БИБЛИОТЕКА ПРОГРАММИСТА

Чед Фаулер

# программист-ФАНАТИК

ПИТЕР®

Chad Fowler

# Passionate Programmer

2nd edition


Чед Фаулер

# программист-ФАНАТИК

ПИТЕР®
Санкт-Петербург • Москва • Екатеринбург • Воронеж
Нижний Новгород • Ростов-на-Дону • Самара • Минск
2019

*Ч. Фаулер*

# Программист-фанатик

*Серия «Библиотека программиста»*

Перевела с английского *И. Рузмайкина*


| | |
|---|---|
| Заведующий редакцией | *П. Щеголев* |
| Ведущий редактор | *Ю. Сергиенко* |
| Литературный редактор | *А. Жданов* |
| Художник | *С. Заматевская* |
| Корректоры | *М. Одинокова, Н. Сидорова* |
| Верстка | *Л. Соловьева* |

ББК 32.973.2-018
УДК 004.43

**Фаулер Ч.**

Ф28 Программист-фанатик. — СПб.: Питер, 2019. — 208 с.: ил. — (Серия «Библиотека программиста»).

ISBN 978-5-4461-0846-6



В этой книге вы не найдете описания конкретных технологий, алгоритмов и языков программирования — ценность ее не в этом. Она представляет собой сборник практических советов и рекомендаций, касающихся ситуаций, с которыми порой сталкивается любой разработчик: отсутствие мотивации, выбор приоритетов, психология программирования, отношения с руководством и коллегами и многие другие. Подобные знания обычно приходят лишь в результате многолетнего опыта реальной работы. По большому счету перед вами — ярко и увлекательно написанное руководство, которое поможет быстро сделать карьеру в индустрии разработки ПО любому, кто поставил себе такую цель. Конечно, опытные программисты могут найти некоторые идеи автора достаточно очевидными, но и для таких найдутся темы, которые позволят пересмотреть устоявшиеся взгляды и выйти на новый уровень мастерства. Для тех же, кто только в самом начале своего пути как разработчика, чтение данной книги, несомненно, откроет широчайшие перспективы.



Издательство выражает благодарность Шувалову А. В. и Курышеву А. И. за помощь в работе над книгой.


**16+** (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ)


ISBN 978-1934356340 англ.
ISBN 978-5-4461-0846-6





Изготовлено в России. Изготовитель: ООО «Прогресс книга».
Место нахождения и фактический адрес: 194044, Россия, г. Санкт-Петербург,
Б. Сампсониевский пр., д. 29А, пом. 52. Тел.: +78127037373.
Дата изготовления: 08.2018. Наименование: книжная продукция.
Срок годности: не ограничен.
Налоговая льгота — общероссийский классификатор продукции ОК 034-2014,
58.11.12.000 — Книги печатные профессиональные, технические и научные.
Импортер в Беларусь: ООО «ПИТЕР М», 220020, РБ, г. Минск, ул. Тимирязева,
д. 121/3, к. 214, тел./факс: 208 80 01.
Подписано в печать 31.07.18. Формат 70x100/16. Бумага офсетная. Усл. п. л. 16,770.
Доп. тираж 500. Заказ 0000
Отпечатано в ОАО «Первая Образцовая типография». Филиал «Чеховский Печатный Двор».
142300, Московская область, г. Чехов, ул. Полиграфистов, 1.
Сайт: www.chpk.ru. E-mail: marketing@chpk.ru
Факс: 8(496) 726-54-10, телефон: (495) 988-63-87

# Содержание

*Для Келли Джин*

# ПРЕДИСЛОВИЕ

Я уверен, что в каждом из нас есть что-то незаурядное, но масса времени уходит на то, чтобы понять, что же на самом деле важно, на то, чтобы вытянуть это из самого себя. Ты не сможешь стать незаурядным, если не любишь свое окружение, свои инструменты, свою область деятельности.

До того как я зажегся проектами 37signals и Ruby on Rails, я прошел через множество работ и заданий, которые вовсе не соответствовали определению «незаурядный». Я торговал водой, и дни были похожи друг на друга, как две капли. Прежде чем я это осознал, прошло 6 месяцев, и они не дали мне абсолютно ничего. Чувство горечи было отвратительным.

Просто ненавижу ощущение, что мое существование не имеет ровным счетом никакого значения, и если я не сделаю свою работу, мир не изменится ни на йоту. Чтобы стать незаурядным, ты должен осознавать, что оставляешь существенный след во Вселенной.

И когда я такого следа не оставлял, это сказывалось и на моей личной жизни. Если я не чувствовал куража во время работы в офисе, становилось все труднее преодолевать себя и начать действовать.

Для меня незаурядная карьера — это оптимальный путь к незаурядной жизни, такой, в которой ты не просто станешь лучшим и более востребованным работником, но и станешь лучше как человек.

Именно поэтому так важна эта книга. Она посвящена не только тому, как делать более качественные виджеты и чувствовать себя на работе защищенным. Она о том, как развить в себе навыки и способности, делающие жизнь комфортнее и наполняющие ее множеством незаурядных вещей, из которых работа — лишь одна из составляющих.

*Дэвид Хэйнемер Хэнсон (David Heinemeier Hansson),*
*создатель проекта Ruby on Rails и партнер проекта 37signals*

# БЛАГОДАРНОСТИ

Я бы никогда не написал эту книгу, если бы не Дэйв Томас и Энди Хант. Их книга «Программист-прагматик. От подмастерья к мастеру» (The Pragmatic Programmer: From Journeyman to Master) стала катализатором и вдохновляющей силой. Если бы не поддержка и руководство Дэйва, я бы так и считал себя недостаточно квалифицированным для написания этой книги.

Сюзанна Пфальцер (Susannah Pfalzer) редактировала второе издание книги. Под редактированием я подразумеваю подталкивание, воодушевление, вдохновение, руководство и, конечно… собственно редактирование. Ее терпение и умение найти правильные слова, мотивируя меня, а не пугая, — это как раз то, что было так необходимо для завершения работы. Если бы не Сюзанна, книга так и осталась бы набором сумбурных, не до конца сформулированных идей.

Хочу поблагодарить Дэвида Хэйнемера Хэнсона (David Heinemeier Hansson), написавшего предисловие. Его карьера в 37signals и Rails — это блестящий пример воплощения идей, лежащих в основе этой книги. Я рад, что в мой труд внесли вклад и другие незаурядные люди, с которыми я сталкивался на протяжении своей карьеры. Огромное спасибо Стефену Акерсу (Stephen Akers), Джеймсу Дункану Дэвидсону (James Duncan Davidson), Вику Чадха (Vik Chadha), Майку Кларку (Mike Clark), Патрику Коллисону (Patrick Collison) и Тому Престену-Вернеру (Tom Preston-Werner), которые вдохновляли и меня, и читателей.

Спасибо рецензентам за ценные замечания, которые помогли в подготовке второго издания. Всегда удивительно, насколько неправильна может быть исходная версия главы и насколько правильной ее может сделать хороший рецензент. Спасибо Сэмми Лэрби (Sammy Larbi), Брайну Дайку (Bryan Dyck), Бобу Мартину (Bob Martin), Кенту Беку (Kent Beck), Алану Фрэнсису (Alan Francis), Джареду Ричардсону (Jared Richardson), Ричу Доуни (Rich Downie) и Эрику Костнеру (Erik Kastner).

Не могу не упомянуть Джульет Томас (Juliet Thomas), редактировавшую первое издание книги. Ее энтузиазм и ви́дение перспектив были неоценимы. Я получил огромное количество отзывов от рецензентов первого издания: Кэри Боаз (Carey Boaz), Карла Брофей (Karl Brophey), Брэндона Кэмбэла (Brandon Campbell), Вика Чадха (Vik Chadha), Мауро Чичио (Mauro Cicio), Марка Донохью (Mark Donoghue), Пэта Эйлера (Pat Eyler), Бэна Гудвина (Ben Goodwin), Якоба Харриса (Jacob Harris), Адама Кейса (Adam Keys), Стива Морриса (Steve Morris), Билла Налла (Bill Nall), Уэсли Рейза (Wesley Reiz), Авика Сенгупта (Avik Sengupta), Кента Спиллнера (Kent Spillner), Сандеша Таттитали (Sandesh Tattitali), Крэйга Утли (Craig Utley), Грега Вона (Greg Vaughn) и Питера У. А. Вуда (Peter W. A. Wood). Они помогли сделать книгу значительно лучше, и я никогда не смогу отблагодарить их за потраченное время, силы и проявленное понимание.

Спасибо всем прекрасным людям, с которыми я работал как официально, так и не официально, за идеи, легшие в основу этой книги. Спасибо за то, что выслушали, научили и просто общались, Донни Уэббу (Donnie Webb), Кену Смиту (Ken Smith), Уолтеру Хоэ (Walter Hoehn), Джеймсу Макмюрри (James McMurry), Кэри Боаз, Дэвиду Алану Блэку (David Alan Black), Майку Кларку, Николь Кларк (Nicole Clark), Вику Чадха, Ави Брайнт (Avi Bryant), Ричу Килмеру (Rich Kilmer), Стиву Акерсу (Steve Akers), Марку Гарднеру (Mark Gardener), Райну Оуненсу (Ryan Ownens), Тому Копелэнду (Tom Copeland), Дэйву Крэйну (Dave Craine), Джону Афайду (John Athayde), Марселю Молина (Marcel Molina), Эрику Костнеру (Erik Kastner), Брюсу Уильямсу (Bruce Williams), Дэвиду Хэйнемеру Хэнсону (David Heinemeier Hansson), Али Сареа (Ali Sareea) и Джиму Уэричу (Jim Weirich).

Спасибо моим родителям за их постоянную поддержку. И, самое главное, — спасибо моей жене Келли за то, что наполняет мою жизнь смыслом.

# ВВЕДЕНИЕ

Книга посвящена тому, как реализовать себя и сделать карьеру. Самореализация и удача редко приходят случайно. Они требуют вдумчивости, целеустремленности, действия и готовности резко сменить курс, если потребуется. В этой книге описывается стратегия, позволяющая спланировать и реализовать совершенную с точки зрения успеха карьеру (и как следствие — жизнь) разработчика программного обеспечения.

Книга о том, как взрастить в себе желание быть незаурядным. Удивительно, но когда мы начинаем строить карьеру, далеко не в каждом живет стремление стать незаурядным. Большинство из нас просто плывет по течению. Причем СМИ, друзья, знакомые и родные лишь занижают наши ожидания. Поэтому, чтобы сделать свою жизнь незаурядной, ты должен поставить себе цель, а это совсем не очевидно.

Большинство взрослых людей подавляющую часть своего времени проводят на работе, а по статистике[1] за 2006 г., средний американец провел на работе половину жизни. Отдых и занятия спортом далеко позади и занимают лишь 15 % времени бодрствования. Факты говорят о том, что наша жизнь в основном *проходит* на работе.

Если бóльшая часть жизни поглощена работой, то любовь к работе — один из важнейших способов возлюбить собственную *жизнь*. Интересная, мотивирующая и достойно оплачиваемая работа будит тебя по утрам гораздо лучше, чем скучные и тривиальные обязанности. Если ты хорошо работаешь, значит, 50 % времени ты занят тем, в чем ты действительно хорош. И наоборот, если ты работаешь плохо, то бóльшую часть времени ты чувствуешь себя некомпетентным или виновным в том, что не способен сделать должным образом.

Все мы ищем счастья. По крайней мере если наши основные потребности в пище и жилище удовлетворены, то бóльшая часть наших устремлений направлена на то, чтобы быть счастливым. К сожалению,

---

[1] http://www.bls.gov/tus/charts/

часто наши действия *не соответствуют* этой наиважнейшей цели. Так происходит из-за того, что все мы люди, а людям свойственно забывать о конечной цели, зацикливаясь на средствах ее достижения.

Был бы я счастливее, имея больше денег? Был бы я счастливее, если бы мои достижения больше ценились? Был бы я счастливее, если бы меня повысили? А если бы я стал знаменит? А если бы я был беден и имел самую обычную работу, то мог бы я быть счастлив? Возможно ли это? И если да, то стоит ли тогда гнаться за деньгами или лучшей работой?

Может быть, да, а может, и нет. Но что действительно истинно, так это то, что, сконцентрировав свои усилия на счастье как главной цели, мы сможем гораздо лучше принимать решения о том, какие именно небольшие шаги необходимо сделать для достижения этой цели. Более высокая зарплата действительно может быть желанной и вести к счастью. Но если ты на миг забудешь о главной цели, это может привести к тому, что ты начнешь добиваться более высокой зарплаты *за счет* собственного счастья. Это звучит дико, но со мной такое случалось. И с тобой, возможно, тоже. Подумай над этим.

На протяжении этой книги я собираюсь давать тебе полезные советы, которые, надеюсь, приведут тебя к более счастливой и стоящей карьере (и, как следствие, к более счастливой жизни). Возможно, следуя этим советам, ты станешь больше зарабатывать. Возможно, ты станешь более узнаваемым или даже известным. Но, пожалуйста, помни, что это — не цель. Это лишь средства для ее (цели) достижения.

## Не ведая преград!

По иронии судьбы одним из важнейших этапов на пути построения незаурядной карьеры для меня стало первое издание этой книги. Она называлась «Моя работа досталась индусам (а все, что получил я, — эта жалкая книжонка), или 52 способа сохранить работу» (My Job Went to India (And All I Got Was This Lousy Book): 52 Ways to Save Your Job). На обложке был изображен парень с табличкой «Код за еду». Это было прикольно, а название и шокирующая красная обложка обыгрывали страх Западного мира, что всю работу захватят гастарбайтеры из развивающихся стран.

Однако все это создавало неправильный образ. Если ты всего лишь хочешь «сохранить» работу, тут я тебе не помощник. Эта книга не о том, как добиться некоего среднего уровня развития, достаточного, чтобы тебя не уволили. Эта книга о том, как стать *крутым*. Как стать *победителем*. Ты не выиграешь гонку, если просто будешь стараться не проиграть. И ты не выиграешь в жизни, думая о том, как не облажаться. Эта книга не о том, как не облажаться. Мне даже думать об этом не хочется, и тебе, надеюсь, тоже.

Я отлично помню момент, когда решил, что моя карьера должна стать незаурядной. Мои первые рабочие места стали продолжением подростковых увлечений — я постепенно двигался к весьма посредственной карьере профессионального саксофониста. Однако благодаря удаче и отчасти природному таланту я умудрился сойти с этого пути, получить высокооплачиваемую работу штатного технического специалиста одной из крупных компаний и стать уважаемым человеком в этой среде. Но это было *только начало*, и я об этом знал.

Как-то вечером после работы я просматривал полки в ближайшем книжном магазине и увидел среди новинок книгу Кента Бека «Экстремальное программирование» (*Extreme Programming Explained. Embrace Change*). Идея любить перемены всегда была мне близка. До того момента у меня всегда были сложности с усидчивостью, я переходил с одной работы на другую, часто меняя работодателей. Хотя описание «методологии разработки ПО» казалось мне невыносимо скучным и отдавало менторством, я решил, что если эта методология поддерживает постоянные перемены, это может помочь мне не скучать и не думать о том, чтобы очередной раз поменять работу.

Покупка этой книги оказалась моей удачей. Начав читать, я уже не мог оторваться, потом я прочел все, что нашел в интернете об экстремальном программировании (Extreme Programming, XP). Эти идеи захватили меня настолько, что я обратился к руководителю информационной службы, пытаясь и его приобщить в своей новой религии. Это мне удалось, и для внедрения экстремального программирования он отправил меня и многих моих коллег на соответствующие курсы.

Это было *великолепно*. Это было похоже на пропуск за кулисы на концерте любимой рок-группы. Пообщавшись с людьми, ведущими эти

курсы, я стал *гораздо* умнее. Стал более креативным. И когда курсы закончились, мне было очень грустно. Я не мог представить, как вернусь в маленький кабинет и буду биться головой о стену обыденности, которую вырастил у себя на работе.

Мой коллега Стив, автор одного из эссе, вошедших в эту книгу, и я пришли к одинаковому выводу. Единственный путь общаться с такими людьми как можно чаще — стать одним из них. Другими словами, если я хочу оказаться в компании людей, поднимающих меня на один-два уровня выше, то проблема не в фирме, в которой я работаю, и не в курсах, которые я посещаю. Я просто должен понять, чем эти люди отличаются от прочих, и постараться стать одним из них. Это я и сказал Стиву.

Именно это *стало* поворотной точкой в моей карьере. Я как-то умудрился об этом забыть, и лишь годы спустя Стив напомнил мне про наш разговор. И тогда я похвастался ему случаем, когда меня впервые пригласили открыть конференцию. Это было просто потрясающе. *Меня* не просто попросили выступить, а предложили стать одним из главных докладчиков на конференции по программным продуктам. Я действительно стал одним из тех людей, которыми восхищался.

Я достиг всего, не имея формального компьютерного образования. До того как стать программистом, я был музыкантом. Я пошел в колледж, чтобы учиться музыке. Так как музыкантам ученая степень не особо требуется, я решил пропускать все занятия, которые не способствовали моей карьере музыканта. Это привело к тому, что я бросил университет, так как у меня было слишком много «хвостов» для получения степени. С этой точки зрения мне не хватало квалификации, чтобы программировать профессионально, по крайней мере если смотреть на типичные требования к программистам на рынке труда.

Однако, несмотря на то что у меня не было профессиональной подготовки обычного разработчика ПО, мой опыт музыканта дал мне возможность пропустить эту ступеньку и не стать обычным программистом (кому охота быть обычным?). *Никто* не становится музыкантом, чтобы вести спокойную и размеренную жизнь. Музыкальная индустрия слишком жестока для этого. *Все* люди, желающие стать профессиональными музыкантами, хотят быть *великими*. По край-

ней мере на начальном этапе в музыкальном мире это стремление бинарно: либо быть великим (и как следствие знаменитым), либо не стоит и соваться.

Я часто задаюсь вопросом, почему так много хороших музыкантов являются также хорошими программистами? Причина проста. Дело не в том, что используются одни и те же функции мозга, обе профессии ориентированы на нюансы, обе требуют креативности. Дело в том, что человек, который хочет стать великим, с гораздо *большей* вероятностью им станет, чем тот, кто просто хочет делать свое дело. И даже если не все могут быть Мартинами Фаулерами, Линусами Торвальдсами или программистами-прагматиками, постановка столь высокой цели делает это более вероятным.

## Ты должен

Большинство людей следует чьим угодно планам, только не своим. Всё, что нужно сделать, чтобы отделить себя от других, — это остановиться и хорошенько присмотреться к своей карьере. Тебе нужно придерживаться *своего* плана, а не чьего-то еще.

Как составить такой план? Разработка программ — это бизнес. Как программисты, мы являемся еще и бизнесменами. Наши компании наняли нас вовсе не потому, что любят нас. Этого никогда не было и не будет. Просто потому, что не имеет отношения к бизнесу. Компании существуют совсем не для того, чтобы нам каждый день было куда пойти. Цель бизнеса — делать деньги. Чтобы преуспеть в компании, ты должен четко представлять себе, как ты вписываешься в план добывания денег.

Как мы увидим позже, сохраняя тебя в штате, компания тратит значительные средства. Она *инвестирует* в тебя. Твоя задача — стать безусловно хорошей инвестицией. Ты станешь судить о своей продуктивности в зависимости от той ценности, которую ты приносишь организации или заказчику, который тебя нанял.

Думай о своей карьере как о жизненном цикле создаваемого тобой продукта. Он рождается благодаря тебе и твоим навыкам. В этой

книге мы рассмотрим четыре грани, на которых должен концентрироваться бизнес при проектировании, разработке и продаже продукта. И увидим, как эти четыре грани проявляются в карьере.

- *Выбери рынок*. Осознанно и осторожно подбери технологии и сферы бизнеса, на которых ты будешь концентрироваться. Как сбалансировать риски и отдачу? Как учесть фактор спроса и предложения?
- *Инвестируй в свой продукт*. Твои знания и навыки — это краеугольный камень всего продукта. Правильные инвестиции в них — ключ к хорошему спросу на рынке труда. Просто знать Visual Basic или Java уже недостаточно. Какие еще навыки могут тебе понадобиться в новых экономических условиях?
- *Действуй*. Простое содержание работников, имеющих крепкие навыки, не приносит компании денег. Их должны *приносить* работники. Как этого добиться, не погрязнув в рутине? Как узнать, что ты представляешь для компании *достаточную* ценность?
- *Продавай*! Даже самый лучший продукт не будет продаваться, если о его существовании никто не знает. Как без заискивания получить признание в компании и в отрасли в целом?

## Новое издание

Это уже второе издание книги «Моя работа досталась индусам (а все, что получил я, — эта жалкая книжонка), или 52 способа сохранить работу» (*My Job Went to India (And All I Got Was This Lousy Book): 52 Ways to Save Your Job*). Цель переиздания — дополнительно сконцентрироваться на основной теме: создании грандиозной карьеры. Для этого я не только придумал более позитивное название, но и изменил содержание.

Дэвид Хэйнемер Хэнсон, создатель Ruby on Rails и партнер в проекте 37signals, написал новое вступительное слово.

В каждую часть я добавил одно (или несколько) эссе, написанных людьми, с которыми я сталкивался или работал, и чьи карьеры по-

настоящему незаурядны. Эти эссе показывают, какие решения принимали новаторы, разработчики, менеджеры и предприниматели на своем пути к успеху. Они подтверждают тот факт, что описываемые в книге подходы — это не просто предположения, применимые лишь в идеальном окружении. Это реальные дела, на которые способны реальные люди.

Некоторые советы были удалены, другие добавлены. Они отражают то, чему я научился с тех пор, как вышла первая версия книги.

К советам, которые были в первом издании, добавлены новые разделы «Действуй!»

Введение и заключение также поменялись, чтобы в полной мере отразить назначение книги и ее акцент на построение грандиозной карьеры.

Назначение книги — показать системный путь построения незаурядной карьеры разработчика программного обеспечения. Мы рассмотрим примеры и опишем действия, которые ты можешь предпринять *прямо сейчас* и которые будут иметь позитивный эффект как в краткосрочном, так и в долгосрочном плане.

И как я уже отмечал, мы не будем говорить о том, как сохранить работу. Если ты боишься потерять работу, то шаги, принимаемые для построения незаурядной карьеры, избавят тебя от этого страха. Незаурядные разработчики не сидят без дела. Они не занимаются бесплодными поисками работы. Поэтому волноваться не надо. Сосредоточься на победе и навсегда забудь о страхе.

# ЧАСТЬ I
# НАЙДИ СВОЙ РЫНОК

Ты собираешься сделать *крупные* инвестиции. Я подразумеваю не большие денежные суммы, а твое время — твою жизнь. Большинство не привыкло прикладывать усилий к построению своей карьеры, предпочитая плыть по течению. Человек изучает язык Java или Visual Basic, и в какой-то момент работодатель оплачивает ему курсы повышения квалификации для знакомства с последними новинками отрасли. И снова начинается рутинная работа, пока не появится очередная направляющая сила. В итоге карьера целиком зависит от стечения обстоятельств.

Дэйв Томас и Энди Хант в книге «Программист-прагматик: путь от подмастерья к мастеру» (*The Pragmatic Programmer: From Journeyman to Master*) рассказывают о *программировании в расчете на совпадения*. Большинство программистов начинают работать над задачей, добавляя небольшие фрагменты кода. Иногда просто берется фрагмент кода с сайта и редактируется под собственные нужды. Принцип работы программы при этом остается непонятым, но в нее вносятся многочисленные коррективы, пока она не начнет соответствовать текущим надобностям. Полученный таким образом продукт напоминает карточный домик, и добавление каждой следующей подсистемы повышает вероятность сбоя.

Любой разработчик программного обеспечения понимает порочность подобного подхода. При этом собственный карьерный рост многие оставляют на волю случая. Каким технологиям следует уделить повышенное внимание? Эксперты в какой области являются самыми востребованными? Что лучше — расширять кругозор или углублять свои знания? Мы просто обязаны задавать себе все эти вопросы.

Представь, что ты создал компанию и работаешь над продуктом, которому суждено стать ее флагманом. Если твое творение не будет пользоваться спросом, компания обанкротится. Насколько внимательно ты подойдешь к выбору целевой аудитории? Как будешь думать о свойствах продукта, перед тем как начать производство? Невозможно представить бизнесмена, который отдаст подобные вопросы на откуп посторонним людям. Деловой человек участвует на всех этапах принятия решения.

Так почему же большинство из нас не столь тщательно относятся к решениям, связанным с карьерным ростом? Если взглянуть на

карьеру как бизнес (а так оно и есть), твоим «продуктом» будут те услуги, которые ты в состоянии оказать.

Что это за услуги? Кому ты собираешься их продавать? Возрастет или сократится спрос на них в будущем? Насколько ты готов рискнуть, делая выбор?

Ответы на все эти важные вопросы поможет найти первая часть книги.

## Совет 1
## Будь впереди или погибнешь?

Существует множество возможностей инвестировать свои деньги. Можно отнести их в банк, но зачастую проценты не покрывают инфляции. Можно вложиться в государственные сберегательные облигации. Много таким способом не заработаешь, хотя какой-то доход наверняка будет.

Еще можно вложиться в молодой стартап. За несколько тысяч долларов у тебя появится небольшая доля. При условии интересной идеи, лежащей в основе нового проекта, и эффективного управления ты можешь получить *хорошую* прибыль. Но, разумеется, даже возврат первоначальных инвестиций тебе не гарантирован.

