создавать в накачанных кофеином фирмах-"дот-комах", когда рядом с программистами ребята-маркетологи что-то орут по телефону, производительность у вас падает, так как работников умственного труда периодически прерывают, и они не могут войти в колею.

Это особенно тяжело по отношению к программистам. Производительность зависит от способности жонглировать в кратковременной памяти сразу множеством мелких деталей. Любое вмешательство может привести к тому, что эти детали окажутся рассыпанными. Возобновив работу, вы не сможете вспомнить какую-то из них (например, используемые имена локальных переменных, или на каком месте вы остановились, когда реализовали алгоритм поиска). Поэтому вам придется снова просмотреть эти детали, что значительно замедлит работу, пока вы снова не наберете скорость.

Вот вам простая арифметика. Скажем, если мы прервем программиста даже на минуту, то (как явно подсказывает практика) мы заберем у него 15 минут производительности. Возьмем, к примеру, двух программистов, Джеффа и Матта,

которые сидят по соседству друг с другом в открытых каморках какой-то стандартной фирмы по откорму скота на убой, изображаемой в карикатурах про Дилберта. Матт не может вспомнить имя Unicode-версии функции `strcpy`. Он может поискать ее в справочнике, что займет 30 секунд, или спросить Джеффа, на что уйдет 15 секунд. Матт сидит рядом с Джеффом, поэтому и задает вопрос соседу. Джефф выбит из колеи и теряет на восстановление работоспособности 15 минут (сэкономив 15 секунд для Матта).

Теперь рассадим их по отдельным кабинетам, огороженными стенами и дверями. И когда Матт не сможет вспомнить имя той функции, он поищет ее в справочнике, на что попрежнему уйдет 30 секунд, или спросит Джеффа, на что теперь уйдет 45 секунд, да еще придется встать (а это нелегко, учитывая среднюю физическую подготовку программистов!). Поэтому он поищет функцию в справочнике. Итак, сейчас Матт теряет 30 секунд производительности, зато экономит 15 минут для Джеффа. Вот так!

9. Используете ли вы лучшие средства, какие только можно купить?

Компиляция кода — одна из последних вещей, которые еще нельзя делать мгновенно на домашнем компьютере. Если процесс компиляции занимает больше, чем несколько секунд, то вы сэкономите время, если приобретете новейший и сильнейший компьютер. Если даже компиляция занимает 15 секунд, то, пока работает компилятор, программисты будут скучать и начнут читать журнал *Onion*, на что в целом уйдет несколько часов рабочего времени.

Отладка GUI-кода с помощью системы с одним монитором — процесс болезненный, а то и невозможный. Если вы пишете GUI-код, то два монитора значительно облегчат вам жизнь.

Большинству программистов в конце концов надо работать с растровыми изображениями, чтобы создавать пиктограммы или панели инструментов, и большинство про-

граммистов не имеют хорошего растрового редактора. Обработка растровых файлов с помощью Microsoft Paint — это несерьезно, но именно так приходится действовать большинству программистов.

На моей предыдущей работе системный администратор все время посылал мне спам, жалуясь на то, что я использую на сервере больше (обратите внимание) 220 Мбайт жесткого диска. Я подчеркнул, что такова нынешняя цена использования жестких дисков и что стоимость этого пространства значительно меньше стоимости *туалетной бумаги*, которую я использую. А трата даже 10 минут на чистку своего каталога может обернуться невероятной потерей производительности.

Выдающиеся команды не мучают своих разработчиков. Даже мелкие расстройства, вызванные применением маломощного оборудования, вносят свой вклад, делая программистов раздражительными и несчастными. А раздражительный программист — непродуктивный программист.

Кроме всего этого ... программиста легко подкупить, предложив самую крутую и самую новую программу. Этот способ заставить их работать намного дешевле, чем платить им по-настоящему конкурентоспособные зарплаты!

10. Имеются ли у вас тестеры?