Ничего нового в этой концепции нет. Осваивать ее мы начинаем еще с детских игр. *Если не убегать, а рвануть в сторону других игроков и попытаться проскочить, все удивятся, и никто меня не поймает.* О ней напоминают события нашей обычной жизни. Когда ты опаздываешь на работу, приходится рисковать в поисках самого быстрого пути. *Если не будет пробок, то, проехав по 32-й улице, я выиграю 15 минут. А если там пробка, мне конец.*

Оценка рисков является необъемлемой частью выбора технологий и областей для будущих инвестиций. Пятнадцать лет назад ты практически ничем не рисковал, решив изучать COBOL. Программистов, пишущих на этом языке, было много, и их зарплаты не поражали своим размером. Найти работу в этой области было легко, но особых

доходов ждать не приходилось. Риск был невелик, но и награда не впечатляла.

Если бы в то время ты обратил внимание на язык Java, разработанный компанией *Sun Microsystems*, тебе пришлось бы поискать фирму, в которой этот язык был бы востребован. Кто знал, будет ли *вообще* нужен Java?

Но если вернуться назад и взглянуть на состояние отрасли в то время, как это сделали в *Sun*, то в языке Java обнаружится кое-что особенное. Можно было понять, что его ждет большое будущее. Инвестиции в эту область позволяли любому стать лидером в новом, подающем большие надежды технологическом направлении.

Правильно разыграв свои карты, ты мог получить высокий доход от инвестиций в язык Java. Высокий риск в полной мере себя оправдывал.

А теперь представь, что 15 лет назад ты участвовал в презентации новой операционной системы BeOS от фирмы *Be*. Это было что-то невероятное. Она создавалась для поддержки многопроцессорной архитектуры. Мультимедийные возможности поражали. Платформа наделала много шума и вскружила голову специалистам, которые ждали появления нового сильного игрока на рынке операционных систем. Должны были родиться новые способы программирования, новые API и новые концепции пользовательского интерфейса. Приходилось многое изучать «с нуля», но, казалось, дело того стоит. Приложив немало усилий, можно было создать первый FTP-клиент или программу-календарь для BeOS. Но сразу после выпуска Intel-совместимой версии операционной системы появились слухи, что *Apple* покупает фирму *Be*, которая собирается использовать разработанные технологии как основу для следующего поколения операционной системы *Macintosh*.

*Apple* не стала покупать *Be*. Более того, стало понятно, что *Be* не собирается захватывать даже узкоспециализированный рынок. Продукт просто не пришелся ко двору. Множество разработчиков, занимавшихся программированием для BeOS-окружения, медленно, но неуклонно понимали, что в долгосрочной перспективе их инвестиции себя не оправдали. В конечном счете *Be* приобрела компания *Palm*,

и работа над операционной системой прекратилась. В данном случае мы имеем пример рискованных, хотя и привлекательных технологических инвестиций, которые не принесли долгосрочной выгоды вкладчикам. Награды за высокий риск не последовало.

Я показал вам разницу между новыми и уже успевшими найти свое место на рынке технологиями. Выбор стабильной, используемой в промышленности по всему миру технологии является более безопасным, но потенциально менее выгодным, чем ставка на только что появившуюся и никем не освоенную технологию. А как насчет технологий, которые себя исчерпали? Ожидающих, когда в крышку их гроба будет вбит последний гвоздь?

Кто забьет этот гвоздь? Можно вспомнить, например, последних оставшихся седовласых и считающих часы до пенсии программистов на языке RPG, о котором молодое поколение *даже не слышало*. Сейчас все увлечены Java и .NET. Легко представить, как карьеры последних приверженцев устаревшей и умирающей технологии движутся по той же самой смертельной спирали, что и сама технология.

Старые системы умирают не сразу. Их замещают другие технологии. В большинстве случаев это многоэтапный процесс. И пока он протекает, старые системы должны взаимодействовать с новыми. Нужен человек, знающий, как склеить старое и новое. Молодежь, как правило, не знает (или не хочет знать), с какой стороны подойти к старой системе. А закосневшие в своей любви к старым системам специалисты пенсионного возраста не имеют представления о способах перехода к новым технологиям.

Расчетливый технический специалист в подобных случаях может взять на себя роль посредника. Помочь старой системе спокойно и с достоинством уйти на покой — задача, важность которой нельзя недооценить. Разумеется, большинство специалистов постарается — выйдя на пенсию или переключившись на другую технологию — убежать с этого корабля, пока он окончательно не утонул. Решив до последнего поддерживать все еще нужную систему, ты сможешь диктовать свои условия. Это риско-

Доходным может оказаться любой из концов кривой восприятия технологий.

ванно, ведь после окончательной смерти технологии ты окажешься экспертом в не существующей более области. Но при умении быстро переключаться ничто не мешает найти следующее поколение устаревших систем и начать все заново.

Кривая восприятия технологий имеет два конца. Насколько близко к ним ты готов подойти?

**Действуй!**

1. Составь список новых, самых популярных и угасающих технологий. Перенеси их на лист бумаги: слева должны оказаться только что заявившие о себе направления, а справа — идущие к своему логическому концу. Постарайся вспомнить как можно больше примеров для каждой части спектра. Тщательно оценивай их положение друг относительно друга.

   Когда схема будет готова, пометь те технологии, с которыми ты знаком наиболее полно. Затем выдели другим цветом сферы, в которых у тебя есть некий опыт, но недостаточный, чтобы считаться квалифицированным специалистом. В каком месте на кривой восприятия окажется наибольшее количество пометок? Образуют ли они группу? А может быть, равномерно распределены по всей длине? Есть ли среди граничных технологий интересующие тебя особо сильно?

## Совет 2
## Предложение и спрос

Когда интернет только начал входить в нашу жизнь, можно было хорошо заработать на создании простых HTML-страниц для фирм. Буквально каждый жаждал иметь собственную страничку, а вот создавать их умели немногие. Компании были готовы платить большие деньги «опытным» веб-дизайнерам, которые знали лишь основы HTML, процедуру связывания посредством гиперссылок и имели общее представление о том, что такое структура сайта.

Создание HTML-страниц — процесс несложный. Трудно создавать красивые страницы, а основы освоить нетрудно. Как только выяс-

нилось, сколько веб-дизайнеры запрашивают за свою работу, многие приобрели книги по HTML и занялись самообразованием. Спрос рос, зарплаты и почасовая оплата были крайне привлекательными, и ответом на это стал рост количества специалистов по HTML.

По мере наполнения рынка веб-дизайнерами началось расслоение на настоящих художников и приверженцев утилитарного подхода. Кроме того, конкуренция привела к снижению цен. В итоге возросло количество фирм, желающих быть представленными во Всемирной паутине. Те, кто не мог заплатить за первый сайт $5000, получили его за $500.

Разумеется, остались компании, готовые платить огромные деньги за *фантастический* сайт. А значит, некоторые веб-дизайнеры все еще могли рассчитывать на *фантастическое* вознаграждение.

В конечном счете поток веб-дизайнеров в нижнем и среднем ценовых сегментах иссяк. Наименее талантливых вытеснили пользователи и люди, имеющие отношение к IT, но не специализирующиеся на HTML-дизайне. Наступило равновесие между предложением, спросом и ценой на написание HTML-кода.

Эволюция связанной с веб-дизайном профессии демонстрирует знакомую нам всем экономическую модель *спроса и предложения*. Большинство при этих словах подумает о цене, которую можно запросить за товар и за которую он в итоге будет продан. Если товара больше, чем покупателей, его цена начнет падать. Если людей, жаждущих приобрести определенный товар, больше, чем количество выставленных на продажу единиц товара, цена будет расти, так как между покупателями возникнет конкуренция.

Модель спроса и предложения предсказывает не только цену на товары и услуги, но и влияние изменившейся цены на количество желающих продать и приобрести некий товар или услугу. Обычно на дешевый товар находится куда больше покупателей, чем на дорогой.

Почему нам это важно? Мода на создание программного обеспечения в других странах привела к появлению на нашем рынке большого количества дешевых IT-специалистов. В то время как мы на внутреннем рынке беспокоимся о потере работы, на самом деле снижение затрат на программистов *повышает* общий спрос. А одновременно с ростом спроса падает цена. Конкуренция в сфере пользующихся

высоким спросом товаров и услуг зависит от цены. На рынке труда ценой является заработная плата. Здесь ты не конкурент. Ты просто не можешь себе этого позволить. Так как же быть?

Низкооплачиваемые программисты из стран третьего мира работают с относительно небольшим количеством технологий. В Индии полным-полно людей, пишущих на Java и работающих с .NET. Там множество специалистов по базам данных Oracle. Экспертов в области менее популярных технологий в этой группе намного меньше. Выбирая направление развития, учитывайте перспективы роста предложений в этой сфере и, как следствие, снижение зарплат.

> Конкурировать, предлагая более низкую цену на свои услуги, ты не можешь. Точнее, ты не можешь себе этого позволить.

Конкуренция среди программистов, работающих с платформой .NET, на десятки тысяч человек выше, чем среди тех, кто пишет программы на Python. В результате средняя цена такого специалиста сильно падает, что потенциально может привести к повышению спроса (то есть к созданию большего количества рабочих мест в этой области). В итоге у тебя не будет проблем с трудоустройством, но вряд ли зарплата будет тебя устраивать. Программистов на языке Python намного меньше, чем программистов для .NET, но и спрос на них невелик. Но если на рынке труда такой специалист будет стоить заметных денег, это привлечет людей, которые начнут *предлагать* эту услугу. Конкуренция повысится, и цена начнет падать.

В какой-то момент наступает баланс между спросом и предложением. Но пока с уверенностью можно утверждать только одно. Среднестатистические индийские фирмы никогда не занимаются нестандартными технологиями. Они не первопроходцы и, как правило, не склонны к риску. Они дожидаются стабильной ситуации на рынке сервисных услуг, а затем дестабилизируют этот рынок относительно дешевой рабочей силой.

Исходя из этих наблюдений имеет смысл выбрать для приложения своих усилий область рынка труда, спрос на которую невысок. Это может прозвучать абсурдно, но если тебя беспокоит заполнение рабочих мест иностранной рабочей силой, достаточно избегать об-

ластей, на которых специализируются офшорные компании. Они выполняют работы, пользующиеся высоким спросом. Значит, сконцентрировавшись на узкоспециализированных технологиях, ты пусть и не облегчишь себе конкурентную борьбу (так как предлагаемых вариантов работы в этом случае куда меньше), но будешь соперничать уже не в расценках, а в умении. Именно это и требуется. Ты не способен конкурировать в расценках, но *в состоянии* соревноваться в компетентности.

Но не стоит упускать из виду, что снижение *средней* стоимости среднестатистического программиста приводит к повышению спроса. В свою очередь, возросший общий спрос, к примеру, на Java-программистов, может вызвать увеличение, а не уменьшение количества рабочих мест определенного типа в твоей собственной стране. Рост числа дешевых иностранных специалистов в состоянии повлиять на рынок в целом, в том числе — на разработки в более высокой ценовой категории.

На практике так и происходит. Многие компании обнаруживают, что для нормального функционирования иностранных филиалов требуются высококлассные специалисты на местах, которые будут задавать стандарты, гарантировать качество и обеспечивать ведущее положение технологии. Повысившийся спрос на Java-программистов естественным образом привел к росту спроса на подобных сотрудников. Работы нижнего ценового сегмента могут передаваться офшорным фирмам, но именно благодаря этому растет и количество высокооплачиваемых рабочих мест. И так же, как на узкоспециализированном рынке труда, фактором конкуренции в области Java-разработок становится не цена, а квалификация.

Рассмотренная в этом разделе экономическая модель позволяет понять важный факт: повышение спроса увеличивает конкуренцию по цене. Проверенная временем стратегия следования за рынком заставит тебя конкурировать по цене с иностранными разработчиками, потому что твои навыки будут соответствовать именно этому сегменту рынка. Для успешной конкуренции в области устоявшейся технологии нужен переход на более высокий уровень. В качестве альтернати-

Используй дисбаланс рынка труда.

вы можно обратить внимание на *нестабильные* сферы, не интересные офшорным компаниям. В любом случае, важно понимать расстановку сил и корректно реагировать на ее изменение.

**Действуй!**

1. Исследуй рынок труда. Какие технические навыки наиболее или наименее востребованы? В этом тебе помогут рекрутинговые сайты. Найди несколько сайтов офшорных компаний (или поговори с теми, кто работает в таких компаниях). Сравни навыки, которые требуются для работы в такой компании, с навыками, наиболее востребованными на рынке труда. Выдели те из них, которые пользуются более высоким спросом внутри страны и практически не представлены среди иностранной рабочей силы.

   Таким же образом проанализируй ведущие технологии и квалификацию специалистов в офшорных фирмах. Обрати внимание на области, не занятые иностранными разработчиками. Сколько времени потребуется для их появления на этом рынке (если такое вообще произойдет)? Именно за этот промежуток и произойдет стабилизация рынка.

## Совет 3
## Умения писать код мало

Недостаточно выбрать технологию, на которую стоит сделать ставку. В конце концов, разве знание технологии — не товар, который нужно продать? Ты не сможешь расслабленно совершенствовать навыки программирования, оставив связанные с бизнесом аспекты специалистам. Очень легко найти человека, способного писать код. Но если ты хочешь стать незаменимым, придется детально разобраться в особенностях бизнеса, с которым связана твоя деятельность.

По сути дела, технический специалист должен понимать принцип ведения бизнеса на куда более глубоком уровне, чем требуется для разработки программного обеспечения. На предыдущем месте работы мне довелось столкнуться с подобными специалистами. Команда, отвечающая за администрирование баз данных, состояла из людей, которых не слишком интересовали технологии баз данных как тако-

вые. При первой встрече с ними я испытал недоумение. *Почему эти люди работают в области информационных технологий?* Хотя их техническая квалификация оставляла желать лучшего, это была особенная команда. Они не только хранили и защищали данные нашего предприятия, но и разбирались в его хозяйственно-экономической деятельности лучше любого из наших экономистов, занимающихся вопросами конъюнктуры. Именно знание и понимание всех аспектов бизнеса обеспечили им повышенный спрос на рынке труда. В то время как мы, компьютерные гики[1], смотрели на них свысока, *коммерческая структура*, для которой они работали, оценила их по достоинству.

Умение разобраться в особенностях бизнеса должно стать неотъемлемой частью твоих навыков. Например, музыканту, который добавляет в репертуар новое произведение, недостаточно один раз его сыграть. Его следует по-настоящему *выучить*. Именно так следует воспринимать знание бизнес-сферы. Ты можешь поработать в отрасли медицинского страхования, но так и не понять, в чем состоит разница между EDI-транзакциями HIPAA 835 и HIPAA 837. А ведь именно знание подобных тонкостей выделило бы тебя из числа разработчиков программного обеспечения, квалификация которых аналогична твоей.

Можно быть «обычным программистом», но умеющим говорить с деловыми клиентами на их языке. Ты только представь, насколько упростилась бы твоя жизнь, если бы все, с кем тебе приходится иметь дело, понимали принципы разработки программного обеспечения. Не приходилось бы тратить время на объяснения, почему не стоит выводить по 30 000 записей на одной странице веб-приложения или почему ссылки не должны вести на сервер разработки. Клиенты испытывают по отношению к тебе совершенно аналогичные чувства. *Им хотелось бы работать с программистами, которые сразу понимают, что от них хотят, без объяснений на пальцах и абсурдного рассмотрения мельчайших деталей!* А ведь именно они платят тебе деньги.

Направления коммерческой деятельности, как и технические дисциплины, имеют разную популярность. Например, в разработке про-

---

[1] Гик (англ. geek) — человек, фанатично увлеченный чем-либо, чаще всего компьютерными технологиями. — *Примеч. ред.*

граммного обеспечения в настоящий момент лидируют Java и .NET. Изучив эти технологии, ты сможешь претендовать на работу в одной из многочисленных компаний, использующих именно их. Аналогично обстоят дела в сфере бизнеса. Поэтому к выбору отрасли, в которой ты будешь работать, следует подходить с таким же тщанием, как и к выбору подлежащих изучению технологий.

Так как бизнес-специализацию важно определить еще на стадии формирования своего портфолио, выбор фирмы и отрасли, в которой ты собираешься работать, становится существенной инвестицией в будущее. Если ты еще не думал над этим аспектом, самое время подумать. Каждый потерянный день означает упущенные возможности. Отказаться от развития в области бизнеса — все равно что оставить сбережения на сберегательном счету с низким процентом при наличии более доходных вариантов вложения средств.

> Самое время выбрать сферу бизнеса, на изучение которой ты будешь тратить свое время.

**Действуй!**

1. Пригласи на обед знакомого бизнесмена и поговори с ним о его работе. Подумай, что бы ты хотел изменить или изучить, если бы собрался потрудиться в этой области. Расспроси об особенностях трудовых будней. Узнай, как в выполнении рабочих обязанностей помогают (или мешают) современные технологии. Взгляни на *свою* работу его глазами.

   Сделай такие встречи регулярными.

   Допускаю, что мысль о подобном шаге вызывает у тебя неловкость или дискомфорт. Это нормально. Но начав несколько лет назад общаться с деловыми людьми, я стал намного лучше разбираться в бизнесе, техническую поддержку которого я обеспечивал. Кроме того, мне стало намного проще разговаривать с клиентами.

   Почитай отраслевой журнал, связанный со сферой деятельности твоей компании. Скорее всего, тебе даже не придется его покупать. В большинстве случаев старые выпуски просто где-то валяются. Именно они помогут тебе сделать первые шаги. Возможно, ты не все поймешь, но следует проявить настойчивость. Составь список вопросов, которые можно задать руководи-

телю или кому-нибудь из клиентов. Даже если какой-то вопрос покажется глупым, твое желание учиться оценят по достоинству.

Найди отраслевые сайты, которые ты сможешь просматривать на регулярной основе. В журналах и на сайтах обращай пристальное внимание на важные новости и основные статьи. С какими проблемами приходится бороться в вашей отрасли? Какая из них является наиболее актуальной на текущий момент? Обсуди это со своими заказчиками. Попроси их объяснить ситуацию и высказать свое мнение. Подумай, как эти вещи влияют на *твою* компанию, на твой отдел, на твою группу и в конечном счете на твою работу.

## Совет 4
## Будь худшим

Легендарный джазовый гитарист Патрик Мэтини дал начинающим музыкантам главный совет: «В какой бы группе ты ни был, всегда будь в ней худшим»[1].

До начала моей карьеры в области информационных технологий я был профессиональным джазовым и блюзовым саксофонистом. Именно тогда мне повезло получить этот урок и начать следовать ему. Быть худшим в группе означает работать с людьми, играющими лучше тебя.

Ты можешь спросить, зачем по собственной воле становиться худшим в любом коллективе: «Разве это не бесит?» Да, сначала такая ситуация крайне нервирует. Будучи новичком, я оказывался в ситуациях, когда моя «худшесть» откровенно бросалась всем в глаза. Придя на работу, я не хотел доставать саксофон и боялся, что меня просто сбросят со сцены. Я находился среди людей, которые ждали, что я буду играть на их уровне и даже солировать!

> В какой бы группе ты ни был, всегда будь в ней худшим.

[1] http://clabs.org/blogki

И в этой ситуации обязательно (к счастью!) происходило какое-то чудо: мне удавалось вписаться. Разумеется, среди остальных музыкантов я не производил впечатления звезды. Но не было также впечатления, что я не дотягиваю до общего уровня. Этому чуду были два объяснения. Во-первых, на самом деле я не был настолько плохим, как мне казалось. Но об этом мы подробнее поговорим чуть позже.

Интереснее то, что я встраивался в коллектив этих знаменитых музыкантов — порой бывших моими кумирами, — начиная играть, как они. Было бы лестно думать, что я, как сверхчеловек, обладаю умением становиться гением просто потому, что рядом со мной находится гений, но теперь по прошествии времени я вижу более прозаическое объяснение. Скорее я был запрограммирован на стадное поведение. Именно эта программа заставляет меня, к примеру, перенимать словечки и грамматические конструкции тех, с кем приходится общаться. После полутора лет в Индии я стал говорить так, что моя жена то и дело принималась хохотать. Она спрашивала меня: «Ты *слышал*, что ты сейчас сказал?» Я говорил по-английски как индус.

Будучи худшим парнем в группе, я начинал вести себя так же, как и в музыкальной группе. Естественным образом подстраивался под игру остальных членов коллектива. Но есть фактор, который превращает данное свойство в проблему. Дело в том, что, играя в казино и в захолустных барах с не самыми лучшими музыкантами, я спускался на *их* уровень. Более того, подобно алкоголику, язык которого заплетается даже в трезвом состоянии, я демонстрировал ту же самую отвратительную манеру игры, выступая в более приличных местах.

> Окружающие тебя люди влияют на твой результат. Поэтому тщательно подходи к выбору коллектива.

В итоге я понял, что улучшить или, наоборот, ухудшить свои навыки можно в зависимости от того, с кем приходится работать. Ведь продолжительный опыт сотрудничества с одной группой оказывает сильное влияние на стиль твоих выступлений.

Позднее, после перехода в компьютерную индустрию, я обнаружил, что привычка искать самых лучших музыкантов сопровождает меня

и как программиста. Возможно, неосознанно я искал работу среди лучших специалистов в области IT. Неудивительно, что тенденция сохранилась. Быть самым худшим парнем (или девушкой) в команде — все равно что быть худшим музыкантом в группе. Ты обнаружишь, что стал необъяснимо *умнее*. Ты даже начнешь грамотнее говорить и писать. Твой код и дизайн станут более элегантными, и вдруг окажется, что ты способен предлагать все более творческие решения сложных задач.

Вернемся к первому объяснению моей способности встраиваться в группу легче, чем ожидалось. Я и в самом деле не был настолько плох, как думал. Действительно, очень легко понять, как оценивают твой уровень другие музыканты. Если ты играешь хорошо, они пригласят тебя выступить с ними снова. В противном случае тебя начнут избегать. Это куда более надежный критерий, чем просьба высказать свое мнение, ведь хорошие музыканты не любят играть в одной группе с плохими. К моему изумлению, превосходные музыканты часто звали меня выступать с ними и даже предлагали организовать собственную группу.

На самом деле попытка быть худшим не дает тебе недооценить себя. Можно быть первоклассным музыкантом, но всю жизнь играть в заштатных коллективах просто потому, что ты боишься. Прямо признавая себя далеко не лучшим, мы избавляемся от страха, что на нашу недостаточную квалификацию укажет кто-то другой. На самом же деле, даже если ты *попытаешься* стать худшим, легко это не получится.

### Действуй!

1. Найди ситуацию, в которой ты можешь быть «худшим». Разумеется, далеко не у всех есть роскошная возможность менять команды и фирмы просто из-за появившегося вдруг желания работать с лучшими специалистами. Но можно найти некоммерческий проект, в рамках которого ты будешь сотрудничать с теми, кто способен постепенно повысить твой уровень. Найди информацию о встречах специалистов, живущих в твоем городе, и начни принимать в них участие. Разработчики часто ищут проекты, которыми можно заниматься в свободное время, используя новые приемы и оттачивая имеющиеся навыки.

Если в твоем городе нет организованного сообщества программистов, используй интернет. Найди привлекательный открытый проект с командой, которая находится на более высоком уровне, чем твой. Узнай чем они занимаются, просмотри архивные рассылки, выбери функциональность, над которой работают, или обнаруженную критическую ошибку, и приступай к написанию кода! Имитируй стиль готового кода по данному проекту. Преврати это в игру. Пусть твой код и дизайн будут так похожи на остальные материалы проекта, что даже отцы-основатели, стоявшие у истоков, не смогут вспомнить, кто это написал. Как только результаты твоих трудов тебя устроят, покажи их. Если все сделано на уровне, твой фрагмент кода будет включен в проект. После этого можно приступать к написанию следующего фрагмента. Если разработчики проекта не согласны с твоим решением, внеси исправления с учетом их отзывов и повторно отправь на рассмотрение или же возьми на заметку данные ими рекомендации. Работая над следующим дополнением к проекту, попытайся обойтись меньшим количеством переделок. В итоге ты станешь полноправным членом команды. Ты удивишься, чему можно дистанционно научиться у опытных разработчиков, даже если вы никогда не встретитесь лично.

## Совет 5
## Инвестируй в интеллект

При выборе поля будущей деятельности порой так и подмывает остановиться на технологиях, с которыми связано максимальное количество рабочих мест. Например, заняться изучением Java. Или программировать для .NET. Знание Java позволяет претендовать, а возможно, и получить работу, связанную с написанием Java-кода.

По такой логике не имеет смысла вкладываться в узкоспециализированные технологии, особенно если вы не планируете работать в этой области.

Компания *TIOBE Software* воспользовалась поисковыми службами интернета для определения относительной популярности языков программирования. За основу были взяты обсуждения различных языков[1]. На сайте написано, что «общемировой рейтинг составлен

[1] http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

по информации о наличии квалифицированных инженеров, курсов и независимых производителей». Разумеется, речь не идет о научно обоснованной оценке, но тенденции говорят сами за себя.

На момент написания данной книги самым популярным был Java, затем следовал C. C# занимал почетное шестое место, но его популярность росла. ABAP — внутренний язык программирования компании SAP — оказался на семнадцатом месте, медленно утрачивая свои позиции. Мой любимый язык программирования Ruby (именно на нем написаны все мои наиболее серьезные проекты, кроме того, я принимаю участие в организации ежегодных международных конференций по этой теме) был на одиннадцатом месте. Впрочем, к моменту выхода первого издания этой книги он перестал входить даже в первую двадцатку. Он проиграл даже ABAP!

Я спятил, если все еще пользуюсь Ruby? Или попросту глуп? На ум первым делом приходят эти два объяснения, не так ли?

Поль Грэм в статье «Великие хакеры»[1] утверждал, что программисты, пишущие на Java, далеко не так умны, как приверженцы Python. Он взбесил множество глупых Java-программистов (неужели я это написал?), которые принялись писать на своих сайтах развернутые контраргументы. Бурная реакция показала, что он задел за живое. Я присутствовал при первой презентации этой статьи. И она заставила меня вспомнить один эпизод.

Я приехал в Индию, чтобы набрать сотрудников, и просматривал сотни кандидатов, претендующих на десяток рабочих мест. Наша команда, проводящая собеседования, страдала от нехватки сил и времени из-за низкого процента успешно пройденных собеседований. Несмотря на поздний вечер, головную боль и красные глаза, мы не расходились, пытаясь найти новый подход к отбору кандидатов. Требовалось оптимизировать процесс, увеличив число интервьюируемых или каким-то образом привлекая *более толковых* людей (а лучше и то и другое). Тем, что осталось от моего голоса после двенадцатичасовых попыток получения ответов от ошеломленных программистов, я предложил добавить к ключевым словам, по которым наши

[1] http://paulgraham.com/gh.html

специалисты по поиску персонала отбирали резюме из базы данных, слово «Smalltalk». «В Индии никто не знает этого языка», — кричал заведующий отделом кадров. Но на это и был направлен мой расчет. Ведь программирование на Smalltalk коренным образом отличается от программирования на Java. Вариативный опыт даст нам новый уровень требований к кандидатам, а динамическая природа Smalltalk позволит Java-программистам подойти к решению задач с другой стороны. Я надеялся, что эти факторы дадут нам специалистов с высоким уровнем технической подготовки, которого не было у уже просмотренных соискателей.

Добавление к списку требований Smalltalk удивительным образом уменьшило кадровый пул. Но теперь к нам приходили более одаренные люди. Они действительно разбирались в объектно-ориентированном программировании. Они знали, что Java не является универсальным, как его порой пытаются представить. Многие из них *обожали* программировать! Нам оставалось недоумевать, *где же вы все были в предыдущие две недели?*

К сожалению, наши возможности по привлечению подобных разработчиков сильно ограничивал уровень зарплат, которые мы могли предложить. В данном случае они могли диктовать свои условия, и большинство из них предпочло остаться на прежней работе или продолжить поиски. Мы потерпели неудачу с наймом, но получили бесценный урок: искать лучше среди кандидатов с многообразным (и даже нетрадиционным) опытом, а не среди тех, кто посвятил себя решению однотипных задач. Я объясняю это тем, что хорошие специалисты сами стремятся к разнообразию, потому что им нравится изучать новое. Да и вынужденная работа в незнакомой сфере формирует более зрелых и всесторонне образованных разработчиков программного обеспечения. Впрочем, *по какой бы причине* подобная ситуация ни сложилась, мы поняли, что она работает. Я до сих пор пользуюсь этим приемом при поиске разработчиков.

Но какие еще причины, кроме попытки попасть в поле *моего* зрения, когда я занимаюсь подбором кадров, могут заставить тратить время на освоение второстепенных технологий, на применении которых вам, возможно, ни разу в жизни не удастся заработать?

Как лицо, отвечающее за подбор персонала, могу сказать, что в первую очередь это показывает степень вашей заинтересованности. Если я *узнаю* о том, что человек изучает некую область для саморазвития или, еще лучше, — для удовольствия, то пойму, что передо мной целеустремленный и любящий свою профессию специалист. Когда я спрашиваю людей, довелось ли им познакомиться или использовать некоторые нестандартные технологии, меня сводит с ума ответ: «У меня не было возможности работать в этой области». *Не было* возможности работать? Но у меня ее тоже не было! Но я использовал свою возможность учиться.

Тебе не дают возможности…? Используй свой шанс!

Эти второстепенные технологии и методологии не только создают образ правильно мотивированного и увлеченного своим делом человека. Они и в самом деле делают вас глубже, лучше, умнее и изобретательнее.

Если вы не считаете все вышеперечисленное достойными внимания причинами, возможно, вы неверно выбрали профессию.

### Действуй!

1. Изучи новый язык программирования. Не имеет смысла переходить с Java на C# или с C на C++. Новый язык должен изменить тип твоего мышления. Если ты программист, работающий на Java или C#, попробуй освоить Smalltalk или Ruby, в которых отсутствует статическая типизация. А если ты долгое время занимаешься объектно-ориентированным программированием, обрати внимание на функциональные языки, например Haskell или Scheme. Ты не обязан достигать вершин мастерства. Просто попытайся написать достаточный объем кода, чтобы прочувствовать отличия новой среды программирования. Если тебе ничего не кажется странным, возможно, выбран неверный язык или ты используешь старый тип мышления в новых условиях. Отойди от привычных принципов и освой новые подходы. Попроси специалистов посмотреть на твой код и дать совет, как переписать его в соответствии с особенностями данного языка.

# Совет 6
# Не слушай родителей

В нашей культуре следование родительским советам считается чем-то сакральным. Это обязанность ребенка, почти такая же незыблемая, как религиозные догматы. Книжные, киношные и телевизионные сюжеты завязаны на родительскую мудрость. Но в нашей индустрии этот принцип скорее вреден.

Родители хотят, чтобы с ребенком все было в порядке. А блистательная карьера и связанные с этим риски в такую схему не вписываются. Родители дают панические советы чаще, чем кто бы то ни было другой, к кому мы склонны прислушиваться. Советы позволяют *обойтись без потерь*, но, стараясь обойтись без потерь, мы *теряем* возможность выиграть! Победители рискуют. Они думают о том, чего хотят добиться, а не как подстраховать свои тылы. Планирование карьеры под влиянием страха — это, скорее, не дорога к величию, а ссылка в офис до конца своих дней. Да, это безопасно, но никакого удовольствия при этом получить нельзя.

Поколение назад удовольствие не принималось во внимание, когда речь заходила о выборе профессии. Работа не должна была приносить удовольствие. Она должна была приносить в дом мясо. Удовольствие же было связано с тем, что мы делали в свободные от работы часы. Удовольствиями мы занимались по вечерам и в выходные. Но постепенно выяснилось, что если работа не доставляет удовольствия, ни о каком *фантастическом* выполнении своих обязанностей не может быть и речи. В наши дни ситуация не сильно изменилась, но культурные представления о том, как должна выглядеть работа, сместились в лучшую сторону. Многие из нас понимают, что энтузиазм ведет к совершенству. А без удовольствия в работе, связанной с программным обеспечением, скорее всего, не возникнет никакого энтузиазма.

Другим влияющим на карьеру фактором, идущим вразрез с представлениями наших родителей, является допустимость (а часто и желательность) смены рабочих мест. Всесторонне образованный специалист по программному обеспечению знаком с разными аспектами выбранной отрасли: проектированием продуктов, информационно-технологической поддержкой, разработкой внутренних коммерческих

систем, работой по государственному заказу. Чем больше областей ты изучил, чем больше разных архитектур прошло через твои руки, тем проще тебе принимать решение в сложных случаях. Работа в одной и той же компании с постепенным продвижением по карьерной лестнице замедляет твой рост как разработчика. Времена, когда можно было присоединиться к крупной фирме и строить там карьеру, миновали. Подобное поведение считалось признаком надежности. Но теперь это недостаток. Информацию, что у тебя было всего одно рабочее место и тебе приходилось иметь дело только с одним набором систем, многие (умные) менеджеры по персоналу воспримут в отрицательном ключе. Лично я предпочту взять на работу человека, у которого было множество успехов и неудач в различных областях, а не того, кто умеет работать всего одним проверенным способом.

Несколько лет назад я понял, что моя карьера в изрядной степени базировалась на профессиональных ценностях моих родителей и их поколения. Я работал в одной из самых больших и стабильных в мире корпораций и медленно, но неуклонно двигался к успеху. Но я был совершенно инертен. Я уверял себя, что мое положение не относится к категории «синица в руке». Корпорация была настолько велика, что давала возможность выполнять множество различных обязанностей на бесконечном на первый взгляд числе мест. Но в конечном счете я сидел на одном месте и выполнял одну и ту же работу.

Помню, я заговорил с другом о потенциальном уходе из корпорации, и он сказал мне: «Разве твоя судьба до конца дней работать в *название_крупной_компании*?». *Нет, черт побери!* Я быстро нашел другую работу и уволился.

Это движение положило четкое начало моему неравномерному подъему в индустрии программного обеспечения. Я знакомился с новыми областями, я работал над более сложными проблемами и получал за это куда более весомое вознаграждение, чем раньше. Временами мне бывало страшно, но когда я решил, что страх и консервативность не должны влиять на выбор моего пути, форма и характер моей карьеры — и моя жизнь — изменились к лучшему.

В карьерных вопросах нужно идти на обдуманный риск. Не позволяй страху завладеть тобой. И если ты не получаешь удовольствия от работы, ты никогда не достигнешь в ней совершенства.

## КАК Я ПРОМЕНЯЛ $300 000 В *MICROSOFT* НА ПОЛНЫЙ РАБОЧИЙ ДЕНЬ В GITHUB

*Том Престон-Вернер, один из основателей социальной сети GitHub*

Шел 2008 високосный год. 366 дней назад почти минута в минуту я в одиночестве сидел в спорт-баре на Третьей улице Сан-Франциско. Обычно я не болтаюсь по барам, но в тот четверг была организована вечеринка «I Can Has Ruby». Уже в то время я считал, что слова «I can has ... »[1] можно добавить к чему угодно. В данном случае речь шла о полузакрытом сборище Ruby-программистов, которое быстро превратилось в обычную ночную пьянку. Воспоминания о подобных развлечениях, как правило, исчезают, как только проходит утреннее похмелье. Но не на этот раз. Ведь этой ночью родилась социальная сеть GitHub.

Я занял отдельный кабинет, заказал бокал холодного пива и хотел отдохнуть от бурного общения за длинными столами в тускло освещенной задней части бара. Не успел я сделать пять или шесть глотков, как ко мне вошел Крис Ванстрас. Сейчас и не вспомню, воспринимал ли я его тогда как друга. Мы пересекались на встречах и конференциях, посвященных Ruby, но достаточно поверхностно — на уровне взаимного «Крутой код ты пишешь». Не помню, что меня подтолкнуло, но я сделал приглашающий жест и произнес: «Дружище, ты только глянь». Неделей раньше я начал работу над проектом Grit, который позволял мне через Ruby-код в объектно-ориентированном стиле получить доступ к Git-репозиториям. В то время Крис был одним из немногих Ruby-программистов, серьезно относящихся к системе Git. Он сел, и я начал демонстрировать ему немногочисленные наработки. Но этого оказалось достаточно, чтобы Криса охватил приступ энтузиазма. Почувствовав его настроение, я поделился еще не оформившейся до конца идеей создания сайта, который будет работать как центр обмена Git-репозиториями отдельных кодеров. Я даже придумал для него имя: GitHub. Возможно, я несколько искажаю его слова, но высказался он решительно: «Я в деле!»

Следующей ночью — в пятницу, 19 октября 2007 года, в 22:24, — Крис сделал первый вклад в репозиторий GitHub и оставил запись о запуске нашего совместного детища.

Никаких соглашений по поводу дальнейших действий у нас не было. Мы были всего лишь двумя парнями, которые решили воплотить в жизнь то, что здорово звучало на словах.

---

[1] Намеренно неграмотно написанная фраза «могу ли я получить ... ?»

### КАК Я ПРОМЕНЯЛ $300 000 В *MICROSOFT* НА ПОЛНЫЙ РАБОЧИЙ ДЕНЬ В GITHUB (продолжение)

Помните потрясающую сцену из фильма «Парень-каратист»? Ту самую тренировку Дэниэла, когда он превращается в мастера боевых искусств. Помните эту музыку? Думаю, вам нужно еще раз послушать песню Джо Эспозито «You're the Best», так как я собираюсь сразу перейти к сути.

В течение следующих трех месяцев мы с Крисом провели много восхитительных часов, занимаясь проектом GitHub и написанием кода. Я продолжал работать над Grit и пользовательским интерфейсом. Крис разрабатывал приложение Rails. По субботам мы встречались для обсуждения проекта и ломали голову над будущим тарифным планом. Я помню один до ужаса дождливый день, когда мы добрых два часа рассматривали стратегии ценообразования, поедая лучшие в городе спринг-роллы. При этом мы выполняли и другие обязанности. Лично я был разработчиком инструментов в группе рейтинга и соответствия фирмы *Powerset*.

В середине января, три месяца проработав по ночам и выходным, мы разослали нашим друзьям приглашения на закрытое бета-тестирование. В середине февраля в проект пришел Пи Джей Хиет, ставший третьим соучредителем. Официальный запуск состоялся 10 апреля. *TechCrunch* об этом не информировали. На тот момент мы все еще были обычными программистами в возрасте чуть за двадцать, а у проекта отсутствовали сторонние инвесторы.

Я все еще работал в *Powerset*, когда 1 июля 2008 года стало известно, что эту фирму примерно за сто миллионов долларов покупает корпорация *Microsoft*. Момент был достаточно интересным, ведь мне предстояло сделать выбор гораздо раньше, чем я предполагал. Можно было стать сотрудником *Microsoft* или уйти и полностью посвятить себя GitHub. Мне было 29, и я был старше своих компаньонов, успев как накопить больше долгов, так и привыкнуть к большим ежемесячным расходам. Мой образ жизни был достаточно роскошным. Кроме того, приближался момент, когда моя жена Тереза должна была вернуться из Коста-Рики, где она занималась сбором данных для своей диссертации. А значит, пора было оставить временные холостяцкие привычки и снова жить как семейный человек.

Еще сильнее осложнил мой выбор тот факт, что в *Microsoft* мне предложили крайне заманчивые условия. Оклад и лакомые $300 000 на протяжении трех лет. Вполне достойная сумма, чтобы заставить серьезно задуматься кого угодно. В итоге мне пришлось выбирать между стабильной и надежной работой с гарантированно высокой зарплатой в *Microsoft* и рискованной тропой частного предпринимателя с неизвестным уровнем дохода. Останься я в *Powerset*, мои от-

**КАК Я ПРОМЕНЯЛ $300 000 В *MICROSOFT* НА ПОЛНЫЙ РАБОЧИЙ ДЕНЬ В GITHUB (продолжение)**

ношения с ребятами из GitHub неминуемо обострились бы. Ведь они некоторое время назад, подкопив денег, ушли в свободное плавание, чтобы посвящать все свое время проекту GitHub. Это был момент, когда на карту было поставлено все. Я должен был либо полностью включиться в работу над GitHub, либо навсегда забыть о нем, предпочтя большие деньги от *Microsoft*.

Если вам нужен рецепт бессонных ночей, могу поделиться. Смешайте страх «Что подумает моя жена?» и 3000 стодолларовых бумажек, взболтайте с «пивом в любое время, когда захочется» и приправьте шансом на финансовую независимость.

Я неплохо научился доносить до работодателя плохую новость о своем уходе. Я завел этот разговор в тот день, когда мне должны были предложить работу. Я рассказал, что полностью собираюсь посвятить себя проекту GitHub. Как любой хороший начальник, он огорчился, но понял. И даже не пытался соблазнять меня большим размером премий или чем-то в этом роде. Думаю, он давно догадался о моем намерении уйти. Возможно, мне изначально предлагали большую зарплату, чем остальным, поскольку учитывали риск моего ухода. В этом деле менеджерам *Microsoft* нет равных, вы уж поверьте. Они владеют наукой выплаты бонусов, направленных на удержание. Но с людьми, обладающими мышлением предпринимателя, это не работает. С такими людьми все проверенные методы дают сбой.

В конце концов, подобно Индиане Джонсу, который просто не мог отказаться от поисков Святого Грааля, я не мог отказаться от шанса работать на самого себя над проектом, который мне по душе, какие бы безопасные альтернативы мне ни предлагали. Я думаю, что глубоким старцем на смертном одре я оглянусь на свою жизнь и скажу: «Да, это было захватывающее приключение», а не «Ну, зато я чувствовал себя комфортно».

## Действуй!

1. Определи свои самые серьезные страхи, связанные с карьерой. Вспомни последние принятые в этом направлении решения. В этот список должны попасть не только ответственные решения (в конце концов, если твой выбор определяется страхом, ответственных решений, скорее всего, не было). Это могут быть взятые на себя дополнительные обязанности, заявление о смене

работы или повышении. Когда список будет готов, честно оцени каждый из пунктов: насколько твоим решением в данной ситуации управлял страх? Что бы ты сделал, если бы страха не было? Если решение было принято под влиянием страха, как изменить его или найти аналогичную ситуацию, в которой можно будет действовать уже более взвешенно и обдуманно?

## Совет 7
## Будь универсалом

По меньшей мере пару десятилетий доведенные до отчаяния менеджеры и предприниматели делают вид, что разработка программного обеспечения представляет собой технологический процесс. Пишутся технические требования, а архитекторы воплощают их в высокоуровневых проектах. Проектировщики дополняют их подробной проектной документацией, которая затем передается кодерам, напоминающим роботов. В одной руке они держат описание задачи, а другой сонно набирают реализацию проекта. Готовый код получает инспектор номер 12, который никогда не поставит гриф «Согласовано» при несоответствии исходным техническим требованиям.

Тому, что менеджеры хотят превратить разработку программного обеспечения в производство, удивляться не стоит. Менеджеры *понимают*, как управлять производством. Многолетний опыт научил нас эффективно и точно создавать физические объекты. Значит, применив те же самые принципы, мы сможем оптимизировать процесс разработки программного обеспечения, превратив его в такой же хорошо отлаженный механизм, как фабричное производство.

Все сотрудники так называемой фабрики по производству программ являются специалистами. Они занимают свои места в конвейерной цепочке, при помощи своих инструментов соединяя друг с другом Java-компоненты или шлифуя шероховатости приложений, написанных на языке Visual Basic. Инспектор № 12 работает тестировщиком. К нему стекаются компоненты программ, и он каждый день единообразно проверяет их и ставит штамп. J2EE-проектировщики создают J2EE-приложения. C-кодеры пишут код на языке C. Это крайне простой и поделенный на четкие категории мир.

Но, к сожалению, аналогия с производством не работает. *Программы* должны быть не менее гибкими, чем требования к ним. В бизнесе все меняется, и бизнесмены знают, что программное обеспечение вполне можно изменить в соответствии с новыми требованиями. Это означает, что архитектура, дизайн, код и тесты должны создаваться и пересматриваться более динамично, чем может предложить самый рациональный производственный процесс.

В столь быстро меняющейся среде только гибкость позволит добиться превосходных результатов. В затруднительной ситуации умные бизнесмены обращаются к профессионалам, которые могут сразу решить проблему. Как же стать человеком, имя которого первым делом вспоминают, когда требуется супергерой, способный всех спасти? В данном случае все дело в умении решать все проблемы, которые только могут возникнуть.

Что это за проблемы? Ты этого не знаешь. И я не знаю. Известно *только*, что они столько же разнообразны, как вопросы развертывания, как критические недостатки дизайна, которые следует быстро устранить и провести повторную реализацию, как интеграция неоднородных систем, как ситуативная генерация отчетов. Столкнувшись с настолько разнородным набором проблем, бедный инспектор номер 12 довольно быстро допустит сбой в своей работе.

Поговорка «и швец, и жнец, и на дуде игрец» изначально имеет уничижительный смысл, подразумевая под собой человека, который из-за неумения концентрироваться не смог детально изучить ни одного дела и достичь в нем подлинного мастерства. Но когда приложение для покупок в твоем интернет-магазине перестает работать, каждый час лишая тебя заказов на сотни долларов, нужен человек, который не только знает, как функционирует код данного приложения, но и может произвести низкоуровневую отладку процессов на сервере, работающем под управлением UNIX, проанализировать конфигурацию системы управления реляционной базой данных на предмет влияющих на производительность «узких мест», проверить конфигурацию маршрутизатора. И, что куда важнее, обнаружив проблему, этот мастер на все руки сможет быстро принять решение по поводу архитектуры и дизайна, внести в код исправления и подготовить новую версию системы к работе. В подобной ситуации производственный сценарий в лучшем

случае имеет странный, а в худшем — совершенно непригодный к применению в рамках производственного процесса вид.

Кроме того, фабрика программного обеспечения выглядит нереализуемой, потому что в отличие от реального конвейера, работа которого происходит в устоявшемся режиме, создание программ обычно представляет собой циклический процесс. Это касается не только фактического течения проекта, но и работы внутри этого проекта. Чтобы избежать периодов простоя на время уточнения требований, архитектуры и дизайна, кодер предпочитает одновременно работать с несколькими проектами. Но в этом случае, как только доходит до дела, кодер предпочитает опираться на предысторию и свой опыт, что противоречит самой идее фабрики программного обеспечения. Документация с описанием требований, архитектуры и дизайна дает преимущество в начале работы, но, по большому счету, если программист не понимает, как должна функционировать система, он попросту не сможет создать хорошую реализацию.

Разумеется, я имею в виду не только кодеров. Написанное верно и для всех остальных участников конвейера. Важно рабочее окружение, а многозадачность не всегда работает. В результате мы получаем неэффективную систему производства. Предпринимались различные попытки повысить эффективность в рамках системы, построенной по типу фабричного производства, но оптимизировать такие фабрики до приемлемого уровня пока не получилось.

*Обычный* кодер, тестировщик, дизайнер или проектировщик в моменты, когда работа над проектом приостанавливается, будет бить баклуши или заниматься имитацией деятельности. *Обычный* программист, работающий с J2EE, .NET или UNIX, не сможет ничего внести в проект на стадиях, не имеющих отношения к его непосредственной области деятельности. И дело тут не в том, каким из звеньев производственной цепочки ты себя ощущаешь (а высшим звеном тут всегда будет разработчик архитектуры). Дело в том, насколько полезным ты можешь стать.

Если ты хочешь оказаться последним человеком в списке на увольнение, имеет смысл стать полезным в как можно большем числе случаев. Если существует вероятность превращения некогда многолюдного офиса разработчиков в прибежище для немногочислен-

ных сотрудников местного разлива, составляющих костяк фирмы, следует осознать, что в этом случае «простые тестировщики» или «простые кодеры» востребованы не будут. И лучше всего, если ты, понимая свое место в общей картине, сразу хочешь выделиться и стать исключительным.

Чтобы стать универсалом, нужно не связывать себя с определенной ролью или технологией. Определить свое место можно разными способами. Чтобы лучше представить себе возможности универсала, имеет смысл разбить перспективы карьеры в IT на набор независимых аспектов. Я выделил пять, но на самом деле их может быть бесконечное количество (все зависит от того, как лично вы делите темы):

> Универсалы встречаются редко… и потому ценятся особо высоко.

- ступенька карьерной лестницы;
- платформа/ОС;
- код в сравнении с данными;
- системы в сравнении с приложениями;
- бизнес в сравнении с IT.

Это различные варианты подхода к задаче превращения в универсала. Один из способов подумать о своей карьере в целом. Допускаю, что для себя ты напишешь другой список. А пока обсудим предложенные варианты.

В первую очередь можно выбирать между руководителем, администратором и техническим специалистом. Возможно, ты классифицируешь себя как разработчика, противопоставляя себя программистам и тестировщикам. Возможность гибко выбирать доступные роли является тем аспектом, ценности которого многие не понимают. К примеру, считается, что сильный лидер должен по возможности избегать выполнения чужих обязанностей. Но фирма, в которой локально работают всего несколько сотрудников, только выиграет от наличия человека, умеющего не только руководить персоналом и проектами, но и, засучив рукава, исправить возникшие в последнюю минуту критические ошибки, пока отсутствующие сотрудники спят. То же самое

можно сказать про проектировщика архитектуры ПО, который в состоянии резко ускорить работу над проектом, занявшись *написанием фрагментов кода*. Но чаще всего люди не пересекают иерархических границ не потому, что не хотят, а потому, что не могут. Фанатики программирования не имеют навыков руководства, руководители не представляют, как пишется код. Редко можно встретить человека, хорошо справляющегося с обеими задачами.

Еще одна искусственная (и непозволительная) граница разделяет платформы и операционные системы. Все более непрактично становится быть специалистом по UNIX, отказывающимся работать с Windows. То же самое касается .NET и J2EE и любых других инфраструктурных платформ. Долгосрочные перспективы на рабочем месте требуют независимости от платформы. У каждого из нас есть свои предпочтения, но их лучше оставить дома. Стань экспертом в одном и как следует разберись во всем остальном. Платформа — только инструмент. Специалиста по Windows можно нанять и на Филиппинах. А вот того, кто реально разбирается в разработке для Windows и UNIX и может помочь с интеграцией в обеих системах, лучше поискать в собственной стране. От подобных навыков не стоит отказываться, так как они относятся к умению работать в команде.

> Ваши навыки не должны ограничиваться одной технологической платформой.

Также сложно провести водораздел между администратором баз данных (должность, которая в прошедшее десятилетие появилась буквально из ниоткуда) и разработчиком программного обеспечения. Во многих организациях администраторы баз данных (DataBase Administrator, DBA) знают, как использовать кое-какие инструменты администрирования и как установить конкретный программный продукт. От них не требуют умения *использовать* базы данных. Разработчики программного обеспечения со своей стороны ленятся работать с базами, а порой и плохо представляют, как это делается. В итоге одно цепляется за другое.