Если в вашей команде нет специальных тестеров, хотя бы по одному на каждых двух-трех программистов, то вы или поставляете продукты с ошибками, или тратите впустую деньги, платя программистам по 100 долл. в час за работу, которую тестер сделал бы за 30 долл. в час. Экономия на тестерах является настолько фальшивой, что меня поражает, почему так мало людей этого не замечают.

11. Пишут ли кандидаты на работу код во время собеседования?

Наняли бы вы волшебника, не попросив его показать какие-нибудь волшебные трюки? Конечно же, нет.

Наняли бы вы поставщика провизии для вашей свадьбы, не попробовав его блюд? Сомневаюсь в этом. (Если только это не тетя Мардж, и она возненавидит вас *навеки*, если ей не разрешат приготовить ее "знаменитый" пирог с рубленой печенью.)

Тем не менее программистов нанимают из-за их впечатляющих резюме или благодаря тому, что ведущему собеседование нравилось с ними болтать. Либо им задают тривиальные вопросы ("В чем разница между `CreateDialog()` и `DialogBox()`?"), на которые можно ответить, посмотрев документацию. Но вас не интересует, помнят ли они тысячи тривиальных сведений о программировании, а интересует, могут ли они создавать код. Или, что еще хуже, им задают "Ага!"-вопросы — такие, которые кажутся легкими, если ответ вам известен, но когда вы его не знаете, то ответить на них нельзя.

Пожалуйста, *прекратите это делать*, только и всего. А в остальном делайте на собеседованиях все, что хотите, но, кроме того, заставьте кандидата *написать код*.

12. Проводите ли вы коридорное тестирование удобства использования программы?

Коридорное тестирование проводится так: вы перехватываете человека, идущего по коридору, и заставляете его использовать код, который вы только что написали. Проделав это с пятью людьми, вы узнаете 95% того, что вам нужно знать о проблемах с удобством использования в вашем коде.

Проектировать хороший пользовательский интерфейс не так уж трудно, как вы, возможно, думаете, зато очень важно заставить пользователей любить и покупать ваш продукт.

Однако самое важное в пользовательских интерфейсах — если вы покажете свою программу горстке людей (на самом деле, пяти или шести будет достаточно), то быстро узнаете самые главные проблемы, с которыми столкнутся

пользователи. Даже не имея квалификации разработчика таких интерфейсов, вы все равно сумеете во много раз улучшить свой пользовательский интерфейс, если заставите себя проводить коридорные тесты на удобство, а они не будут вам ничего стоить.

Предметный указатель

Научно-популярное издание

Джоэл Спольски

Умные и ответственные: как привлечь и удержать хороших IT-специалистов

Литературный редактор *Л.Н. Важенина*
Верстка *О.В. Романенко*
Художественные редакторы *Е.П. Дынник, Т.А. Тараброва*
Корректор *Л.А. Гордиенко*

Издательский дом "Вильямс"
127055, г. Москва, ул. Лесная, д. 43, стр. 1

Подписано в печать 11.12.2007. Формат 84x108/32.
Гарнитура Petersburg. Печать офсетная.
Усл. печ. л. 7,56. Уч.-изд. л. 5,78.
Тираж 2000 экз. Заказ № 0000.

Отпечатано по технологии CtP
в ОАО "Печатный двор" им. А. М. Горького
197110, Санкт-Петербург, Чкаловский пр., 15