Первое, что удивило меня, когда я начал заниматься информационными технологиями, — это количество неплохо образованных программистов (у меня даже создалось впечатление, что таких большинство),

не имеющих понятия о том, как настраивается система, которую они разрабатывают и развертывают. Мне приходилось сотрудничать с разработчиками, не способными установить на компьютер операционную систему, я уж молчу про настройку сервера, на котором они разворачивали свои приложения. Редко можно встретить разработчика, действительно разбирающегося в платформе, на которой он работает. Но ведь именно у таких получаются самые лучшие приложения, а проекты завершаются быстрее.

Ну и, наконец, как отмечено в совете 3, первым делом следует разрушить границу между бизнесом и информационными технологиями. Начинай изучать принципы бизнеса, с которым связана твоя деятельность.

**Действуй!**

1. Составь список областей, в которых ты можешь или не можешь обобщить свои знания и умения. Укажи свою специализацию в каждой из сфер. Например, для *Платформа и операционная система* можно записать *Windows. NET*. Справа от своей специализации перечисли то, что ты собираешься изучать. В приведенном примере можно написать *Linux* и *Java* (или даже *Ruby* или *Perl*).

   Как можно быстрее (не откладывая на следующую неделю!) выдели полчаса в день на изучение хотя бы одной из указанных тем. Не ограничивайся чтением. Используй любую возможность, чтобы попрактиковаться. В случае сетевых технологий можно загрузить пакет с веб-сервером и настроить его своими руками. Если ты решил поближе познакомиться с бизнесом, найди на работе кого-нибудь из клиентов, кто согласится пообедать с тобой и ответить на вопросы.

## Совет 8
## Будь специалистом

«Как бы вы написали на Java программу, которая уронит виртуальную машину Java?» А в ответ — тишина... «Эй, как вас там? Ау!»

«Извините, я вас не понимаю. Вы не могли бы повторить вопрос?» В голосе послышалось отчаяние. По опыту я знал, что повторение

вопроса вряд ли чему-то поможет. Поэтому я повторил вопрос медленно и громко. «Как бы вы написали на Java программу, которая уронит виртуальную машину Java?»

«Э-э-э… прошу прощения. Я раньше никогда этого не делал».

«Я уверен, что не делали. Но что там с вопросом: как бы вы написали на языке Java программу, которая НЕ будет приводить к падению JVM?»

Я искал по-настоящему квалифицированных Java-программистов. Перед началом собеседования я попросил этого человека (как и всех остальных, просмотренных за эту неделю) оценить себя по десятибалльной шкале. Он назвал цифру *девять*. Я ожидал увидеть перед собой звезду. Но если парень так высоко себя ценит, что мешает ему придумать трюк, выводящий виртуальную машину Java из строя?

Это недостаток технической подготовки.

Тем не менее он утверждал, что специализируется на языке Java. Спроси его на вечеринке, чем он зарабатывает, и он ответит: «Я Java-разработчик». Но при этом он оказался не в состоянии ответить на простой вопрос. Я не получил от него даже *неправильного* ответа. За две с половиной насыщенные собеседованиями недели вербовочной поездки по стране это было правило, а не исключение. На вакантные места претендовали тысячи специалистов по Java, и практически никто из них не мог объяснить, как работает загрузчик Java-классов, или рассказать, каким образом виртуальная машина Java обычно осуществляет управление памятью. Разумеется, все эти знания не нужны тем, кто собирается писать базовый код под чужим руководством. Но приходившие на собеседование люди позиционировали себя как *специалисты*.

Многие из нас считают, что термин «специализация» означает неумение разбираться в других вещах. Согласно такой трактовке я могу заявить, что моя мать специализируется в Windows, ведь она никогда не работала ни с Linux, ни с OS X. А мои родственники из арканзасской глубинки окажутся специалистами по кантри, потому что они никогда не слушали другой музыки.

Представь, что ты пожаловался семейному врачу на странную опухоль под кожей правой руки. И врач отправляет тебя к специалисту, который сделает биопсию. Как тебе понравится, если специалист окажется человеком, не посещавшим никаких лекций в мединституте и не имеющим опыта в областях, непосредственно не связанных с выполнением процедуры, которая вам требуется? Я не знаю, изучал ли он дополнительный материал, даже связанный с этой процедурой. Вполне может оказаться, что он получил азы необходимой информации, но больше ничего не знает. Ты спросишь: «А что, если вон тот прибор во время операции начнет пищать?» А он в ответ: «Раньше такого никогда не случалось. Не случится и на этот раз. Я не знаю, что делает этот прибор, но он никогда не пищит».

К счастью, *большинство* разработчиков программного обеспечения не несут ответственности за ситуации, когда речь идет о жизни и смерти. Недостаток их профессионализма обычно приводит к перерасходам по проекту или ошибкам в готовом продукте, за что их работодателю приходится платить деньгами, а не жизнями.

К несчастью, индустрия программного обеспечения поставила на поток выпуск подобных кадров, именующих себя *специалистами*, чтобы оправдать свою некомпетентность в смежных областях. В медицине специалистом называют человека, *детально* разбирающегося в определенной области. Доктора отправляют к ним пациентов потому, что бывают обстоятельства, когда специалист сможет помочь намного лучше терапевта.

Так что же представляет собой специалист в области программного обеспечения? Кого я повсюду искал во время вербовочной поездки? Мне требовались люди, отменно разбирающиеся в программировании на языке Java и среде развертывания. Я хотел ребят, в 80 % случаев говорящих «это мне уже знакомо», глубина знаний которых позволяла бы решить проблему и в оставшихся 20 %. Я жаждал найти того, кто, работая с высокоуровневыми абстракциями, разбирался бы в деталях, лежащих в основе их реализации. Мне был нужен человек, умеющий справляться с возникающими при развертывании проблемами или хотя бы знающий, к кому можно обратиться по конкретному вопросу.

Именно такой специалист выживет в меняющейся компьютерной отрасли. Если ты специалист по .NET, это вовсе не оправдывает тот факт, что ты не разбираешься ни в чем, *кроме* .NET. Это лишь означает, что ты являешься экспертом во всем, что касается .NET. Зависли и нуждаются в перезагрузке IIS-серверы? «Нет проблем». Интеграция системы управления версиями с Visual Studio .NET? «Я покажу вам, как это сделать». Клиент хочет отказаться от обслуживания из-за непонятных проблем с производительностью? «Дайте мне тридцать минут».

Если ты вкладываешь в термин *специалист* другой смысл, пожалуйста, никогда не утверждай, что ты являешься специалистом.

### Действуй!

1. Ты работаешь с языками программирования, которые компилируются и запускаются на виртуальной машине? Если да, найди время и разберись, как работает эта виртуальная машина. Существует множество книг и сайтов, посвященных виртуальным машинам для Java, .NET и Smalltalk. Изучить этот материал проще, чем ты думаешь.

   Даже если язык программирования, которым ты пользуешься, не зависит от виртуальной машины, узнай, что происходит при компиляции исходного файла. Как вводимый тобой код превращается из понятного тебе текста в инструкции, которые может исполнить компьютер. Как бы выглядело написание твоего собственного компилятора?

   Откуда берутся внешние библиотеки, которыми ты пользуешься? Как *на самом деле* реализуется работа с внешней библиотекой? Каким образом компилятор, операционная система или виртуальная машина соединяет друг с другом многочисленные фрагменты кода, формируя однородную систему?

   Изучив эти вещи, ты на несколько шагов приблизишься к званию специалиста в выбранной технологии.

2. Найди возможность — на работе или вне ее — и проведи занятие по каким-либо аспектам технологии, в которой собираешься достичь мастерства. Совет № 14 говорит, что преподавательская деятельность является одним из лучших способов изучения чего бы то ни было.

# Совет 9
# Не клади все свои яйца в чужую корзину

Во времена, когда я руководил группой разработки приложений, я поинтересовался у одного из своих подчиненных: «Что ты думаешь о своей карьере? Кем бы ты хотел стать?» Его ответ меня ужасно разочаровал: «Я хочу быть J2EE-разработчиком». Я спросил, почему не «проектировщиком Microsoft Word» или не «установщиком RealPlayer»?

Этот парень собирался строить карьеру на базе *конкретной* технологии, созданной компанией, *сотрудником которой он не являлся*. Что будет, если эта компания исчезнет? Что делать, если привлекательная в данный момент технология устареет? Как можно доверять свою карьеру технологической компании?

Мы, как и отрасль в целом, склонны обманывать себя, считая, что *лидер рынка* — это то же самое, что *стандарт*. В итоге некоторые полагают, что можно сделать продукт сторонней фирмы частью своих характеристик. Хуже всего, когда человек строит свою карьеру на основе продукта, не являющегося лидером рынка, упорствуя в этом, пока его карьера не подходит к концу столь плачевным образом, что остается только переосмыслить проигрышную стратегию.

Давай на секунду остановимся и вспомним, что карьера должна восприниматься, как бизнес. В принципе, построить бизнес, паразитирующий на другом бизнесе, возможно (например, фирмы, производящие продукцию для удаления шпионского ПО и латания дыр в системе безопасности браузеров от *Microsoft*), но для отдельного человека это крайне рискованный путь. Фирма сможет среагировать на изменение в расстановке сил на рынке, например неожиданный рост безопасности браузеров от *Microsoft* (или решение *Microsoft* выйти на рынок программ по удалению шпионского ПО), в то время как индивид вряд ли обладает способностями или излишками финансов, позволяющими резко изменить направление или фокус своей карьеры.

> Ориентировка на производителя, как правило, недальновидна.

Ставить во главу угла производителей не имеет смысла еще и потому, что детали реализации программного обеспечения, как правило, держатся в секрете. Ты можешь изучать патентованное программное обеспечение ровно до *границы, за которой начинается сервисное обслуживание*. Это искусственный барьер, призванный не дать пользователям самостоятельно разобраться с возникающими проблемами. Компании возводят его, чтобы получить выгоду, продавая такую услугу, как техническая поддержка. Иногда это делается сознательно, а иногда является результатом защиты интеллектуальной собственности (когда, к примеру, не публикуется исходный код).

Поэтому, хотя целенаправленные инвестиции в конкретную технологию являются *плохой идеей*, если по каким-то причинам ты *вынужден* так поступать, постарайся выбрать вариант с открытым исходным кодом. Даже если ты не можешь или не хочешь использовать подобное решение на рабочем месте, пусть оно станет платформой для глубокого погружения в технологию. Например, у тебя может появиться желание стать экспертом по серверам J2EE-приложений. Не имеет смысла детально изучать способы конфигурирования и развертывания коммерческого сервера (в конце концов, поиграть с параметрами конфигурационного файла может *кто угодно*). Загрузи сервер JBoss или Geronimo с открытым исходным кодом и выдели время, чтобы не только разобраться с механизмом работы этих серверов, но и для изучения их внутреннего устройства.

Вскоре ты обнаружишь, что твои взгляды естественным образом поменялись. В J2EE (или что ты там выбрал в качестве области специализации) на самом деле нет ничего особенного. Разобравшись в деталях реализации, ты поймешь принцип работы высокоуровневых концептуальных моделей. Ты начнешь осознавать, что как на Java, так и на любом другом языке или платформе распределенная архитектура в масштабе предприятия остается распределенной архитектурой в масштабе предприятия. Твой кругозор станет шире, а ум начнет открываться. Ты ощутишь, что концепции и шаблоны, которые понял и осмыслил твой мозг, куда более масштабируемы и универсальны, чем любые технологии производителей. И скажешь себе: «Производители могут приходить и уходить, я знаю, как спроектировать систему!»

**Действуй!**

1. Начни небольшой проект, создав две реализации. Сначала используй внутреннюю технологию фирмы, в которой ты работаешь, а затем конкурирующую технологию, постаравшись по возможности воспроизвести ее особенности.

## Совет 10
## Полюби или уходи

Возможно, этот лозунг больше напоминает название какой-то глупой мыльной оперы, но не упомянуть о нем нельзя. Если ты хочешь *достичь высот* в своей работе, ты должен быть влюблен в нее. Если же тебе все равно, это непременно станет заметно.

Переезд в Бангалор вызвал у меня настоящий восторг. Впервые за свою карьеру я ждал, что найду компанию технических специалистов, мыслящих так же, как я, и с такой же страстью к обучению. Я надеялся, что после работы мы будем собираться и вести глубокие философские дискуссии о методологиях и приемах разработки программного обеспечения. Я думал, что индийская Силиконовая долина переполнена мастерами, жаждущими достичь высот в деле разработки ПО.

Но в реальности я обнаружил *огромное количество* людей, интересующихся в основном зарплатой, и *крайне небольшое число* влюбленных в свое дело энтузиастов.

Все как дома.

Разумеется, в тот момент я не понимал, насколько схожи ситуации в двух странах. В США я сменил несколько мест работы, но каждый раз мне казалось, что я просто попал в неудачный город или в плохое окружение. Я считал результаты моих первых попыток трудоустройства исключением из правил и был уверен, что *большинство разработчиков программного обеспечения по праву занимают свои места*. А я просто *пока не оказался в нужном окружении*.

Начал я с работы на кафедре информационных технологий нашего университета, куда меня рекомендовал мой друг Вальтер. Он до-

статочно часто видел, как я работаю за компьютером, и был уверен, что я справлюсь с обязанностями лучше большинства пользователей нашего университета, нуждающихся в помощи. Я в это не верил, так как не имел нужного образования. Я был всего лишь саксофонистом, увлеченным видеоиграми. Но Вальтер заполнил за меня заявление и договорился о собеседовании. Меня взяли, не задав ни одного профессионального вопроса, и мне пришлось сразу приступить к делу.

Сначала я боялся, что коллеги мгновенно распознают во мне шарлатана. *Что этот саксофонист делает среди квалифицированных специалистов?* Ведь я оказался среди людей, обладавших учеными степенями. А я со своим музыкальным образованием пытался делать вид, что знаю еще что-что.

Через несколько дней до меня стала доходить ужасная правда. *Эти люди понятия не имели, чем, черт возьми, они занимаются!* Более того, некоторые смотрели, что я делаю, и *конспектировали*! И это были люди *с учеными степенями в области вычислительной техники*!

В первый момент мне показалось, что меня окружают идиоты. Ведь у меня вообще не было специального образования. Ночи я проводил, играя музыку в барах, а дни — играя в компьютерные игры. Я научился работать с компьютером просто потому, что мне это было интересно. Скажу честно, я учился писать программы, чтобы создавать собственные компьютерные игры. Я возвращался домой после оглушительного вечера в баре и до восхода солнца исследовал Gopher-сайты[1] с обучающими материалами. Затем я ложился спать, просыпался и продолжал изучение до момента, когда снова нужно было отправляться в бар. Я перемежал учебу моими любимыми компьютерными играми, ел и возвращался к экспериментам с протоколом Gopher и компиляторами, которые мне удавалось заполучить.

Оглядываясь назад, можно сказать, что у меня появилась зависимость. Но в хорошем смысле. Основой моего стремления к созиданию

[1] Протокол Gopher представлял собой систему обмена документами, имеющую ту же цель, что и Всемирная паутина. С появлением интернета его популярность резко пошла на убыль.

послужили те же самые побуждения, которые раньше заставляли меня писать классическую музыку или играть джазовые импровизации. Я был одержим идеей изучить все, что только можно. И это не было желанием сделать карьеру в новой области. Более того, многие друзья-музыканты считали, что я безответственным образом отвлекаюсь от своей настоящей карьеры. Но я занимался этим просто потому, что *не мог* иначе.

Именно это отличало меня от моих чрезмерно образованных, но недостаточно продуктивных коллег. Энтузиазм.

Эти люди не имели понятия, *почему* они оказались в IT. Кого-то на эту стезю привела информация о хороших зарплатах у программистов, кого-то толкнула родительская воля, а кто-то в колледже не смог придумать для себя лучшей специализации. К сожалению, все это отражалось на производительности их работы.

> Работай потому, что не можешь не работать.

Вспомнив прочитанные биографии или просмотренные фильмы о людях, достигших высот в своей области, мы обнаружим тот же самый маниакальный энтузиазм. Великий джазовый музыкант Джон Колтрейн упражнялся в игре на саксофоне, пока его губы не начинали кровоточить.

Разумеется, способности в большой степени зависят от природного таланта. Далеко не каждый сможет стать вторым Моцартом или Колтрейном. Но мы можем избежать посредственности, найдя работу, которая нас по-настоящему увлечет.

Это может быть технология или область бизнеса, приводящая тебя в восторг. При этом существуют технологии и области бизнеса, способные тебя полностью деморализовать. Негативным фактором может стать даже тип организации. Возможно, тебе нужно работать в маленьких или, наоборот, в крупных коллективах. Может быть, ты предпочитаешь нетворческий процесс, или, наоборот, лучше себя чувствуешь, когда есть пространство для маневра. Потрать время и найди собственный рецепт.

Ты можешь некоторое время притворяться, но недостаток энтузиазма обязательно скажется как на тебе, так и на твоей работе.

## ДЕРЖИ НОС ПО ВЕТРУ

*Джеймс Дункан Дэвидсон, программист и фотограф*

Моя карьера с самого начала развивалась нетрадиционно. Я всего лишь использовал все подворачивавшиеся мне возможности. Первая возможность появилась, когда я был еще в школе и писал дипломную работу по архитектуре. В возрасте 15 или 16 лет я решил, что хочу стать архитектором, и тратил много времени, вкладываясь в свое будущее. Но основой моей реальной карьеры послужило мое юношеское увлечение электронными досками объявлений. Я был из тех детей, которые обожали модем на 300-бод в семейном компьютере. В конце концов, это привело меня в Сеть, сначала в Gopher, а затем во Всемирную паутину.

Сеть меня захватила. Я быстро создал несколько сайтов, используя преимущества всех имевшихся в моем распоряжении технологий, которые изучал по мере необходимости. В то время я считал эту работу экспериментом по киберархитектуре. Сейчас это звучит чересчур напыщенно и даже в какой-то мере глупо, но именно в таком мире мы жили в первые дни становления интернета. Мы пытались понять, что может принести с собой будущее.

Разумеется, реальная работа по построению будущего интернета происходила совсем не в архитектурных мастерских, а в сфере бизнеса. Достаточно быстро благодаря тому, во что я превратил свою персональную страницу, со мной связалась молодая компания, занимающаяся созданием сайтов для таких организаций, как «Хилтон» или бюро по совершенствованию бизнеса. Они видели плоды моего творчества, и внезапно оказалось, что я обладаю именно теми навыками, которые им требуются. Мне предложили работу с окладом, который по тем временам казался до нелепости огромным. Я принял это предложение, решив, что на некоторое время имеет смысл оседлать волну удачи, подкопить денег и через несколько лет вернуться к учебе.

Был 1995 год. Я слабо представлял, во что все это выльется и куда меня приведет желание освоить новую для меня область. Помогая в создании первой версии сайта отелей «Хилтон», позволяющей бронировать номера в реальном времени, я научился применять различные серверные технологии. Через несколько месяцев мое ученичество закончилось, и я занялся созданием фреймворков на стороне сервера. Сейчас это кажется нелепым, но в тот момент требовался именно такой подход. Я увидел возможность, воспользовался ею и поставил на кон все свои резервы, переосмысливая себя по мере необходимости.

Одно цеплялось за другое. В 1997 году я пришел в JavaSoft, где начал работать над приложениями, исполняющимися на стороне сервера, и, в конце концов,

**ДЕРЖИ НОС ПО ВЕТРУ (продолжение)**

стал отвечать за спецификацию сервлетов. К сожалению, мои усилия финансировались в недостаточном объеме, кроме того, у меня не было команды, помогающей в решении актуальных задач, в том числе построении эталонной реализации. Однако меня это не остановило, итогом был JavaServer Web Development Kit. Немногие помнят эту программу. А вот о ее следующем выпуске знает большинство «серверных» Java-программистов: Tomcat был выпущен Apache Software Foundation одновременно со служебной программой Ant. История выхода этой версии достойна отдельной книги. Упомяну лишь тот факт, что все это сопровождалось удивительным количеством случайностей, которыми я смог воспользоваться.

Поработав на *Sun* четыре года и столкнувшись с вопросом «А дальше что?», я решил стать независимым. И начал писать книги для издательства O'Reilly. Разрабатывать программное обеспечение для Mac. Я создал изрядное количество программ, так и не дошедших до выпуска. В конце концов, я занялся разработкой Ruby on Rails. Мне нравится быть независимым разработчиком программного обеспечения, и у меня это хорошо получается. Фактически хобби, которым я увлекался, превратилось в профессию.

Из мечтающего стать архитектором студента я превратился в серьезного технического специалиста, увлекающегося фотографией. Основам фотоискусства меня научила бабушка, а родители всячески поддерживали мое хобби. В итоге, сколько себя помню, рядом со мной всегда была камера. Она стала атрибутом моей жизни. Более того, невыпущенная программа, которую я писал для себя после ухода из *Sun*, была предназначена для работы с фотографиями.

В 2005 году, через десять лет после того как я прекратил учиться на архитектора и переключился на разработку программного обеспечения, мне позвонили друзья из группы, занимающейся организацией конференций для O'Reilly. Им требовался человек, который мог бы запечатлеть проводимое мероприятие, и меня спросили, не хочу ли я прийти и сделать несколько снимков. Я принял предложение, но несколько вышел за рамки данных мне инструкций. Я с удовольствием снял все важные заседания и с изумившей всех быстротой выложил фотографии на сайт Flickr. Меня снова пригласили на аналогичное мероприятие, и в течение следующих четырех лет благодаря большому количеству клиентов мне удалось открыть свое дело.

Сейчас, когда я пишу эти строки, я все еще программирую время от времени и даже занимаюсь программным обеспечением некоторых клиентов. Но по большей части я зарабатываю как фотограф. Впрочем, ситуация вполне может

**ДЕРЖИ НОС ПО ВЕТРУ (продолжение)**

снова измениться. Заранее этого никогда не знаешь. Сложно сказать, что принесет нам будущее.

Я всего лишь ловлю возможности. Увидев нечто интересное и вызывающее восторг, я включаюсь в это дело и предпринимаю все, чтобы не упустить свой шанс. Обычно это означает приобретение новых навыков и новых способностей. Некоторым это может показаться тягостным, но по какой-то причине мне нравится учиться чему-то новому. В конце концов, новые способности дают новые возможности. И я никогда не характеризовал себя, описывая навыки. Мне кажется, куда лучше меня характеризует то, что я уже сделал и собираюсь сделать. А навыки — всего лишь способ достижения цели.

## Действуй!

1. Найди работу, которая тебе по-настоящему интересна.
2. Начиная со следующего понедельника на протяжении двух недель веди журнал. Каждый будний день, проснувшись, отмечай уровень энтузиазма по десятибалльной шкале: 1 означает, что ты предпочел бы заболеть, лишь бы не идти на работу, а 10 — что высокая температура не может удержать тебя в постели, так тебя захватила идея по поводу решения очередной задачи.

   Через две недели внимательно посмотри на результаты. Есть ли на графике пики? Какие тенденции он показывает? Он прижат вниз или стремится ввысь? Какую бы ты поставил себе оценку, если бы это был школьный тест?

   На протяжении следующих двух недель каждое утро думай над тем, каким образом ты собираешься сегодня работать на 10. Планируй *сегодняшнее* рабочее расписание так, чтобы на следующий день ты не мог дождаться момента, когда приступишь к работе. Каждый день фиксируй *вчерашний* уровень энтузиазма. Если и после двух недель ничего коренным образом не поменяется, имеет смысл задуматься о смене работы.

# ЧАСТЬ II
# ИНВЕСТИРУЯ В СВОЙ ПРОДУКТ

Я не хвастаюсь, утверждая, что обладаю прирожденным даром игры на саксофоне. Но вам придется поверить мне на слово, что зачастую это только мешает. Работая полный рабочий день, я много играл. Порой у меня бывало по два, а то и по три концерта в день. Утро начиналось с джаза во время завтрака, затем я переходил к танцевальной музыке на свадьбе, и заканчивалось все ритм-энд-блюзом на вечеринке в ночном клубе. Благодаря своему прирожденному дару к игре на саксофоне, я смог учиться и совершенствоваться прямо на рабочем месте. Я очень хорошо чувствовал мелодию, мог выучить песню со слуха и имел склонность к импровизации.

Но именно это помешало мне *вложиться в себя* как в саксофониста. Все получалось достаточно легко, и я был доволен. Кроме того, для групп, в которых мне приходилось играть, я был, как правило, «палочкой-выручалочкой», соответственно особого давления со стороны коллег тоже не наблюдалось.

Я не понимал, что потихоньку останавливаюсь в развитии. В юном возрасте я быстро рос, но чем больше становилось отыгранных ритм-н-блюзовых концертов, тем однообразнее я играл. Мои мелодии ночь за ночью звучали одинаково. Мои импровизированные соло были переделками сыгранного предыдущей ночью или даже ранее на этом же концерте. Сейчас я понимаю, что дело было не только во мне. Влияла общая ситуация на профессиональной сцене. Мы были нетребовательны к себе, а публика была нетребовательна к нам (ты когда-нибудь слышал, чтобы публика разразилась аплодисментами и ободрительными криками из-за того, что саксофонист держит ноту более тридцати секунд?).

В течение нескольких лет до недавнего времени мой рабочий график был столь плотным, что занятия музыкой отошли на задний план. Я попросту забросил свои саксофоны и гитары. Но внезапно я почувствовал, как мне всего этого не хватает, и решил предпринять серьезные шаги для возвращения в этот мир. Однако у меня уже не было ни местных друзей-музыкантов, ни времени на профессиональную игру, да и ловкость рук была в изрядной степени утрачена. Поэтому оставалось играть для себя.

Возможно, все дело в том, что я постарел. Или поумнел, хотя в этом я сомневаюсь. Но теперь я понял, насколько *большое значение* могут

иметь даже небольшие инвестиции в себя. Вместо того чтобы взять инструменты и отправиться на очередной концерт, я (в обязательном порядке) играл сам для себя. Это позволило мне более целенаправленно подойти к инструментам. Я слушал музыку и составлял список приемов, которые неплохо было бы изучить. Скажем, я всегда хотел делать такие же штуки, которые показывал в своих соло на саксофоне Фил Вудс. Или узнать, каким образом Принс в конце песни «Let's Go Crazy» из альбома *Purple Rain* заставляет свою гитару так визжать.

И оказалось, что в сочетании с природным талантом эти несколько часов инвестиций в себя помогли воплотить в жизнь все мои «всегда хотел уметь так делать». Как только я начал вкладываться в себя, одно начало цепляться за другое. Изученная техника вела к следующей, упражнение, ломающее барьеры, давало мотив к новым свершениям.

Через несколько месяцев таких целенаправленных стараний я во многих отношениях стал играть лучше, чем когда бы то ни было, лучше, чем я играл, когда музыка была моей работой. Сконцентрированный и вкладывающийся в свои способности (пусть даже в виде хобби), я наголову превзошел себя, полагающегося на природные талант и способности.

Это отличное доказательство того, что для получения отличного *продукта*, способного продаваться на рынке труда, — продукта, который выделяется из общей массы и успешно конкурирует с другими, — тебе придется как следует потрудиться. В бизнесе идей и даже талантов пруд пруди. Поэтому, чтобы твой продукт *действительно начал чего-то стоить*, его придется полить кровью, потом, слезами и деньгами.

В этой части книги мы рассмотрим стратегии инвестиций в карьеру. Мы поговорим о том, как выбрать навыки и технологии, в которые имеет смысл вкладываться, и *какими способами* мы можем вложиться в себя. Именно с этого момента начнется реальная работа.

## Совет 11
## Учимся ловить рыбу

Лао-цзы сказал: «Дай человеку рыбу, и он будет сыт целый день. Научи его ловить рыбу, и он будет сыт всю жизнь». При всей верности

этой поговорки Лао-цзы не учел, что бывают люди, не желающие учиться ловить рыбу. Такой человек предпочтет на следующий день попросить еще одну рыбу. В процессе обучения задействованы как учитель, так и ученик. Но многие из нас зачастую не испытывают желания учиться.

Что в индустрии ПО считать *рыбой*? Процесс работы с инструментом, некий аспект технологии или некую информацию из бизнес-отрасли, в которой ты работаешь. Умение проверить определенную часть системы управления исходным кодом, с которой работает твоя команда, или настроить сервер приложения. Многие из нас считают все эти мелочи само собой разумеющимися и считают, что *с ними обязательно справится кто-то другой*. В системе управления версиями разбирается парень, отвечающий за сборку проекта. Достаточно попросить его все настроить, когда возникнет такая необходимость. Группа, отвечающая за поддержку инфраструктуры, умеет настраивать брандмауэры между вами и заказчиками. Поэтому, если появится такая нужда, достаточно отправить им письмо по электронной почте, и они обо всем позаботятся.

> Не жди, пока тебе расскажут. Спрашивай сам!

Кто хочет зависеть от милостей другого человека? Или поставим вопрос более жестко: ты бы согласился нанять на работу человека, зависящего от *целой группы специалистов*? Лично я отвечаю на этот вопрос отрицательно. И предпочитаю сотрудников, способных к самостоятельным действиям.

Очевидно, что первым делом следует изучить инструменты, необходимые для выполнения непосредственных обязанностей. Например, такой мощный инструмент, как система управления версиями. Он позволяет разработчикам повысить продуктивность своего труда. Это не просто место для кода, с которым ты закончил работать, ни в коем случае не следует воспринимать эту систему подобным образом. Она является неотъемлемой частью разработки. И столь важная вещь — надежное хранилище твоей работы — не должна быть для тебя загадкой. Самодостаточный разработчик легко обнаружит разницу между той версией проекта, над которой он работает, и последней корректной версией из хранилища. А представь, что тебе нужно взять

последний выпущенный код и исправить там ошибку. Обнаружив существенную ошибку посреди ночи, ты вряд ли захочешь кому-то звонить и просить дать тебе корректную версию, чтобы приступить к устранению неполадок. Все то же самое можно сказать о средах разработки, операционных системах и практически каждом фрагменте инфраструктуры, на которой базируется твой код.

Не менее важной является используемая тобой технологическая платформа. Например, ты можешь разрабатывать приложения с помощью J2EE. Тебе приходится создавать различные классы, интерфейсы и дескрипторы развертывания. Можешь ответить на вопрос: *зачем*? Знаешь, как применяются все эти вещи? Что происходит при запуске J2EE-контейнера? Даже если ты не являешься разработчиком сервера приложений, знание всех этих вещей позволит тебе писать надежный код для этой платформы и устранять неисправности, когда что-то идет не так.

Особенно располагают к лени различные модули, генерирующие за тебя код. Они весьма распространены среди Windows-разработчиков, так как *Microsoft* создала множество инструментов, предельно упрощающих решение многих задач. Оборотной стороной этого удобства стало появление множества разработчиков, не имеющих понятия о принципах работы своего кода. Происходящее внутри того или иного модуля остается для них мистической тайной. Не пойми меня неправильно — генерация кода может быть весьма полезной. Ведь именно она позволяет перевести высокоуровневый код C# в байтовый код, запускаемый в среде исполнения .NET. Не думаю, что тебе понравилось бы писать такой код вручную. Но применение высокоуровневых мастеров и модулей не способствует углублению твоих знаний. В итоге твои навыки ограничиваются умением работать с имеющимся инструментарием.

В бизнесе, с которым связана наша деятельность, также можно легко проглядеть рыбу. Например, сотрудник ипотечной компании может попросить специалиста рассчитать процентную ставку для всех необходимых при тестировании сценариев, а может рассчитать ее самостоятельно. Хотя взаимодействие с клиентом приветствуется, как и согласование с ним бизнес-требований (в противовес недопониманию и самостоятельным попыткам нарисовать полную картину),

только представь, насколько быстрее ты смог бы двигаться вперед, разбираясь в специфике бизнеса, на который работаешь. Разумеется, весь деловой регламент ты вряд ли выучишь, да это и не нужно. Но ничто не мешает разобраться в основах. Вспоминая лучших программистов, с которыми мне довелось поработать за прошедшие годы, могу сказать, что многие из них разбирались в бизнесе лучше, чем некоторые из их заказчиков. Это улучшило результаты их труда. Не знакомый с особенностями конкретного бизнеса человек легко допускает глупые ошибки, которых легко можно было бы избежать, обладая базовыми представлениями о том, в какой сфере ты работаешь. Более того, такой человек выполняет свои обязанности медленнее (и в конечном счете стоит компании дороже), чем аналогичный разработчик, разбирающийся в бизнесе.

Мы, разработчики программного обеспечения, можем перефразировать высказывание Лао-цзы так: «Попроси рыбу, и будешь сыт один день. Попроси кого-то научить тебя ловить рыбу, и будешь сыт всю жизнь». Но лучше не *просить* кого-то, а взять и научиться самостоятельно.

### Действуй!

1. *Как и почему?* Прямо сейчас после прочтения данного раздела или в следующий раз на работе подумай о тех аспектах своей деятельности, которые ты не до конца понимаешь. Для погружения в непонятную сферу можно задать себе два крайне полезных вопроса: *Как это работает?* и *Почему это происходит* (*или должно происходить*)*?*

   Возможно, ты даже не сможешь на них ответить, но сам факт постановки подобных вопросов приведет твой ум в новое состояние и поможет лучше осознать твое рабочее окружение. *Как IIS-сервер передает запросы твоим ASP.NET-страницам? Зачем генерировать все эти интерфейсы и дескрипторы развертывания для твоих EJB-приложений? В чем разница в работе компилятора при динамическом и статическом подключении? Почему для заказчиков из Монтаны сумма налога вычисляется по-другому?*

   Разумеется, ответ на любой из этих вопросов даст тебе потенциальную возможность снова задать подобный же вопрос. Как только вопросы *как* и *почему* иссякнут, можно считать, что ты достаточно углубился в тему.

2. *Немного времени*. Выбери в своем инструментарии важный, но игнорируемый инструмент и сосредоточься на нем. Это может быть твоя система управления версиями, библиотека, которой ты активно пользуешься, но не имеешь представления о том, как она устроена, или даже редактор, применяемый при написании кода.

   Ежедневно выделяй немного времени, чтобы *изучить очередной аспект* выбранного инструмента, позволяющий повысить продуктивность твоей деятельности или лучше контролировать среду разработки. Например, ты можешь выбрать для изучения командную оболочку Bourne Again Shell операционной системы GNU. Когда в очередной раз твой ум начнет уходить в сторону от рабочих задач, вместо загрузки привычного новостного сайта поищи в интернете *советы по bash*. В течение минуты-другой ты обязательно найдешь *новую для себя полезную информацию* по работе с этой оболочкой. А узнав новый прием, начинай препарировать его с помощью вопросов *как* и *почему*.

## Совет 12
## Разберись, как работает бизнес

В предыдущей теме мы обсудили важность осознанного выбора бизнес-отрасли для будущей работы. К этому аспекту следует относиться серьезно, так как знание специфики работы в определенной отрасли является фактором, способным серьезно повлиять на возможность твоего трудоустройства. Но перед тем, как вкладываться в изучение деталей бизнеса, следует убедиться, что ты делаешь правильную с собственной точки зрения и с точки зрения состояния рынка инвестицию.

Однако всегда существует актуальный комплекс знаний, не относящийся ни к технике, ни к определенной отрасли — это основы финансирования бизнеса. В какой бы сфере ты ни работал, будь это производство, здравоохранение, некоммерческая деятельность или образовательное учреждение, эта сфера все равно относится к *бизнесу*. А *бизнес* сам по себе является областью, которую можно и нужно изучать.

Помню, как я, будучи молодым программистом, приходил на корпоративные встречи, и мои глаза стекленели, когда важный руководитель, с которым я никогда напрямую не работал, начинал демонстрировать какие-то диаграммы с огромным количеством цифр. Мне казалось, что все это не имеет ко мне совершенно никакого отношения. Я маялся и жаждал *вернуться к себе, чтобы закончить работу над очередной подсистемой приложения*. Коллеги из моей группы вертелись, как дети, истомленные долгой автомобильной прогулкой. Никто из нас не понимал смысла презентации и никого она не интересовала. Вину за эту, как нам казалось, пустую трату времени мы возлагали на некомпетентных менеджеров, организовавших данную встречу.

Оглядываясь назад, я понимаю, как глубоко мы заблуждались. Мы работали для бизнеса, и наша деятельность должна была помогать зарабатывать или экономить деньги. При этом мы даже не представляли, что делает бизнес доходным. И даже хуже — это казалось нам не важным. Мы были программистами и системными администраторами. Мы считали, что наша работа связана исключительно с темами, которым мы себя посвятили. Но разве может *творчески* помогать процветанию бизнеса человек, не понимающий, как этот бизнес *функционирует*?

> Ты не сможешь помогать бизнесу творчески, если не знаешь, как он устроен.

Ключевым в предыдущем абзаце является слово *творчески*. Проще всего считать, что мы специалисты в IT и нам платят именно за это. Корректно поставленные задачи и правильное руководство позволяют сосредоточиться на вопросах, которые способствуют развитию бизнеса. И нам не нужно понимать, как работает бизнес, чтобы приносить ему пользу.

Но *творчески* приносить пользу можно, только имея представление о бизнесе, с которым связана твоя деятельность. В деловом мире то и дело звучат слова «*чистая прибыль*». Многие ли из нас действительно понимают, что такое чистая прибыль и от каких факторов она зависит? И, что более важно, многие ли из нас действительно понимают, какие их действия дают вклад в чистую прибыль? Является ли твое подразделение центром затрат или центром формиро-

вания прибыли (уменьшают или увеличивают твои действия чистую прибыль)?

Понимание финансовых стимулов (и языка) компании, в которой ты работаешь, даст тебе возможность действовать осмысленно, а не блуждать в темноте, принимая решения, кажущиеся интуитивно правильными.

**Действуй!**

1. Найди книгу по основам бизнеса и прочитай ее внимательно. Ищи среди книг для специалистов по управлению предприятиями. Могу порекомендовать тебе книгу Стивена Силбигера «MBA за 10 дней». Я считаю ее весьма полезной (и приятно лаконичной). Ее можно прочитать всего за десять дней, а это не очень большие затраты времени.
2. Попроси кого-нибудь рассказать о финансовых показателях твоей компании или отдела (конечно, если это открытая информация).
3. Попробуй повторить эти объяснения.
4. Разберись, почему чистая прибыль называется *чистой*.

## Совет 13
## Найди наставника

В музыкальной джазовой культуре существует такая правильная вещь, как практика наставничества. В мире джаза общепринято, что молодые музыканты ищут более опытных коллег, которые возьмут их под крыло и познакомят с джазовой традицией. Более того, этим дело не ограничивается. Старые музыканты часто консультируют по поводу карьеры, дают советы в различных жизненных ситуациях и слушают, как играет молодежь. А та платит им своей беззаветной преданностью, обеспечивает поддержку и группы горячих поклонников творчества.

Эти отношения формируют связи и дают музыкантам возможность устроиться на работу. Вокруг отношений наставник/подопечный

в мире джаза были созданы самоорганизующаяся культура и целый набор традиций. Вся эта система работает столь хорошо, что возникает впечатление, будто ею кто-то управляет.

В профессиональной деятельности (особенно в сфере IT) просить чужой помощи не принято. Зачастую зависимость от другого человека считается признаком слабости. Мы боимся признать собственное несовершенство. Все пронизано духом конкуренции. Выживает сильный, и это закон. К сожалению, этот закон мешает развиться институту наставничества. Спроси я у группы джазовых музыкантов «Кто ваш наставник?», большинство из них ответит без проблем. А как будут реагировать на подобный вопрос программисты? Произвольно взятый программист из Соединенных Штатов, скорее всего, удивится: «Что?»

Так было не всегда. Западная история включает в себя развитую систему профессионального наставничества, уходящую корнями в Средневековье. Подход к профессиональному обучению был еще более сильным и формализованным, чем принятая в современном музыкальном сообществе система. Молодые люди начинали свою карьеру подмастерьями признанных мастеров. Работали за символическую плату и привилегию учиться у мастера. А тот, в свою очередь, был обязан обучить подмастерьев созданию вещей в собственной манере (и собственного качества).

> В зависимости от другого человека нет ничего страшного. Главное — убедиться, что это правильный человек.

Основной целью такого подхода является превращение наставника в объект для подражания. Некоторые вещи расцениваются как невозможные, пока кто-то не расширит наши границы привычного. Объект для подражания задает стандарт качества. К примеру, можно считать себя приличным игроком в шахматы, выигрывая у своих знакомых. Но попытайся принять участие в турнире по шахматам, и ты обнаружишь, что это куда более сложная игра, чем изначально казалось. Попытка же сыграть партию с реальным мастером станет для тебя еще большим откровением. Победы над родственниками и знакомыми могут создать у тебя иллюзию, что ты *хорошо умеешь*

играть в шахматы. Без примера для подражания сложно найти стимул к совершенствованию.

Кроме того, наставник структурирует процесс твоего обучения. Как ты мог понять из предыдущих разделов, существует ошеломляющее количество технических и деловых сфер, в изучение которых можно вложиться. Но слишком широкий выбор порой парализует. В разумных пределах лучше двигаться вперед, чем сидеть на одном месте. Наставник может помочь тебе с выбором направления, на котором имеет смысл сосредоточиться.

Устроившись специалистом по поддержке систем, я взял в оборот Кена — святого человека, который работал в нашем университете одним из сетевых администраторов. Он помог мне решить серьезную проблему с NetWare-сетью нашего кампуса, проблему, из-за которой студенты не могли пользоваться компьютерным классом, и после этого отделаться от меня ему было уже сложно (впрочем, он и не пытался). Когда я спросил у него, в каком направлении следует двигаться, чтобы стать более компетентным и независимым, он дал мне простой рецепт: разберись со службами каталогов, научись работать с их UNIX-вариантом и как следует изучи язык сценариев.

Он выбрал для меня три практических навыка из бесконечного числа доступных вариантов. Убежденный, что рецепт мне выписывает настоящий мастер своего дела, я принялся за учебу. Эти три навыка заложили фундамент моей профессиональной деятельности и остаются актуальными по сию пору. И дело не в том, что Кен дал мне единственно правильный ответ — подобных ответов просто не существует. Важно, что он сократил длинный список того, что я *мог бы* изучать, до краткого списка того, что я реально *изучил*.

Еще наставник служит доверенным лицом, которое наблюдает и оценивает принимаемые тобой решения и твои успехи. Если ты программист, покажи ему свой код и слушай советы. Собираясь сделать презентацию в офисе или на местной встрече пользователей, сначала покажи ее наставнику и учти его комментарии и замечания. Наставник — это лицо, которому ты доверяешь настолько, чтобы задать вопрос: «Что я как специалист должен в себе изменить?» Ведь ты знаешь, что услышишь от него не только критику, но и советы, помогающие стать лучше.

В конце концов, как и в джазе, не только у тебя возникает личная привязанность к наставнику и ответственность перед ним, но и наоборот. Помогая другому человеку, ты вкладываешься в его успех. Ты продвигаешь его в профессиональном плане по пути, который считаешь правильным. И если этот путь приведет к успеху, это будет и твой успех тоже.

У наставника возникает стимул способствовать успеху своего подопечного. При этом сам он как более опытный и уже достигший успеха, как правило, пользуется уважением изрядного числа людей. Наставник становится надежным связующим звеном между тобой и своими знакомыми. Важность таких связей переоценить невозможно. В конце концов, клише «Связи решают всё» появилось не на пустом месте.

Формальности в отношениях с наставником не имеют особого значения. Вовсе не обязательно в явной форме просить другого человека взять над тобой шефство (хотя сделать это не помешает). Но наставник может даже не *знать* о своей роли. Важно, чтобы рядом с тобой был человек, которому ты доверяешь и которым восхищаешься, помогающий правильно сориентироваться в плане профессионального роста и как следует отточить нужные навыки.

### Действуй!

1. *Воспитай себя сам*. В идеале было бы здорово найти человека, который будет активно тебя направлять, но это далеко не всегда реализуемо на практике. Вот способ, позволяющий стать наставником самому себе.

   Подумай, кем из профессионалов, занятых в твоей сфере деятельности, ты восхищаешься больше всего. Большинство из нас на определенном этапе трудового пути уже имеют небольшой список таких людей. Это может быть кто-то из сослуживцев или просто человек, работой которого ты восхищаешься. Выпиши десять наиболее важных характеристик этого образца для подражания. В список должны войти характеристики, из-за которых ты выбрал этого человека образцом для подражания. Это могут быть конкретные варианты навыков, например широкая эрудиция в области некой технологии или глубокие знания какой-то одной сферы. Не стоит забывать и про личные черты, такие как умение идеально вписываться в команду или выдающиеся ораторские навыки.

Расположи эти характеристики в порядке возрастания важности, чтобы на первом месте оказалось наименьшее важное, а на десятом — наиболее важное качество. Этот список достойных восхищения и важных атрибутов показывает направление, в котором нужно двигаться, чтобы стать похожим на выбранную ролевую модель. Но как понять, на чем следует сосредоточиться в первую очередь?

Добавь к списку еще один столбец и подумай, как выбранный тобой пример для подражания оценил бы *тебя* по десятибалльной шкале (10 — наивысшая оценка). Попытайся посмотреть на себя глазами другого человека.

Затем вычти свой рейтинг, указанный в последнем столбце, из уровня важности характеристики. Скажем, если свой рейтинг по самой важной характеристике ты оценил как 3, итоговый приоритет будет равен 7. Расположи пункты списка по убыванию итоговых приоритетов, и ты получишь перечень областей, в которых тебе требуется совершенствование.

Начни с верхних двух или трех пунктов и составь список конкретных задач, которыми *нужно будет заняться прямо сейчас*, чтобы в итоге стать лучше.

## Совет 14
## Стань наставником

Если хочешь по-настоящему что-то изучить, попробуй научить этому кого-то другого. Нет лучшего способа обобщить свое понимание вопроса, чем заставить себя объяснить другому человеку непонятные моменты так, чтобы он все понял. Простое проговаривание является давно известным средством прояснения мысли. Выступления перед куклами и другими неживыми объектами как способ решения задач давно уже вошли в фольклор разработчиков программного обеспечения.

Я слышал, как Мартин Фаулер (Martin Fowler)[1], выступая перед разработчиками в Бангалоре, сказал, что когда он хочет как следует что-то изучить, он об этом пишет. Мартин является известным разработчиком программного обеспечения и автором книг. Его можно назвать одним из самых выдающихся и влиятельных *преподавателей* в своей области, если рассматривать его книги как средство дистанционного обучения.

[1] Нет, мы не родственники.

Мы учимся, обучая. Это звучит странно, ведь предполагается, что учитель уже знает свой предмет. Разумеется, я не имею в виду изучение нового материала путем его преподавания — откуда человек сможет узнать этот материал? Но знание фактов не означает понимания их причин и следствий. Именно такое, более глубокое понимание мы развиваем в себе, уча других. Для объяснения сложных понятий ищутся аналогии, и мы проводим внутреннюю работу, объясняя себе, почему одна аналогия кажется рабочей, но на самом деле совершенно не подходит, в то время как другая, с виду менее удачная, объясняет суть. Преподавателю порой приходится сталкиваться с вопросами, которые ни за что не возникли бы у него сами по себе. Работа учителем позволяет выявить и устранить пробелы в собственных знаниях.

> Чтобы понять, на самом ли деле ты хорошо разбираешься в теме, попробуй объяснить эту тему кому-то еще.

А значит, ты можешь получить выгоду, не только найдя себе наставника, но и *став* наставником для кого-то еще.

Положительные последствия наставничества проявляются и в социальной сфере. Пересекающаяся группа наставников и их учеников формирует устойчивую и мощную социальную сеть. Связь учитель—ученик очень сильна, и в профессиональной среде подобные связи куда крепче, чем более отвлеченные знакомства. Отношения наставничества формируют лояльность друг к другу. Именно в сетях подобного рода лучше всего рассматривать сложные задачи или заниматься поиском работы.

Не следует также недооценивать тот факт, что помогать людям *приятно*. Если мысленно вы можете завладеть вниманием публики, имеет смысл воспользоваться этой способностью в альтруистических целях. В неопределенности современной экономической ситуации реальная *помощь* другим — это работа, с которой тебя не могут уволить. И платят за нее валютой, которая не обесценивается с инфляцией.

Для поиска учеников вовсе не нужно расхаживать с умным видом и объявлять себя знатоком. Здесь помогут только твоя компетентность и желание терпеливо делиться знаниями. Не стоит переживать

об отсутствии доскональных знаний по теме. Скорее всего, у тебя есть *некий опыт*, который даст возможность помогать менее опытным товарищам. Определи, в какой сфере ты накопил достаточно опыта, и начинай приносить пользу.

Например, ты много и плодотворно работал на ниве PHP. Можно посетить местное собрание группы PHP-пользователей и предложить неопытным пользователям помощь в решении их проблем. Если же возможность давать очные консультации отсутствует, можно отвечать на вопросы на специальных форумах или IRC-каналах либо помогать с решением проблем, возникающих при работе с приложениями. Но помните, что наставничество — это прежде всего работа с людьми. Отношения, возникающие при оказании помощи в сети, никогда не сравнятся с отношениями, зародившимися при личном общении.

Как правило, наставники не попадают под сокращение.

Для получения всех описанных преимуществ вовсе не нужно устанавливать формальные отношения учитель—ученик. Просто начинай помогать людям, а остальное получится естественным образом.

**Действуй!**

1. Найди человека, которого можно взять под свое крыло. Это может быть более молодой и неопытный коллега, возможно, стажер. Еще можно договориться с отделом информатики и информационных систем в местном университете и взять на себя работу со студентами.
2. Найди в интернете форум и выбери тему. Начни помогать. Приобрети репутацию человека, который хочет и может терпеливо помогать другим.

## Совет 15
## Практика, практика и еще раз практика

В студенческие времена я проводил долгие ночи в здании своего факультета. Сквозь тонкие стены классов до меня постоянно доноси-

лись самые отвратительные звуки, которые только можно себе вообразить. И дело не в том, что в моей школе учились плохие музыканты. Совсем наоборот. Просто они упражнялись.

Музыкальные упражнения и *не должны* звучать хорошо. Если во время занятий ты всегда играешь хорошо, это означает, что ты не пытаешься выйти за пределы своих возможностей. Но ведь упражнения предназначены именно для этого. В спорте происходит то же самое. Спортсмены до предела напрягаются во время тренировок, *расширяя* пределы возможного для реальных выступлений. Все некрасивые моменты происходят за закрытыми дверьми, а не когда начинается настоящая работа.