ПЕРЕЛОМНЫЙ МОМЕНТ
КАК НЕЗНАЧИТЕЛЬНЫЕ ИЗМЕНЕНИЯ ПРИВОДЯТ К ГЛОБАЛЬНЫМ ПЕРЕМЕНАМ

Малкольм Гладуэлл

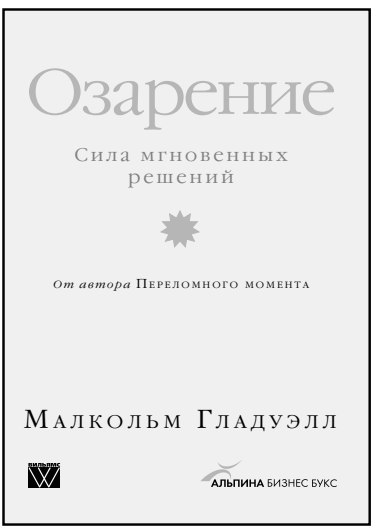
www.williamspublishing.com

Книга *Переломный момент* посвящена механизмам возникновения и развития социальных эпидемий. Автор исследует феномены молвы, молодежной моды, преступности, эпидемий самоубийств и анализирует факторы, лежащие в их основе. Социальные эпидемии начинаются вследствие накопления незначительных изменений, каждое из которых в отдельности может быть проигнорировано, но вместе они приводят к переломному моменту — критической точке, с которой начинается неконтролируемое развитие событий. В работе сформулированы три правила эпидемий, выделены законы, ответственные за развитие эпидемий, а также описаны типажи личностей переломного момента, выступающие катализаторами масштабных социальных перемен.
Книга будет интересна психологам, социологам, социальным работникам, вообще всем специалистам, имеющим дело с человеческим фактором, а также всем пытливым читателям.

ISBN 978-5-8459-1276-3 **в продаже**

ОЗАРЕНИЕ
СИЛА МГНОВЕННЫХ РЕШЕНИЙ

Малкольм Гладуэлл



www.williamspublishing.com

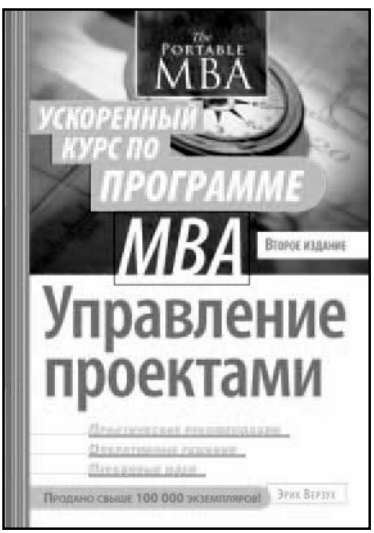
Полицейские расстреляли невинного человека. Специалисты за год исследований не смогли установить поддельность статуи. На пост президента США в 1921 году был избран Уоррен Хардинг — посредственный и незадачливый политик. Почему произошли эти роковые ошибки? Можно ли было избежать их? В своей увлекательной книге *Озарение* Малкольм Гладуэлл, автор бестселлера *Переломный момент*, анализирует процесс принятия решений. На богатом материале из области искусства, науки, дизайна, медицины, политики и бизнеса он раскрывает закономерности бессознательных решений и анализирует факторы, искажающие этот процесс. Книга будет интересна психологам, политологам, маркетологам — всем специалистам, успешная деятельность которых зависит от умения принимать важные решения (подчас в условиях острого дефицита времени), а также широкому кругу читателей.

ISBN 978-5-8459-1273-2

в продаже

УПРАВЛЕНИЕ ПРОЕКТАМИ
УСКОРЕННЫЙ КУРС ПО ПРОГРАММЕ MBA
Второе издание

Эрик Верзух



www.dialektika.com

Цель этой книги — представить реалистичный взгляд на требования, предъявляемые проектной средой, и мастерство, которое необходимо для успешного выполнения проекта. Читая ее, вы познакомитесь с инструментами, которые помогут вам овладеть всеми важнейшими составляющими успеха. В книге изложены глобальные концепции, увязывающие управление проектами с другими управленческими функциями, такими как качество и разработка продуктов. Кроме того, здесь максимально подробно представлены инструменты и методы, т.е. практическая наука — управление проектами. Описание этих методов подается в виде простых примеров, затем следуют рекомендации по управлению более крупными проектами. В книге изложены самые эффективные методы управления бюджетами, мониторинга проекта и соблюдения расписания реализации проекта, которые необходим каждому руководителю проекта.

ISBN 978-5-8459-1106-3

в продаже