В отрасли, связанной с компьютерами, часто встречаются разработчики, действующие на пределе своих умений. К сожалению, как правило, это означает, что их квалификация недостаточна для решения поставленных задач. Но при этом существует тенденция практиковаться непосредственно на рабочем месте. Представь себе музыканта, который, выйдя на сцену, воспроизводит ту же какофонию, что и в классе. Кто будет терпеть подобное? Музыканты получают деньги за *выступления* перед публикой, а не за отработку навыков. Аналогичным образом мастер боевых искусств или боксер, тренирующийся до изнеможения во время подготовки к соревнованиям, далеко в спорте не уйдет.

Как отрасль мы должны выделить время для практики. Мы на Западе часто приводим доводы в пользу отечественных программистов, базируясь на относительно высоком качестве производимого ими кода. В сравнении с тем, что пишут их зарубежные коллеги. Но чтобы стать конкурентоспособными по качеству, нужно перестать относиться к работе как к месту практики. Следует *инвестировать время* в свое ремесло.

Несколько лет назад я начал экспериментировать с упражнениями по программированию, которые я смоделировал после практических занятий музыкой. Первое правило состояло в том, что разрабатываемые мной программы не предназначались для последующего использования. Я не хотел халтурить, чтобы быстрее добраться до цели, но у меня получались неработоспособные в реальной жизни программы.

Я не работал спустя рукава, но был страшно разочарован тем, что многие из приходящих мне в голову идей оказались неэффективны. Хотя я старался сделать свою работу как можно более качественно, проектное решение и код то и дело получались совсем не такими элегантными, как мне хотелось.

Сейчас, оглядываясь назад, я понимаю, что испытываемое мной во время этих упражнений чувство неловкости было *хорошим признаком*. Продуцируемый мной код порой содержал выдающиеся фрагменты. Но я продолжал напрягать свои ментальные мускулы и вырабатывать собственные стандарты. Все как в случае с игрой на саксофоне. Если бы я, начав упражняться, стал играть только приятную для слуха музыку, это означало бы, что тренировки не происходит. Точно так же во всех моментах элегантный код, выходящий из-под моего пера, указывает, что я уселся где-то в центре своих текущих способностей, а вовсе не на их границе, где происходит реальная тренировка.

> Тренируйся на пределе своих способностей.

Но как понять, что именно следует отрабатывать? Что расширяет твои границы? Тема наработки навыков, необходимых разработчику программного обеспечения, потянет на отдельную книгу. Я первым делом прибегаю к моему опыту джазового музыканта. Тренировки имеет смысл поделить на три категории (я специально упрощаю для читателей, не имеющих отношения к музыке):

- физические упражнения/координация;
- игра с листа;
- импровизация.

Этот список может послужить основой *одного из вариантов* тренировки для разработчиков программного обеспечения.

*Физические упражнения/координация*. Музыканты должны нарабатывать технику обращения с инструментом: генерация звука, координация движений (например, легкость перемещения пальцев), скорость и точность — все это крайне важно наработать.

Какие эквиваленты этим вещам существуют у разработчиков программного обеспечения? Скажем, как обстоят дела с неясными моментами твоего основного языка программирования, на которые ты редко обращаешь внимание? Поддерживает ли выбранный язык регулярные выражения? Ведь это мощная и катастрофически игнорируемая функция многих программных сред. Зачастую разработчики не пользуются ими просто потому, что им не хватает знаний. Результатом становится многострочный код, который потом требуется поддерживать.

Эти же правила применимы к API или к библиотекам функций выбранного тобой языка. Не научившись работать с многочисленными инструментами окружения, ты вряд ли сможешь применить их в ситуации, когда они действительно потребуются. Попробуй, к примеру, разобраться, как функционирует в выбранной среде разработки многопоточное программирование. А как у тебя обстоят дела с библиотеками потоков ввода-вывода, API сетевого программирования или наборами служебных программ для работы с коллекциями или списками? Большинство современных языков программирования предлагает богатые и мощные библиотеки для всех упомянутых областей, но разработчики предпочитают освоить только небольшую часть из них, что негативно влияет на эффективность написания кода.

*Чтение с листа*. Для профессиональных музыкантов, записывающихся в студии, способность играть с листа имеет первостепенное значение. Однажды пришлось принять участие в записи рекламного ролика для *Blockbuster* (она специализируется на прокате видеокассет). Большой оркестр играл быструю мелодию, а я не только солировал на саксофоне, но и исполнял партию второго альта. В первый раз я увидел ноты в момент начала записи. Сначала мы сыграли солирующую партию, потом — сопровождающую. При малейшей ошибке все музыканты были бы вынуждены начать с начала, и не забывайте про стоимость аренды студии.

Что для разработчика программного обеспечения означает умение читать с листа код? Или технические требования? Или проектное решение? Искать код для подобных упражнений лучше всего в сообществах разработчиков ПО с открытым исходным кодом. У тебя

есть любимые фрагменты таких программ? Не хочешь попробовать добавить в них какую-нибудь функцию? Прочитай список задач для программы, с которой ты хотел бы поработать, и определи конкретную дату реализации новой функциональной возможности (или хотя бы прикинь, сколько времени может занять эта реализация).

После этого загрузи код программы и приступай к его анализу. Как понять, куда следует смотреть? Какими приемами ты пользуешься для поиска конкретного места в большом фрагменте кода? В какой точке ты начнешь работу?

Такое упражнение можно делать на время. При этом ты вовсе не *обязан* реально реализовывать выбранную функциональную возможность. Используй это как отправную точку. На самом деле ты учишься с максимальной скоростью распознавать, что делает код, на который ты смотришь. Для каждого следующего упражнения выбирай новую программу. Попробуй выполнить его с разными типами программы, написанными в разных стилях и на разных языках. Отмечай моменты, которые облегчают или затрудняют понимание происходящего. Какие стандартные действия помогают тебе в работе с кодом? Какие виртуальные «хлебные крошки» ты оставляешь себе, чтобы упростить ориентирование при перемещении вверх и вниз по стеку вызовов сложной функции?

*Импровизация*. Импровизацией называется взятие некой структуры или ограничения и мгновенное создание на его основе чего-то нового. Как программист, я обнаружил, что чаще всего импровизирую в моменты стресса. *Ох, нет! Сервер приложений в беспроводной сети перестал работать, и мы теряем заказы!* Именно в такие моменты у меня возникают самые творческие экспромты. Я делаю сумасшедшие вещи, такие как восстановление потерянных данных вручную путем повторной передачи по беспроводной сети пакетов из бинарного файла системного журнала. Никто не рассчитывает, что ты способен на подобные вещи, особенно под горячую руку. Такая способность к резким и быстрым решениям напоминает приходящую на помощь в нужный момент магическую силу.

Отличным способом заострить свой ум и улучшить способности к импровизации при написании кода является практика с самостоятельно наложенными ограничениями. Выбери простую программу

и попытайся переписать с учетом ограничений. Моим любимым упражнением является написание программы, выводящей на экран строки старой песни «99 бутылок пива». Ты можешь это сделать, обходясь без присвоения переменных? Или попробуй минимизировать размер этой программы. В качестве дополнительного ограничителя может выступить время. Насколько *быстро* ты в состоянии написать код? Как насчет решения небольших сложных задач с таймером в руках?

Это всего лишь один из вариантов практики. Примеры можно добыть в любой области, от изобразительного искусства до религиозных практик монахов. Важно понять, в каких именно вещах тебе требуется тренировка, и, разумеется, никогда не тренироваться во время выступлений (на работе). Для практики нужно выделить *отдельное время*. Это только *твоя* ответственность.

### Действуй!

1. *TopCoder*. TopCoder.com — это сайт корпорации, проводящей соревнования по спортивному программированию. Ты можешь зарегистрироваться там и участвовать в соревнованиях с призами. Для тех, кто не любит соревноваться с другими, на сайте есть раздел с набором задач, которые послужат прекрасной основой для отработки практических навыков. Регистрируйся и начинай заниматься.

2. *Code Kata*. Один из самых прагматичных программистов, наш любимый издатель Дэйв Томас (Dave Thomas), превратил идею наработки программистских навыков в… кое-что прагматичное. Его *кодовая ката* (code kata) — набор небольших, но провокационных упражнений. Программисты могут выполнять их на любом языке. Каждая ката делает упор на определенную технику или мыслительный процесс, нагружая один из твоих ментальных мускулов.

   На момент написания этой книги Дэйвом была создана двадцать одна ката. Все они доступны в его блоге (http://codekata.pragprog.com/). Там же можно найти список рассылки и чужие решения этих упражнений, а также обсуждения способов решения.

   Твоя задача — выполнить всё. Записывай результаты в дневник или веди блог. Закончив дело, напиши *собственную* кату и поделись ею с широкой публикой.

# Совет 16
# Подход к работе

«Разработка программного обеспечения» — это не некая вещь, а *процесс* создания некой вещи. При написании кода важно сосредоточиться не только на разрабатываемом продукте, но и на самом процессе разработки. Отвлекаясь от процесса, мы рискуем опоздать со сроками выполнения задания, получить некачественный продукт или не получить вообще ничего. Ни один из этих вариантов заказчика не обрадует.

К счастью, процесс создания хорошего программного обеспечения (и в целом продукции) давно как следует обдуман. И изрядная часть этой информации легла в основу целой группы *методик*. Они описаны в книгах, доступных как в Сети, так и в обычном книжном магазине.

К сожалению, зачастую разработчики не пользуется преимуществами, которые дает данная информация. В большинстве команд процесс является делом второстепенным или даже навязываемым сверху. Слово *методика* с их точки зрения является синонимом бумажной работы и долгих бессмысленных собраний. И слишком часто методика превращается в нечто, навязываемое им руководством.

Интуитивно руководство понимает необходимость следования *определенному* процессу, но зачастую понятия не имеет о доступных в настоящее время вариантах. В результате начальник отряхивает пыль с процедур, которым его научили в 1980-е, пересказывает их с использованием модных словечек и учит всему этому подчиненных. И пока кто-нибудь не прервет порочный круг, исследовав, что работает, а что не очень, ситуация повторяется, как только очередной разработчик дорастает до руководящей должности.

Очевидно, должен существовать лучший способ разработки программного обеспечения. И у большинства групп он есть.

Как программист, тестировщик или дизайнер ПО ты можешь считать, что сам по себе процесс разработки не входит в сферу твоей ответственности. Возможно, это правильно, так как ты всего лишь наемный работник. Но, к сожалению, обычно эта ответственность

*повисает в воздухе*. И даже если ее на кого-то возлагают, она передается «группе организации процесса» или другому отдельному подразделению. Но дело в том, что для успешного внедрения метода разработки программного обеспечения его должны принять те, кто будет им пользоваться, — такие, как ты.

Лучшим способом ощутить ответственность за подход к работе является участие в его внедрении. Если в фирме, где ты работаешь, производственный процесс отсутствует, изучи методики, которые могут тебе помочь. Во время обеденных перерывов обсуждай с другими членами группы проблемы текущего процесса разработки и варианты смягчения этих проблем путем перехода к некоему стандарту. Составь план внедрения выбранного тобою рабочего процесса и получи всеобщее одобрение. Затем приступай к реализации этого плана.

Чтобы почувствовать причастность к процессу разработки, помоги с его внедрением.

Возможен и вариант, когда ты работаешь в фирме, где производственный процесс спускается сверху. Здесь зачастую, когда инструкция добирается до группы разработчиков, к описанию технологических приемов добавляют столько лишних слов и интерпретируют их столь вольно, что они теряют свой первоначальный смысл. Инструкцию постигает судьба фразы в игре Испорченный телефон[1]. Тут ты снова можешь взять на себя инициативу. Изучи спущенную сверху методику и помоги своей группе и руководству корректно интерпретировать описание рабочего процесса. Если ты не собираешься бороться против спускаемых сверху инструкций, по крайней мере обеспечь корректность их практической реализации.

Мир методик быстро начинает напоминать наполненную трескучими словами полую раковину. Но если привыкнуть к подобной манере изложения, рассматривая процесс разработки программного обеспечения, всегда можно найти *какую-то* полезную информацию — даже если это информация о том, как *не* нужно делать. Человек, хорошо разбирающийся в способах организации рабочих процессов, более убедительно обосновывает принципы, в соответствии с которыми работает его группа.

---

[1] https://ru.wikipedia.org/wiki/Испорченный_телефон

### МЕТОДИКИ: НЕ ТОЛЬКО ДЛЯ ЗАНУД

Управление проектами далеко не всегда имеет отношение к методам разработки программного обеспечения, но ты можешь оказаться первым в своей компании, кто решил изучить данную тему. Многочисленные методики управления проектами широко используются во всей отрасли. Вероятно, самым известным является Свод знаний по управлению проектами (Project Management Book of Knowledge)[1] от Института управления проектами (с его общепризнанной программой сертификации).

Шесть сигм (Six Sigma)[2] — еще одна качественная методология, не связанная, впрочем, напрямую с программным обеспечением. Подход, продвигаемый такими монстрами, как *General Electric* и *Motorola*, придает особое значение измерению и анализу процессов и продукции для улучшения качества обслуживания клиентов и роста эффективности. Этот не имеющий отношения к разработке программного обеспечения, строгий и методичный подход предлагает данные, непосредственно применимые к работе программиста.

[1] http://pmbok.narod.ru/

[2] http://www.six-sigma.ru/

Несмотря на обилие нормативных методик, вряд ли ты когда-нибудь попадешь в компанию, полностью внедрившую какую-нибудь из них. И это нормально. Самым лучшим рабочим процессом считается тот, который обеспечивает наибольшую продуктивность группы и выпуск самой лучшей продукции.

Единственным способом создания подобного гибрида (ну, кроме божественного откровения) является изучение доступных вариантов и выбор оттуда фрагментов, приемлемых для тебя и твоей команды, которые будут постоянно совершенствоваться на основе реального опыта.

В конечном счете, если ты не в состоянии организовать рабочий процесс, ты не сможешь производить продукцию. Найти человека, который сможет писать программы, куда проще, чем человека, умеющего *организовать* процесс написания программ. Поэтому имеющиеся в твоем арсенале сведения о принципах организации процесса разработки ПО будут тебе только в плюс.

**Действуй!**

1. Выбери методику разработки программного обеспечения и найди посвященные ей ресурсы. Начни читать книги и сайты, присоединись к списку рассылки. Посмотри на эту методику критически. Выдели с твоей точки зрения сильные и слабые стороны. Что мешает внедрить ее на твоем рабочем месте? Изучи аналогичным способом еще одну методику. Сравни достоинства и недостатки обеих методик. Можешь ли ты скомбинировать предлагаемые подходы?

## Совет 17
## На плечах гигантов

Будучи джазовым музыкантом, я часто слушаю музыку. И это не фоновая музыка, звучащая, когда я читаю или веду машину. Я в нее *вслушиваюсь*. Ведь джазовая импровизация представляет собой воспроизведение того, что ты слышишь кроме аккордов песни, и поэтому вдумчивое слушание является важным источником вдохновения и сведений о том, что работает, а что не очень. Что звучит здорово, а что всего лишь приемлемо.

Огромное количество записей, сделанных за историю существования джаза, представляет собой невероятный комплекс знаний, которым может воспользоваться любой, обладающий умением слушать. А значит, прослушивание музыки для джазового музыканта — не пассивная деятельность. Это обучение. Более того, умение понимать, что именно ты слышишь, — это навык, развивающийся со временем. Мои друзья-музыканты для этого устраивают специальные встречи. Мы сидим и слушаем джаз, а потом обсуждаем услышанное. Иногда мы играем в *игру*, в которой один из нас исполняет импровизированное соло, а остальные должны по стилю понять, какая запись послужила основой импровизации.

Разумеется, подобные мероприятия происходят не только у джазовых музыкантов. Классические композиторы делают то же самое. И даже писатели и поэты. Скульпторы и художники. Изучение работ мастеров является важным компонентом становления нового мастера.

Прослушав джазовую запись, мы обсуждаем, какими музыкальными устройствами пользовался импровизатор для передачи своей мысли. «Класс! Ты слышал, как он начал отступать от основной темы в конце этой формы?» или «Как-то странно он играл на фоне этого бита во время перехода». Эти обсуждения помогают нам открыть и понять приемы, которые можно вставить в следующую импровизацию.

Дизайн и программирование ПО в этом отношении во многом похожи на различные области искусства. В поисках шаблонов и трюков приходится анализировать огромные объемы готового кода. Образцы разработки, с которыми можно познакомиться в книге «Приемы объектно-ориентированного проектирования. Паттерны проектирования» (*Design Patterns: Elements of Reusable Object-Oriented Software*) появились как попытка обнаружить и документировать допускающие многократное использование решения часто возникающих при разработке программного обеспечения проблем. Они формализовали изучение существующего кода, сделав его доступным большому количеству профессионалов в области ПО. Тем не менее это всего лишь небольшой фрагмент обучающих практик, которыми мы можем воспользоваться посредством чтения кода.

> Чтобы проникнуть в суть, анализируй чужой код.

Какими алгоритмами пользуются другие программисты для решения отдельно взятых задач? Как со стратегической точки зрения они именуют переменные, функции и структуры? Если бы я захотел реализовать протокол мгновенных сообщений Jabber на другом языке, как это можно было бы сделать? Какие нестандартные способы существуют для управления взаимодействием UNIX- и Windows-процессов? Изучение чужого кода позволит тебе найти ответы на эти и другие подобные вопросы.

Чужой код не только позволяет найти ответы на конкретные вопросы, но и служит своего рода увеличительным стеклом для рассмотрения собственного стиля и способностей. Прослушивание записей Джона Колтрейна (John Coltrane) всегда напоминает мне о моем уровне игры на саксофоне, аналогичный смирительный эффект несет чтение трудов выдающихся разработчиков программного обеспечения. Но это нужно не только для того, чтобы вспомнить про свое место. В про-

цессе чтения их кода обнаруживаются вещи, с которыми ты сам вряд ли когда-либо справился бы. Ты может обнаружить даже то, о чем ты, скорее всего, никогда не додумался бы. Почему? О чем именно думал этот разработчик? Какой была его мотивация? Подобное придирчивое и углубленное изучение результатов чужого труда позволяет учиться даже на примере плохого кода.

В мире искусства легко учиться на результатах чужого труда, так как ни картина, ни музыка не имеют скрытого исходного кода. Слушая музыку или рассматривая картину, ты можешь учиться. К счастью, разработчики программного обеспечения имеют доступ к практически бесконечному набору программ с открытым исходным кодом.

Используй чужой код для оценки собственных способностей.

Доступное количество таких программ столь велико, что вряд ли их все можно прочитать. Разумеется, попадаются среди этого изобилия и плохие проекты, тем не менее нам доступно и довольно много *великолепных* примеров. Существует открытый код, реализующий практически любую решаемую программно задачу и почти на всех доступных языках программирования.

Критически анализируя этот код, ты постепенно начнешь вырабатывать собственный вкус, как это бывает в музыке, живописи и литературе. Различные стили и приемы тебя позабавят, удивят, рассердят и (надеюсь) *стимулируют* к работе. Ищущий найдет тут всё — от трюков, повышающих продуктивность работы, до парадигм проектирования, полностью меняющих подход к целому классу проблем. Как и в искусстве, изучая особенности чужих работ и учась на них, ты выработаешь свой ни на кого не похожий стиль разработки ПО.

Положительным побочным эффектом чтения кода станет осведомленность о существующих видах программ. Получив новое задание, ты сможешь вспомнить: «Вот в этом и в этом проектах я видел библиотеку, опирающуюся на обработку MIME-типа». И если позволяют условия лицензионного соглашения, ты можешь сэкономить свое время и деньги фирмы, изучив особенности реализации уже готового решения. Возможно, ты удивишься, осознав, сколько денег в индустрии ПО снова и снова тратится на повторное изобретение колеса (впрочем, слово *изобретение* звучит в данном случае слишком сильно).

Сэр Исаак Ньютон сказал: «Если я видел дальше других, то лишь потому, что стоял на плечах гигантов». Такие умные ребята, как Исаак, знают, что мы многому можем научиться у тех, кто был до нас. Будь таким же.

**Действуй!**

1. Найди проект и прочитай его как книгу. Конспектируй по ходу чтения. Выдели сильные и слабые стороны. Напиши критический обзор и опубликуй его. Найди в этом проекте хотя бы один прием или шаблон, которым ты сможешь в дальнейшем пользоваться. Найди хотя бы один пример того, как делать не надо.

   Найди группу единомышленников для ежемесячных встреч. На каждой встрече один из членов группы должен предлагать для разбора код длиной от 2 до 200 строк. Группа должна проанализировать и обсудить его. Подумайте, какие решения принимались при его написании и что можно было бы добавить.

## Совет 18
## Автоматизация задач

Моя карьера постоянно сопровождалась конфликтами между желанием руководства нанять для работы над проектами бюджетную (зачастую заграничную) консалтинговую компанию и моей уверенностью, что самый дешевый разработчик далеко не всегда гарантирует низкие затраты. Я много спорил с директором по информационным технологиям и вице-президентом, увлеченно доказывая, что лучше нанять несколько сильных разработчиков вместо толпы неквалифицированных, хотя и дешевых, кодеров.

К сожалению, меня часто прерывали на полуслове. И проблема была вовсе не в моей неправоте (это очевидно!). Но простого способа доказать мою правоту не существует. А с точки зрения затрат единственные имеющиеся у нас объективные данные заставляли сделать вывод о выгоде найма сотрудников с более низкой почасовой оплатой.

Представь себе гипотетический проект по созданию программного обеспечения для какой-либо сферы по твоему выбору. Сколько про-

граммистов потребуется, чтобы написать такую программу за три месяца? Говоришь, пять? Шесть? (Потерпи минутку.) Хорошо. А как насчет выполнения этого проекта за два месяца? Как ты уберешь целый месяц?

Руководство отделов информационных технологий, как правило, заявляет, что для ускорения процесса следует нанять дополнительных программистов. Это неправильно, но люди так считают. И раз можно ускорить один проект, увеличив число исполнителей, значит, экстраполируя эту тенденцию, получим, что продуктивность прямо пропорциональна количеству рабочего персонала.

Но достичь поставленной цели можно несколькими способами. Для увеличения объемов производства программного обеспечения можно:

- нанять тех, кто будет работать быстрее;
- нанять *дополнительных* работников;
- автоматизировать работу.

Но так как мы пока не знаем, как корректно измерить продуктивность разработки программного обеспечения, сложно доказать, что один человек работает быстрее другого. И именно поэтому руководители финансовой службы предпочитают сосредоточиться на почасовой оплате.

Это дает простую формулу, привязанную к фиксированному периоду времени:

$$\text{Продуктивность} = \frac{\text{Количество проектов}}{\text{Количество программистов} \times \text{Почасовая оплата}}.$$

В некоторых сферах деятельности действительно можно посчитать реальный доход от инвестиций в программное обеспечение. Но в большинстве случаев речь пойдет о таких нечетких показателях, как *число проектов* или *количество требований*, без контролируемого способа их измерения.

Итак, получается, что мы не в состоянии доказать преимущество *более способных программистов*, но при этом *не хотим нанимать много дешевых программистов*. Значит, остается только вариант с автоматизацией.

Я помню сенсационную безработицу в США в 1980-х. В то время мы возлагали вину не только на другие страны, но и на машины, а особенно на компьютеры. На предприятиях стали появляться гигантские роботы-манипуляторы. Эти роботы настолько превосходили людей в производительности и точности, что состязаться с ними не было никакой возможности. Расстроены такой ситуацией были все, кроме *создателей* этих роботов.

Представь, что ты работаешь в компании, которая создает сайты для малого бизнеса. По сути, тебе приходится раз за разом проделывать одну и ту же работу, добавляя контактную информацию, опросы, корзины и прочие аксессуары. Можно нанять либо несколько очень продуктивных программистов, которые будут делать за тебя сайты, либо целую армию дешевых программистов, которые будут вручную делать одно и то же. Либо создать систему, *генерирующую* сайты.

Введя в формулу нашего финансового директора некоторые (вымышленные) цифры, мы получим показанное на рис. 1 уравнение.

Быстрые программисты:

$$\frac{5}{3\times\$80}=0{,}02.$$

Дешевые программисты:

$$\frac{5}{20\times\$12}=0{,}02.$$

Один программист + робот:

$$\frac{5}{1\times\$80}=0{,}06.$$

**Рис. 1.** Сравнение производительности

Автоматизация является неотъемлемой частью нашей отрасли. Но по каким-то причинам мы пока не хотим автоматизировать *наш* труд разработчиков программного обеспечения. Как *гарантированно* создавать программы быстрее и дешевле зарубежных конкурентов? Нужно сделать роботов. Автоматизируй свою работу!

## ОТ КОНСУЛЬТАНТА ПО ВОПРОСАМ ИНФОРМАТИЗАЦИИ ДО ГЕНЕРАЛЬНОГО ДИРЕКТОРА

*Вик Чадха, генеральный директор Enterprise Corp.*

Путь от консультанта по информационным технологиям в *General Electric* до должности entrepreneur-in-residence[1] в *bCatalyst* (бизнес-акселератор с фондом в 5 миллионов долларов) — вовсе не та карьера, о которой я мечтал.

Каким образом я перешел от работы в фирме, имеющей тысячи сотрудников и, по версии журнала «Fortune», входящей в пятерку крупнейших в США, в фирму, занимающуюся инвестициями и консультированием недавно созданных компаний, специализирующихся в области высоких технологий? Попытавшись оглянуться назад и нарисовать полную картину, я обнаружил важные закономерности, которыми хотел бы с вами поделиться. Возможно, вы сможете применить их к собственной ситуации.

Вскоре после получения степени магистра в области электроники и вычислительной техники в Политехническом университете Виргинии я устроился консультантом в *General Electric*. Набирало обороты использование интернета в коммерческих целях, и я работал над несколькими проектами, предназначенными для извлечения максимальной выгоды из этой невероятно мощной платформы и лежащих в ее основе технологий. Произошел стремительный переход из IT-отдела финансового департамента в группу технологий и сервисного обслуживания, затем — в группу отдела продаж, занимающуюся автоматизацией, и наконец — в группу складского учета. В каждой группе я занимался разработкой новых программ. Я обожал исследование и последующее внедрение, изучение новых сетевых технологий и их применение к решению сложных коммерческих задач.

Работа с новейшими технологиями далеко не всегда приносила приятные чувства. Постоянно возникали проблемы из-за их недостаточной проработанности. Приходилось тратить время и силы, помогая нашим поставщикам отлаживать продукцию. Как заказчик я понял, что как бы заманчиво ни выглядела та или иная технология, ее ценность состоит только в способности решать актуальные

[1] Должность в фирмах из сферы венчурного капитала, юридических фирмах, бизнес-школах, которую занимает опытный предприниматель, назначенный компанией с венчурным капиталом, университетом или другой организацией. Человек, отвечающий за запуск новых компаний, помощь в оценке потенциальных инвестиций и функциональную экспертизу для помощи с текущим инвестированием. — *Примеч. пер.*

## ОТ КОНСУЛЬТАНТА ПО ВОПРОСАМ ИНФОРМАТИЗАЦИИ ДО ГЕНЕРАЛЬНОГО ДИРЕКТОРА (продолжение)

проблемы, обеспечивая измеримую выгоду. Постепенно я осознал, что фокусироваться следует не на технологиях, а на решениях. Через несколько лет, когда я уже работал в компании *bCatalyst*, это очень пригодилось мне при оценке новых технологических проектов.

Но какой бы замечательной ни была моя работа в *General Electric*, кое-чего мне все-таки не хватало. Как специалист по информационным технологиям я в основном развивался в одном направлении без возможности понять, как функционируют фирмы, на чем они делают деньги, что обеспечивает их стабильность, каким образом осуществляются инновации. Но я не расстроился, а предпочел взять инициативу в свои руки, чтобы больше узнать о бизнесе и предпринимательстве.

Я не учился бизнесу, поэтому единственным способом узнать специфику открытия новой компании стала практика (словом, я предпочел учиться методом проб и ошибок).

Вместе с моим предприимчивым бывшим соседом, а ныне близким другом (Раджем Хаджелой) и моей женой (Видаей) я занялся мозговым штурмом, направленным на выявление неудовлетворенных потребностей на рынке. Нас привлекала электронная коммерция, но заниматься продажей товаров широкого потребления мы не хотели. Мы интересовались искусством и имели соответствующий опыт. Плюс ко всему привлекал тот факт, что каждое произведение искусства уникально по своей природе. Мой дядя-художник всю свою жизнь испытывал трудности с заработком. Исследования показали, что с этой проблемой сталкивается большинство художников. Появилась идея платформы, позволяющей художникам публиковать и рекламировать свои работы, поддерживая связь с постоянными покупателями. В результате появился сайт Passion4Art.com, а мы занялись сложным делом поиска художников, которые согласились бы там зарегистрироваться и предоставить цифровые копии своих картин. После регистрации первой тысячи пользователей, открывших у нас собственные страницы, выгода нашего предложения стала очевидной, поэтому мы занялись поисками внешнего финансирования.

В это же время (примерно в 1999 году) сайт eMazing.com ежедневно публиковал советы на самые разные темы. Мы подумали, что могли бы с ним скооперироваться (работая с нашими художниками и их каналами сбыта) и публиковать у них совет дня по искусству. С нами встретился один из их руководителей, оценил наше предложение и согласился попробовать.

### ОТ КОНСУЛЬТАНТА ПО ВОПРОСАМ ИНФОРМАТИЗАЦИИ ДО ГЕНЕРАЛЬНОГО ДИРЕКТОРА (продолжение)

После рассказа о поиске субсидий для развития инфраструктуры он любезно порекомендовал отправить наш бизнес-план в недавно появившийся в городе бизнес-акселератор *bCatalyst*.

Через несколько дней нам позвонил генеральный директор *bCatalyst* Кит Вильямс и сказал, что хотел бы встретиться и подробнее узнать о нашей компании. Разумеется, мы крайне обрадовались. Лишь гораздо позже я понял, насколько важным оказался тот факт, что информация о нас поступила из доверенного источника. Поэтому если вам когда-нибудь потребуется связаться с венчурным фондом, приложите все усилия для получения хороших рекомендаций, так как они являются наилучшим средством достижения нужного вам результата.

Несколько встреч с Китом позволили установить доверительные отношения между нашими командами, но из-за недавно лопнувшего пузыря доткомов время для инвестиций в данную сферу было неподходящим. На последней встрече мы узнали, что наша команда им очень понравилась, но ее финансирование является неоправданным.

Впрочем, нас заверили, что если мы придем с другой идеей, которая им понравился, проблем с финансированием не будет. Я спросил, следует ли воспринимать эти слова как вежливый отказ или они действительно собираются с нами работать. Представители *bCatalyst* уверили нас в серьезности своих намерений.

Тогда я попросил еще об одной встрече с Китом и сказал, что собираюсь уйти из *General Electric*, чтобы на следующие несколько месяцев посвятить себя работе в его команде и совместному исследованию возможностей для открытия бизнеса. Я объяснил, что они ничем не рискуют, ведь я не прошу никаких долговременных гарантий (по аналогии с условно-бесплатным программным обеспечением). В итоге меня взяли на работу, потому что я смог убедить в своем желании поставить на карту все, уйдя из *General Electric* даже при отсутствии четких гарантий на будущее.

В течение следующих двенадцати месяцев мы ежедневно встречались с разными командами, представлявшими нам свои идеи, и я обнаружил закономерность в ответах на вопросы, которые мы им задавали.

Я составил список и предлагаю вам с ним ознакомиться на случай, если вам когда-нибудь потребуется получить деньги от венчурной компании http://www.enterprisecorp.com/resources/assessment.htm.

**ОТ КОНСУЛЬТАНТА ПО ВОПРОСАМ ИНФОРМАТИЗАЦИИ ДО ГЕНЕРАЛЬНОГО ДИРЕКТОРА (продолжение)**

Навыки, полученные за год работы в компании *bCatalyst*, привели меня на должность генерального директора *Enterprise Corp*. За семь прошедших лет мне довелось работать с более чем ста компаниями, которым я помог получить около 75 миллионов долларов. Я испытываю глубокое удовлетворение от своей работы, которую никогда бы не получил, не рискни я попробовать себя в совершенно новой области. Подобный процесс невозможен без крутых поворотов. Надеюсь, что вы, читатель, сможете использовать мою историю как источник вдохновения для поиска собственного уникального пути, который позволит в полной мере задействовать ваши способности.

### Действуй!

1. Выбери повторяющуюся задачу, с которой тебе часто приходится сталкиваться, и напиши для нее генератор кода. Начни с простого. О возможности многократного использования пока можно не беспокоиться. Просто сделай так, чтобы генератор экономил твое время. Подумай, как поднять уровень абстракции генерируемого кода.
2. Исследуй архитектуру, управляемую моделями (Model-Driven Architecture, MDA). Попробуй поработать с доступными инструментами. Посмотри, где в твоей работе можно применить если не весь инструментарий, то хотя бы некоторые *концепции* MDA. Подумай о применении этих концепций к ежедневно используемым тобой инструментам.

# ЧАСТЬ III
# ИСПОЛНЕНИЕ

Ты сделал правильные инвестиции и правильно выбрал рынок. К примеру, ты становишься специалистом по внедрению сервис-ориентированной архитектуры для приложений по доставке пиццы с поддержкой беспроводной связи, и службы доставки начинают процветать, как никогда раньше. Но перед тем, как предаться самолюбованию, послушай мое предостережение. Все, о чем мы говорили до этого, — не более чем подготовка. Но рано или поздно наступит момент, когда нужно будет *сделать что-то* реальное.

Конечно, бывают случаи исключительного везения, но, как правило, никто не будет платить тебе просто потому, что ты умен. Не дадут тебе денег и за то, что ты являешься ведущим экспертом в новейшей отрасли. Ты работаешь в компании, которая, скорее всего, пытается делать деньги. И твоя задача — делать *что-то*, что помогает в осуществлении этого намерения. Все предварительные размышления и подготовительные задания готовили тебя к возможности показать себя в деле и начать плодотворную работу на благо компании.

Подобно парню, желающему «стать J2EE-разработчиком», о котором шла речь в совете № 9, большинство из нас не отождествляет себя с фирмами, дающими нам работу. Я имею в виду, что я прежде всего программист и только потом человек, помогающий компании *Fortune 500* продавать посудомоечные машины. Я разработчик архитектуры приложений, а не служащий энергетической компании. Это неудивительно, если смотреть на создание программного обеспечения как на ремесло. Выбранное нами ремесло обычно никак не связано с отраслью, в которой мы его применяем. Архитектор, проектирующий офис для военного ведомства, остается архитектором, а не превращается в военного подрядчика.

Подобное сохранение идентичности приводит к некоторым проблемам, так как наши макроцели могут войти в противоречие с нашими микрообязанностями. Если архитектор создает офис, не подходящий для нанявшего его военного подрядчика, ценность его работы равна нулю. И как бы красиво ни выглядело его творение, это плохой архитектор.

Нам платят за создание ценностей. Отсюда следует, что пора вылезти из удобного кресла и приняться за реальные дела. На одних способностях далеко не уедешь. На финишную прямую выходят только *лучшие* — те, кто умеет доводить дело до конца.

Доводить дела до конца — *здорово*. Часто людям сложно приучить себя к систематической работе (посмотри значение термина *прокрастинация*), но как только ты ощутишь огонь воодушевления, останавливаться уже не захочешь. Так давай зажжем этот огонь.

## Совет 19
## Прямо сейчас

Представь, что ты принимаешь участие в конкурсе с призом 100 000 долларов наличными. Приз достается команде, успевшей первой создать программу для получения новых неоплаченных счетов. Твоя рабочая группа подала заявку на участие. Работа должна быть выполнена за выходные. Код должен обеспечить реализацию минимального набора указанных функциональных возможностей, и его следует полностью протестировать. Вы приступаете в субботу утром. Выигрывает команда, которая до утра понедельника первой предоставит готовое приложение. Если до этого момента ни одна из команд не закончит работу, приз достанется тому, кто реализует максимальное количество функциональных возможностей.

Ты уверенно читаешь технические требования. Глядя на перечень функциональных возможностей системы, ты понимаешь, что масштабом и параметрами она напоминает те многочисленные системы, с которыми тебе приходилось работать раньше. И в то время как целью группы является завершение проекта к середине воскресенья, ты начинаешь задавать себе вопрос: *каким образом приложение, по масштабу аналогичное тем, над которыми ты работал в офисе неделями, может быть закончено за одни выходные?*

Что можно сделать прямо сейчас?

Но как только наступает время приняться за работу и ты принимаешься писать код, становится понятно, что группа достигнет поставленной цели. Все в ударе, реализуют одну функциональную возможность за другой, исправляют ошибки, за доли секунды принимают проектные решения, и работа оказывается выполненной. Это потрясающее ощущение. В офисе во время анализа проектов и планерок

ты часто предавался мечтам о том, как здорово было бы с небольшой командой оторваться от бюрократической среды и сотворить новое приложение в рекордные сроки.

Об этом мечтают многие. Мы *знаем*, что рабочий процесс нас тормозит. И не только он. Скорость окружения также негативно влияет на сроки выполнения.

Закон Паркинсона гласит: «Работа заполняет все отпущенное на нее время». Печально то, что даже не желающие подчиняться этому закону могут попасть в ловушку, особенно при наличии задач, решать которые нет никакого желания.

В случае соревнования, длящегося всего пару дней, у тебя попросту нет времени на откладывание задач. Ты не можешь задержаться с принятием решения и не задерживаешься. Ты не можешь избежать скучной работы и знаешь, что все нужно сделать настолько быстро, что ни одна задача не успеет тебе надоесть *слишком сильно*.

Закон Паркинсона является результатом наблюдений, а не неизбежным приговором. Ощущения неотложности, даже надуманного, достаточно, чтобы легко удвоить, а то и утроить твою продуктивность. Попробуй и сам убедишься. Ты можешь справиться с делом быстрее. Можешь сделать все прямо сейчас. Можешь закончить работу, а не обсуждать, как ее закончить.

Восприняв рабочий процесс не как пребывание в тюремной камере, а как соревнование, ты сможешь завершить его куда быстрее. Создавай движение. Становись тем, кто толкает вперед. Не расслабляйся.

Всегда будь тем, кто спрашивает: «А что мы можем сделать *прямо сейчас*?»

### Действуй!

1. Посмотри на список стоящих перед тобой задач. Найди задачи, попавшие туда в незапамятные времена, проекты, которые уже начали покрываться плесенью, или те, выполнение которых несколько *застопорилось* — из-за бюрократии или неоправданно больших затрат на анализ и проектирование.

Выбери задачу, решением которой ты можешь *заняться* в рабочий перерыв, в то время, которое обычно тратится на просмотр сайтов в интернете, проверку почты или долгие перекусы. Преврати многомесячный проект в задачу, которую нужно решить меньше чем за неделю.

## Совет 20
## Читай чужие мысли

Довелось мне работать с парнем по имени Рао. Он родился в Южной Индии, в штате Андхра-Прадеш, но жил в США и работал в нашей фирме. Рао мог превратить в код все, что вы попросите. К нему обращались, когда требовалось низкоуровневое программирование систем. Если речь заходила о высокоуровневом программировании приложений, он тоже мог удовлетворить практически любой запрос.

Но по-настоящему уникальным Рао был потому, что он делал все *до того*, как его об этом просили. Он обладал необъяснимой способностью предугадывать, какую задачу ему могут поручить, и решал ее еще до того, как начальству в голову приходила подобная идея. Это напоминало магию. Кажется, в какой-то момент я обвинил его в обмане, но было непонятно, как он это делает. Я сказал: «Рао, я решил поменять способ инкапсуляции контроллера в среде разработки приложений. Достаточно внести небольшие изменения, и мы сможем использовать эту среду не только для веб-приложений. Что ты по этому поводу думаешь?»

«Я уже сделал это на той неделе, — ответил он. — Это зафиксировано в системе управления версиями. Посмотрите». И такие вещи происходили с Рао *постоянно*. Это случалось настолько часто, что объяснить подобные совпадения можно было, только представив, что Рао проделывает *все мыслимые изменения* каждого фрагмента кода, поддержкой которого занималась наша группа.

В это время я руководил группой, отвечающей за архитектуру приложений. Кроме всего прочего, мы занимались созданием и поддержкой программных сред, на основе которых работали выпускаемые нашей компанией приложения. Мои коллеги тратили много времени на об-

суждение способов улучшения разработки программного обеспечения. Много говорилось и о роли, которую в этих улучшениях играли базовые компоненты инфраструктуры.

Именно здесь скрывалась основа магических трюков Рао. Он практически не принимал участия в этих разговорах, но очень внимательно слушал. И в отличие от настоящих фокусников он поделился со мной своим секретом. Он выполнял только те пожелания, которые я озвучивал. Но пожелания озвучивались настолько неявно, что *даже* я сам этого не осознавал.

Например, стоя в очереди за кофе, я мог рассуждать о том, как здорово было бы реализовать в коде новые более гибкие функциональные возможности. И если подобные рассуждения звучали достаточно часто или я высказывался достаточно убедительно, несмотря на отсутствие этих задач в списке текущих дел нашей группы, Рао заполнял перерывы в работе, проверяя целесообразность реализации моих пожеланий. И если это можно было легко (и дешево) внедрить, писал код и проверял его.

Телепатия применима не только к руководству, но и к заказчикам. Наводящая информация дает возможность добавлять функциональные возможности, о которых заказчик только *собирается* попросить или о которых он *попросил бы*, если бы считал, что это возможно. Всегда делая, что просят заказчики и когда они это просят, ты доставляешь им удовольствие. А сделав больше или раньше, чем тебя просят, ты восхитишь их — разумеется, если способность читать мысли тебя не подведет.

> Хороший трюк с телепатией приводит к тому, что люди начинают от тебя зависеть.

Трюк с чтением мыслей не совсем безопасен. Без страховки на этот канат лучше не ступать. Вот перечень основных рисков (с вариантами сглаживания отрицательных последствий):

- Делая работу, которую тебя никто не просил делать, ты тратишь деньги фирмы. А что, если ты ошибаешься? Начинай с малого. Реализуй свои догадки только в перерывах, чтобы это никак не отражалось на выполнении твоих рабочих обязанностей. Если

у тебя есть к этому склонность, делай дополнительную работу в свободное время.

- Каждое добавление кода в систему рискует сделать ее менее отказоустойчивой. Избегай работы, которая может повлиять на архитектуру системы или каким-то образом ограничить ее функциональность. Если влияние изменений достаточно велико, нужно принимать бизнес-решение. Разработчики практически никогда не имеют полномочий самостоятельно решать подобные вопросы.
- Модификация некой функциональной возможности в соответствии с *пожеланиями* заказчика может отрицательно сказаться на функциональности приложения в целом или дать другие нежелательные последствия. Когда дело доходит до пользовательского интерфейса, с догадками лучше быть крайне осторожным.

Руководство людьми и проектами — трудная и интересная работа. Люди, обеспечивающие бесперебойную работу над проектом без многочисленных указаний сверху, высоко ценятся постоянно занятыми начальниками и заказчиками. Правильно применяемый трюк с чтением мыслей приведет к тому, что люди начнут от тебя зависеть — отличный рецепт карьеры, направление которой ты выбираешь сам. Эту способность стоит осваивать и развивать.

### Действуй!

1. Первый рецензент этой книги Карл Брофи предлагает тебе записывать свои *соображения* по поводу возможных просьб руководства и заказчиков для твоего следующего проекта или поддерживаемой системы. Будь изобретательным. Попытайся посмотреть на систему с их точки зрения. Составляя список неочевидных функциональных возможностей, которые могут потребоваться, подумай о способах их максимально эффективной реализации. Вспомни о граничных ситуациях, о которых пользователи могут сразу не вспомнить.

   Столкнувшись с проектом или с заявками на расширение, подсчитай число своих угадываний. Какое количество записанных тобой функциональных возможностей действительно попросили реализовать? Когда о них зашла речь, смог ли ты воспользоваться идеями, рожденными во время мозгового штурма?

# Совет 21
# Ежедневные победы

Нам всем нравится верить в силу собственных знаний, в то, что мы хорошие программисты, способные легко обеспечить решение задач и достичь высот производительности. Немногие счастливчики (коим, я настаиваю, *сказочно повезло*) действительно добиваются успеха при помощи подобной стратегии.

А вот извлечь выгоду из планирования и мониторинга может кто угодно. Здравый смысл подсказывает, что для попадания в список лучших достаточно превзойти ожидания руководства. Превышение ожидания работодателя является вполне достойной целью, однако на удивление немногие из нас знают, как достичь этой цели.

Как всегда бывает с подобными задачами, в выдающегося исполнителя проще всего превратиться при наличии конкретной запланированной работы. Когда ты в последний раз выходил за пределы своих должностных обязанностей? Твой начальник об этом знал? Как сделать более заметными проявления подобного поведения?

Мой друг и коллега Джеймс Макмурри[1] на заре нашей с ним карьеры поделился разработанной им системой, которая гарантировала хороший результат работы. Система поразила меня своей глубиной, особенно если учесть небольшой опыт ее создателя (возможно, он учел уроки, полученные от родителей), и я пользуюсь ею до сих пор. Не уведомляя руководство, он фиксирует *хиты дня*. Его задача — каждый день делать нечто такое, о чем можно было бы с гордостью доложить наверх. Это может быть пришедшая ему в голову идея или некое действие, могущее улучшить работу его отдела (рис. 2).

Простое обозначение целей (на день, на неделю, на сколько можешь) и фиксация достигнутых результатов позволит быстро поменять твое поведение. Необходимость составления списка хитов естественным образом заставляет оценить свою деятельность и расставить приоритеты в соответствии с коммерческой ценностью отдельных рабочих заданий.

---

[1] Его персональный сайт http://www.semanticnoise.com

*Понедельник* — *Автоматизировать сборку!* ✓
*Вторник* — *Написать тесты для синтаксического разбора кода* ✓
*Среда* — *Разобраться с инструментами объектно-реляционного проецирования, чтобы можно было больше вообще не писать SQL-запросы* ✓
*Четверг* — *Написать сценарий развертывания веб-приложения*
*Пятница* — *Избавить проект от предупреждений компилятора*

**Рис. 2.** Одна неделя хитов

Достаточная частота фиксации хитов гарантирует отсутствие простоев: человек, который должен создавать хит каждый день, уже просто не сможет тратить две недели на *совершенствование* результатов своего труда. Такой тип мышления и работы становится привычкой. И подобно разработчикам, испытывающим энтузиазм при виде зеленой полосы в интерфейсе тестирования модулей, ты ощутишь внутреннюю потребность в каждодневных достижениях. Тебе не придется слишком заботиться о мониторинге состояния работ, так как функционирование на данном уровне больше напоминает нервный тик, чем выполнение заданий из списка, созданного в приложении Microsoft Project.

### Действуй!

1. Выбери для себя полчаса, сядь с карандашом и бумагой в тихом месте, где никто тебе не помешает. Вспомни о небольших проблемах, с которыми твоя группа сталкивается каждый день. Перечисли их на бумаге. Что это за досадные мелочи, заставляющие группу каждый день впустую тратить несколько минут, с которыми никто не может или не хочет ничего сделать?

   Какие выполняемые вручную задания текущего проекта можно было бы автоматизировать? Перечисли их. Как насчет сборки или развертывания? Есть ли там аспекты, которые нужно привести в порядок? Как уменьшить количество сбоев при сборке? Запиши все идеи, пришедшие тебе в голову.

   Выдели на подобные раздумья двадцать минут. Запиши все идеи, как хорошие, так и плохие. Не отвлекайся от процесса, пока двадцать минут не истекут. Когда список будет готов, на новом листе выпиши пять наиболее

раздражающих тебя позиций. На следующей неделе в понедельник сделай что-нибудь с первым пунктом получившегося списка. Во вторник проработай второй пункт. В среду — третий и т. д.

## Совет 22
## Помни, на кого работаешь

Легко сказать: «Позаботься о том, чтобы твои цели и твоя работа совпадали с целями нанявшей тебя на работу компании». Легко сказать, но трудно сделать, особенно если ты программист и над тобой столько организационных слоев, что ты с трудом представляешь себе работу самой компании. В начале моей карьеры мне довелось попасть в группу разработчиков программного обеспечения в крупную фирму, занимающуюся доставкой грузов. Там существовала настолько огромная иерархическая структура, что я на своем уровне ни разу не столкнулся ни с чем, что позволило бы мне проникнуть в суть бизнеса доставки грузов. Вспоминаю, как во время посещения ежеквартальных общих собраний члены нашей группы чувствовали себя совершенно разобщенными и чужими. «Что за достижение мы празднуем? Что значат все эти показатели?»

Конечно, на том этапе своей карьеры я больше интересовался построением элегантных систем и исследованием программ с открытым исходным кодом, чем сутью бизнеса по доставке грузов. (Должен признать, что и сейчас мои приоритеты *не изменились*.) Но если бы у меня тогда возникло желание согласовать свою работу с основными целями фирмы, не уверен, что знал бы, с чего начинать.

Итак, у нас есть прекрасный постулат о том, что мы должны согласовывать свою деятельность с целями компании, чтобы гарантированно влиять на конечный результат работы и все такое. Но по правде говоря, многие из нас со своего уровня просто не видят, как этого добиться. За деревьями леса не видать.

Возможно, это не наша вина. Возможно, мы хотим от себя слишком многого. Возможно, идея непосредственно влиять на конечный результат работы фирмы напоминает попытку вскипятить океан. Итак,

нам требуется более детальное представление, разбивающее океан бизнеса на отдельные доступные для кипячения лужицы.

Начинать имеет смысл с самой очевидной лужи — своей рабочей группы. Скорее всего, она достаточно мала и специализированна, чтобы ты смог составить представление о ее сущности. Скорее всего, ты понимаешь проблемы, с которыми она сталкивается. Ты знаешь, над улучшением каких аспектов она работает, будь это производительность, доход, сокращение ошибок или что-то еще. Информацию, в которой ты не уверен, можно уточнить у своего непосредственного начальства.

В конечном счете в хорошо структурированной среде цели руководства являются целями группы. Реши проблему своего начальника и ты решишь проблему группы. Кроме того, если ваш начальник придерживается аналогичного подхода, то проблемы, которые ты решаешь для него, на самом деле являются проблемами уже его начальства. И так далее, вплоть до самых верхов фирмы — генерального директора, акционеров или даже заказчиков.

Делая свою небольшую часть работы, ты вносишь вклад в достижение целей фирмы. Это позволяет почувствовать свое предназначение. И придает твоей работе смысл.

Возможно, у тебя не возникнет желания следовать подобной стратегии. «Я не собираюсь делать за него работу». Или «он просто поставит мои результаты себе в заслугу!»

Ну да. Примерно так это и работает. Том ДеМарко и Тимоти Листер пишут в книге «Человеческий фактор. Успешные проекты и команды» (*Peopleware: Productive Projects and Teams*), что роль хорошего начальника — это не подмена подчиненных, не знание, как выполнить работу всей группы, не приход на помощь в трудных ситуациях. Хороший начальник расставляет подчиненным приоритеты, следит за тем, чтобы у группы было все, что требуется для выполнения рабочих обязанностей, поддерживает мотивацию и продуктивность и в конечном счете обеспечивает нужный результат. Хорошая работа группы означает хорошую работу ее начальника.

Успех твоего начальника — это и твой успех тоже.

Если работа твоего начальника — знать и расставлять приоритеты, а не лично *выполнять* все задания, то твоя работа состоит именно

в *выполнении* всех заданий. Ты делаешь не работу начальника, а свою работу.

Если тебя волнует, кому в заслугу поставят выполненную тобой работу, вспомни, что твоя карьера (по крайней мере в этой фирме) зависит от начальника. В большинстве организаций за аттестацию сотрудников, уровень зарплаты, бонусы и продвижение по карьерной лестнице отвечает непосредственное руководство. Так что признание, которого ты ждешь, в конечном счете превращается в наличные именно начальством.

Помни, на кого ты работаешь. Тогда ты не только будешь действовать заодно с целями бизнеса, но и бизнес начнет отвечать *твоим* нуждам. Если ты собираешься достичь *высот* в исполнении своих служебных обязанностей, это даст тебе уверенность, что ты движешься в верном направлении.

**Действуй!**

1. Запланируй встречу со своим начальником. Ведь ты заинтересован в том, чтобы понять цели, которые начальник ставит перед твоей группой на следующий месяц, квартал и год. Узнай, как ты можешь помочь в их достижении. После встречи подумай, как твоя ежедневная работа согласуется с целями группы. Смотри на все через призму этих целей. На их основе определи приоритеты различных рабочих задач.

## Совет 23
## Будь на своем месте

Как руководитель могу рассказать, что больше всего меня раздражает общение с сотрудниками, стремящимися на следующую ступеньку карьерной лестницы. Ты знаешь таких парней: оказавшись во время обеда за твоим столом, он обязательно начнет разговор о том, какое повышение получил тот или иной сотрудник. Он всегда готов к распространению офисных сплетен и придерживается корпоративных политик как сюжетной линии своей любимой мыльной оперы. Он жалуется на профессиональную несостоятельность руководства и с горечью несет бремя

служебных обязанностей, будучи полностью уверенным, что справился бы с работой начальника куда лучше, чем сам начальник. Начальство просто слишком некомпетентно, чтобы оценить его потенциал.

Он считает ниже своего достоинства браться за многие задания. Избегает их, если это возможно, а если уклониться не получается, выполняет неохотно (и медленно). Он тщательно выбирает работу, которая, как ему кажется, пусть даже на подсознательном уровне, соответствует его уровню и может приблизить столь желанное ему повышение.

Поскольку мысленно подобные сотрудники трудятся на более высокой должности, со своими текущими обязанностями они справляются крайне посредственно. Я сам был в подобной ситуации много лет назад. Ненавижу стричь газоны. Это занятие заставляет тебя чесаться и потеть. Хуже всего, что оно не дает тебе заняться тем, *чем ты хочешь*. Для стрижки газонов обычно кого-то нанимают. Вот таким наемным парнем и был я в студенческие годы. Что я делал, когда *нужно* было стричь газон? Спешил. Делал все небрежно. Все время думал о том, как я закончу эту нудную работу и займусь куда более приятными вещами. Словом, результат моего труда взгляд не радовал.

> Будь честолюбив, но не показывай этого.

К счастью, в данном случае никто не наблюдал за тем, что я делаю, и не оценивал меня (хотя моя жена крайне недовольна состоянием газона перед нашим домом). Но вид газона после того, как я закончил с ним возиться, это моя и только моя проблема. Никто не вынудит меня всю жизнь работать «простым газонокосильщиком» из-за качества моей работы. А вот в сфере информационных технологий подобное поведение может привести к карьерной катастрофе.

Вернемся к парню, описанному в начале этого раздела. Как, с твоей точки зрения, будет смотреть на него руководство? Поймет ли начальник, что недооценивал его гениальность и такой бриллиант нуждается в продвижении? Дадут ли ему повышение, просто чтобы сделать его счастливее?

Разумеется, нет. Он — посредственный исполнитель с плохими психологическими установками. Так что *даже если* он и обладает вы-

соким потенциалом, он его никак не демонстрирует. Его потенциал не приносит фирме дохода. Акционеры не будут держаться за свои инвестиции, если их ожидания не оправдаются. Более того, такая позиция вызовет у руководства желание прекратить вкладываться в *подобного сотрудника*.

Именно так видит ситуацию начальство. Тут я, разумеется, должен признать свою вину. До некоторой степени я сам проявлял описанные черты. С той стороны все выглядит не лучше. Ты все время чего-то хочешь. Это стремление не дает тебе почувствовать себя довольным. Проснувшись утром, ты должен тащиться на «проклятую работу», где никто не подозревает о твоих гениальных способностях. С раздражением ты несешь бремя служебных обязанностей, продумывая стратегию продвижения по карьерной лестнице. Ты фантазируешь, как бы *ты* поступил на месте своего начальника в ситуации, когда тот допустил ошибку, уверенный, что вышел бы из положения куда лучше. Ты откладываешь свою жизнь в трудовом коллективе до тех времен, когда сможешь работать на должности, которой заслуживаешь.

Здесь есть один секрет: это чувство не покинет тебя никогда. Когда ты наконец получишь желанное повышение, о котором столько мечтал, то быстро ощутишь усталость и поймешь, что это вовсе не *та* работа, для которой ты предназначен, — твое место *еще* выше. И все повторится. Я еще не достиг вершины, но сильно подозреваю, что существуй такая должность и займи я ее, я оглянулся бы назад и понял, что гонялся за призраком. Все мои карьерные устремления прошли впустую.

Но разве у людей не должно быть честолюбия? Разве появились бы компании *Microsoft* или *General Electric*, будь их основатели лишены честолюбия и не имей они далеко идущих целей?

Разумеется, это нужное качество. Я не защищаю апатию и бездеятельность. Наличие целей и жажда преуспеть — это здорово. Но вспомни обиженного парня, с описания которого начался этот раздел. Думаешь, ему суждено преуспеть? Это кажется парадоксальным, но при концентрации на настоящем продвижение к цели осуществляется куда продуктивнее, чем при концентрации на самой цели.

Сначала это сложно понять. Отказ от насущного стремления к преуспеванию звучит как аскетическая недосягаемая цель. Однако со

временем ты поймешь прагматичность такого подхода. Концентрация на настоящем позволяет получать удовольствия от маленьких побед в ежедневной рабочей рутине: чувство хорошо сделанной работы, чувство, что тебя привлекают как эксперта по важным экономическим проблемам, чувство принадлежности к сплоченному рабочему коллективу. Все это пройдет мимо тебя, если ты будешь мысленно витать в облаках. Ты до бесконечности будешь ждать *великих свершений*, игнорируя повседневные мелочи, которые и придают смысл твоим походам на работу.

Не только *ты сам* почувствуешь себя лучше, это почувствуют и окружающие. Твои коллеги, начальство и клиенты. Это отразится на твоей работе. Как бы парадоксально ни выглядел этот постулат, но избавившись от желания преуспеть, ты станешь куда более *способным* к успеху.

Ты вплотную работаешь с клиентами. Рядом с тобой находятся руководители и ответственные лица, формирующие твою карьеру в краткосрочной, а возможно, и в долгосрочной перспективе. Разработчики из Филиппин или Индии лишены подобного преимущества. Поэтому *будь* там, где ты *есть*.

### Действуй!

1. Забудь на неделю о своих карьерных устремлениях. Составь список планов, связанных с *текущей* работой. Думай не о том, на какое место ты хочешь попасть после повышения, а о задачах, которыми придется заняться, завершив текущие дела. Что выдающегося ты можешь сделать на текущей должности? Напиши стратегический и тактический планы. В течение недели используй разработанную тактику для достижения долгосрочной цели — «завершить» выбранную работу.

   Сосредоточь вокруг этой цели разговоры с коллегами во время обеденных перерывов. Избегай бесед о продвижении по службе и офисной политике. Избегай сплетен.

   В конце недели подведи итоги своим попыткам достичь заданных целей. Сколько времени займет завершение всех заданий, которые тебе нужно выполнить в текущей должности? Как понять, что все закончено? Распланируй следующую неделю и повтори всю процедуру.

# Совет 24
# Великолепная задача на сегодня

Хорошо сделать свою работу и получить за это высокую оценку приятно. Интуитивно большинство из нас это понимает, но когда дело доходит до реального приложения усилий, позволяющего выделиться, мы демонстрируем удивительную избирательность. Мы не нарадуемся дизайну проекта технологического прорыва для отдела сбыта или мобилизуем все силы, чтобы спасти положение в случае явной катастрофы, потому что наши мозги расценивают подобные моменты как возможность показать, на что мы способны. Мы даже старательно и кропотливо работаем всю ночь над задачами, которые в обычном состоянии быстро нас утомляют. Зачастую экстренная ситуация заставляет нас проявлять свои лучшие качества.

Во время самых ужасных провалов и сорванных сроков только пьянящее чувство восторга давало мне энергию и силы для результативной работы. Но почему без давления внешних факторов мы не в состоянии вызвать в себе это альтруистическое и в высшей степени продуктивное безумие? Насколько производительным ты мог бы стать, если бы сумел подойти к самым неинтересным и опостылевшим задачам с тем же лихорадочным желанием сделать все правильно?

> Насколько интересной ты мог бы сделать свою работу?

Последний вопрос лучше переформулировать. Насколько *интереснее* стала бы твоя работа, если бы ты мог подойти к самым неинтересным и опостылевшим задачам с тем же лихорадочным желанием сделать все правильно? Когда нам интересно, мы лучше справляемся со своими обязанностями. А если интерес отсутствует, нам скучно, в результате страдает качество работы.

Как сделать скучную работу более интересной? При взгляде с противоположной стороны ответ на этот вопрос становится очевидным. Почему скучная работа столь скучна? Почему она *изначально* не интересна? В чем разница между работой, которая доставляет тебе удовольствие, и работой, к которой ты питаешь отвращение?

Для большинства технарей работа оказывается скучной по двум основным причинам. Во-первых, дело, которое нам нравится, заставляет напрягать мышцы нашей изобретательности. Разработка программного обеспечения является творческой деятельностью, и по этой причине многие из нас пришли в эту область. Работу, которая нам *не* нравится, мы практически никогда не можем назвать творческой. Обдумай это. Вспомни список своих рабочих дел на следующую неделю. Задания, выполнения которых ты предпочел бы избежать, скорее всего, не заставляют работать воображение. Это всего лишь рутина, которую ты с удовольствие перепоручил бы кому-то другому.

Вторая причина, по которой такие задания нас не увлекают, по общему мнению, тесно соединена с первой и заключается в том, что они не бросают вызов нашим способностям. Людям нравится искать и находить решения сложных проблем, в которых другие потерпели неудачу. Именно это чувство заставляет представителей нашего вида ради развлечения заниматься альпинизмом или прыгать на тарзанке с моста. Мы любим делать вещи, демонстрирующие наши возможности. Нудные задания, как правило, элементарны. Их выполнение по степени трудности и интересности можно сравнить с выносом мусора.

Так как же нам задействовать свой творческий потенциал и бросить вызов своим способностям в рутине будничных служебных обязанностей (которые у большинства из нас занимают более 80 % рабочего времени)?

А что, если попробовать делать рутинные дела *идеально*? К примеру, ты ненавидишь модульное тестирование. Ты обожаешь программирование, но чувствуешь раздражение, когда дело доходит до написания автоматизированных тестов. Что, если ты попробуешь сделать эти тесты идеальными? Как это повлияет на твое поведение? Что означает определение *идеальный* в случае модульного тестирования? Скорее всего, это связано с эффективностью теста. Идеальная эффективность означает, что ты протестировал функциональность своего кода на 100 %. Идеальные модульные тесты обычно свободны от ошибок, удобны в сопровождении и не зависят от множества сторонних факторов, которые может быть сложно воссоздать на другом компьютере. На новой машине они должны запускаться непосредственно после проверки версий. И разумеется, эти тесты должно проходить в 100 % случаев.

Задача начинает усложняться. 100-процентная эффективность теста звучит как практически нереальная вещь. Много сложностей представляет и разделение тестов, позволяющее избавить их от внешних зависимостей. Скорее всего, для обеспечения подобной возможности тебе придется править свой код. Но, справившись с этой задачей, ты получишь невероятный результат.

Не знаю, что подумаешь ты, но с моей точки зрения это звучит весьма увлекательно. Конечно, это искусственный пример, но аналогичным образом можно подойти к решению большинства встречающихся тебе задач. Попробуй сделать это завтра. Посмотри на свой рабочий день и задай себе вопрос: «Насколько хороша сегодняшняя работа?» Ты обнаружишь, что как только тебе начнет нравиться твоя работа, ты начнешь нравиться ей[1].

**Действуй!**

1. *Сделай это наглядным*. Преврати рутинные задачи в соревнование со своими коллегами. Кто может справиться с ними лучше? Не нравится писать модульные тесты? Распечатай набор требований к тестам для кода, проверка которого осуществляется каждый день, и повесь их перед собой на стену. Сделай доску оценок для всей группы. Соревнуйтесь за похвалу (или даже за призы). После завершения проекта можно сделать, например, так, чтобы остальные члены группы на неделю освободили победителя от рутинных обязанностей, взяв их на себя.

## Совет 25
## Сколько ты стоишь?

Ты когда-нибудь задумывался, сколько ты стоишь фирме, на которую работаешь? Разумеется, ты знаешь размер своей зарплаты. С этим все просто. А как насчет льгот, административных расходов, обучения и всех тех вещей, которые не входят в сумму заработной платы?

Очень легко привыкнуть все время *ждать большего*. По всей видимости, это общечеловеческая склонность. После увеличения зар-

[1] Спасибо за эту идею Энди Ханту (http://blog.toolshed.com/2003/07/how_good_a_job_.html).

платы человек некоторое время радуется, а потом начинает думать о следующем повышении. «Получай я на 10 % больше, можно было бы позволить себе этот новый...» Мы все так чувствуем. В какой-то момент величина зарплаты становится абстрактной. Речь идет уже не о дополнительных 5000 долларов в год. Нужно, чтобы увеличивалась базовая ставка. Не получив удовлетворяющего нашим требованиям повышения зарплаты, мы начинаем выражать недовольство своей работой и фирмой. «Почему они меня не ценят?»

Сколько ты стоишь на самом деле? Как я уже упомянул, это больше, чем твоя зарплата. Для простоты допустим, что твоя цена равна двум твоим окладам. То есть если ты зарабатываешь в год $60 000, фирма тратит на тебя около $120 000.

Пока все просто. Но пришла пора ответить на неприятный вопрос: какой доход ты принес в прошлом году? Каково твое *положительное* влияние на чистую прибыль компании? Мы уже знаем, что ты стоишь компании (в нашем воображаемом сценарии) примерно $120 000. Как ты компенсировал эту сумму? Сколько денег ты помог сэкономить? Какой вклад ты внес в годовой доход?

Превосходит ли это число две твоих зарплаты?

Это сложное упражнение, так как часто затруднительно соотнести все аспекты твоей деятельности с чистой прибылью компании. Более того, такая постановка вопроса может показаться тебе необоснованной. «Откуда я знаю? Я всего лишь кодер!» Но именно в этом суть дела. Если, работая в области бизнеса, ты не производишь никаких реальных ценностей, деньги на тебя тратятся впустую. Считая повышение зарплаты своим неотъемлемым правом, легко попасть в ловушку. Ведь при таком подходе и фирма имеет право каждый год увеличивать цену на свою продукцию. Но потребителя ничто *не заставит покупать продукт*, цена которого ему не нравится.

Теперь, когда ты начал думать, сравнивая свою стоимость с приносимым тобой доходом, возникает следующий вопрос. Какой доход, по твоему мнению, ты должен приносить, чтобы компания рассматривала тебя как выгодное вложение? Мы говорили о примерно двукратном превышении твоей заработной платы, но достаточно ли этого? Если ты приносишь доход в размере двух своих зарплат, ком-

пания все равно на грани рентабельности. Разве это хороший способ тратить деньги?

В качестве точки отсчета вспомним проценты по типичному сберегательному счету. Они не очень велики, не так ли? Тем не менее это лучше, чем ничего. Будь у тебя выбор, под какие проценты ты бы поместил свои годовые накопления: 0 или 3? Способность приносить доход в размере двукратной зарплаты представляет для фирмы такую же непривлекательную перспективу, как для тебя счет с 0 % годовых. Они на год замораживают расходы на тебя до 120 000 долларов, а ты не приносишь дохода, покрывающего даже уровень инфляции. В этом случае безубыточность оборачивается потерями.

Помню, подобные рассуждения поначалу доводили меня до приступов паранойи. Проходил месяц, и я думал: «Какой доход я принес в этом месяце?» Затем я начал рассматривать недели и дни: «Сколько я сегодня стоил?»

Можно подойти к делу конкретно. Насколько ты *способствуешь* росту прибыли предприятия? Уточни у своего начальника, как лучше всего выразить эту величину в цифрах. Сам факт, что ты *хочешь* это сделать, будет воспринят как хороший знак. Как творчески подойти к экономии денег фирмы? Как повысить эффективность разработчиков твоей группы? А как обстоит дело с конечными пользователями программ, которые ты пишешь? Ты удивишься, сколько возможностей обнаружится после того, как ты начнешь задавать эти вопросы. И сразу начинай действовать в соответствии с полученными ответами. Всегда помни про ориентир: *в два раза больше своей зарплаты*. Не позволяй себе уходить от ответственности, пока не превысишь это число.

> Спроси себя: «Сколько я сегодня стоил?»

**Действуй!**

1. Делая инвестиции, фирмы пытаются гарантировать максимально эффективное использование денег. Простого подсчета доходности по вложенным средствам (я вкладываю 100, а получаю 120) недостаточно для принятия взвешенного решения. Среди прочих факторов им нужно учесть инфляцию,

издержки и риски. Понятие временно́й стоимости денег кажется странным для тех, кто не обучался в школе бизнеса. Рискуя чрезмерно упростить ситуацию, я описал бы это так: доллар сегодня стоит больше доллара, который мы получим в следующем году, потому что сегодняшний доллар можно использовать для *получения большего числа долларов*.

Большинство компаний устанавливают *планку уровня доходности*, ниже которой инвестиции просто не делаются. Инвестиции должны дать определенный процент за определенное время, в противном случае фирма не вкладывает свои деньги. Это число называется *минимально допустимой рентабельностью инвестиций*.

Узнай минимально допустимую рентабельность инвестиций фирмы, в которой ты работаешь, и сопоставь ее со своей зарплатой. Являешься ли ты удачной инвестицией?

## Совет 26
## Камешек в ведре воды

Что произойдет, если ты встанешь и выйдешь из офиса, чтобы никогда туда не возвращаться? Я знаю многих программистов, которые утешаются, представляя подобную сцену. Ты просто встаешь, идешь в кабинет начальника и кладешь ему на стол заявление об уходе. *Они еще поймут, какого ценного работника потеряли!* Подобные мечты неплохо помогают пережить по-настоящему неудачные дни, но предаваться им постоянно — не самая лучшая позиция.

Начнем с того, что на самом деле это неправда. Люди увольняются с работы каждый день. Многих увольняют. Многие увольняются сами. Некоторые даже реализуют твою мечту и демонстративно уходят без уведомления. Но покидаемые ими фирмы крайне редко ощущают последствия их ухода. В большинстве случаев, даже когда речь идет о ключевых позициях, эффект оказывается на удивление слабым. Для фирмы твое присутствие на работе подобно камешку в ведре воды. Конечно, от его наличия общий уровень воды немного поднимется. Ты выполняешь свои обязанности. Ты вносишь свой вклад. Но если убрать из ведра камешек и посмотреть на поверхность воды, вряд ли кто-то сможет увидеть разницу.

Я не пытаюсь тебя унизить. Нам всем нужно чувствовать, что наш вклад что-то значит. И это действительно так. Но мы настолько привыкаем смотреть на все сквозь призму собственной *личности*, что забываем о наличии вокруг других *личностей*. Все окружающие тебя сотрудники компании — разумные и автономные существа, намертво сцепленные со своими личностями и смотрящие на свою работу исключительно с точки зрения своего «я». Подумай вот о чем: если завтра ты уволишься, это произведет (в среднем) примерно такой же эффект, как увольнение любого из твоих коллег.

Однажды мне довелось работать с одним из наиболее производительных директоров по информационным технологиям одной из самых влиятельных фирм в мире. Он и его команда (частью которой был я) получали все награды и задавали в фирме все стандарты в области информационных технологий. Очевидно, этот парень нашел рецепт магического эликсира, который распылял в бесплатные закуски, подававшиеся на организуемых им Y2K-вечеринках.

Один из наиболее действенных советов, которые я услышал от этого директора, — и он повторял его снова и снова — звучал так: ты никогда не должен чувствовать себя слишком комфортно. Он рассказывал, что каждый день после пробуждения намеренно и недвусмысленно напоминал себе, что может в любой момент лишиться своего высокого поста. *Это может случиться* даже сегодня.

Его сотрудники смотрели с недоверием. Нет. Сегодня этого не произойдет. Все идет так хорошо. Все работает на вас.

Но это была его точка зрения. Такое качество, как скромность, вырабатывают не для того, чтобы претендовать на большую духовность. Оно позволяет давать более четкую оценку своим действиям. Наш директор по информационным технологиям учил, что *чем более ты успешен*, тем выше вероятность сделать роковую ошибку. Когда все работает на тебя, ты начинаешь намного реже сомневаться в своих суждениях. Если используемый подход всегда срабатывал, вряд ли ты будешь искать новые лучшие подходы. Ты становишься самонадеянным, что ведет к появлению областей, в которых ты не разбираешься. Чем более незаменимым ты себя *считаешь*, тем менее незаменимым ты становишься (и тем меньше людей хотят с тобой работать).

Чувство незаменимости является плохим симптомом, особенно у разработчика программного обеспечения. Заменить нельзя только того, кто справляется со своей работой особым, недоступным другим способом. Хотя мы все хотели бы претендовать на гениальность, крайне немногие разработчики настолько уникальны, что их и в самом деле *нельзя* заменить.

Я слышал, как программисты полушутя говорили, что создать «гарантированное рабочее место» можно, просто написав трудный в сопровождении код. Мне доводилось даже встречать тех, кто предпринимал подобные попытки. И каждый раз эти люди становились *мишенями* для других. Разумеется, в фирме боялись их увольнять. Но в конечном счете страх — это худшее, что может возникнуть в подобном случае. Пытаясь стать незаменимым при помощи оборонительных маневров, человек вызывал враждебные чувства у своего работодателя (и коллег), а существовать в подобной атмосфере становится затруднительно.

Помни о том, как ослепляет собственный успех.

Логика подсказывает, что, пытаясь стать *незаменимым*, нужно строить дружеские рабочие отношения. Заменить можно любого. Те из нас, кто в состоянии это признать и начать работать с учетом этого факта, начинают выделяться из остальной массы и неосознанно повышают свои шансы. Кроме того, раз ты заменим, ничто не мешает тебе искать новую *более выгодную* работу.

**Действуй!**

1. Составь список написанного или поддерживаемого тобой кода и всех выполняемых заданий. Отметь все аспекты, в которых группа полностью от тебя зависит. Возможно, ты единственный, кто полностью понимает процесс развертывания своего приложения. Или существуют фрагменты кода, понять которые остальным членам группы особенно сложно.

   Помести все эти позиции в список текущих дел. Задокументируй, автоматизируй или перепроектируй каждый фрагмент кода или задание таким образом, чтобы их легко мог понять любой член группы. Когда все дела из списка будут завершены, ознакомь с документацией группу и ее руководителя. Убедись, что доступ к этим документам есть у всех членов группы.

   Периодически повторяй это упражнение.

# Совет 27
# Возлюби поддержку

Несколько лет назад мне пришлось с нуля создавать центр разработки программного обеспечения на 250 человек. Мы начали с пустого здания, имея задачу нанять весь персонал и полностью организовать рабочий процесс. При этом мы столкнулись с совершенно неожиданными трудностями. Все хотели создавать новые системы, но никто не горел желанием заниматься поддержкой старых. Мы рассчитывали сформировать новую креативную среду, поэтому с самого начала было важно понять *приоритеты* новых сотрудников.

Созидание нравится всем. Мы расцениваем это как возможность оставить свой след. Ощутить причастность. Выразить себя в своем творении. Кроме того, существует мнение, что проектные работы являются наиболее заметными в глазах руководства. Ведь славу и признание получают те, кто создает новое, не так ли? О том, насколько распространена такая позиция, я знал от программистов, с которыми сотрудничал ранее. Но столкнувшись с парой сотен разработчиков, предоставленных нам кадровой службой, я обнаружил совершенно неожиданные крайние проявления такого подхода.

Разработчики программного обеспечения являются, как правило, творческими, свободолюбивыми людьми, но при этом программистский «социум» демонстрирует удивительную кастовость. Программисты хотят быть проектировщиками, которые, в свою очередь, мечтают быть архитекторами, и т. д. Работа по техническому сопровождению не позволяет ни проявить себя, ни даже похвастаться конкретной высокой должностью (как, например, *архитектор*) перед родителями или бывшими сокурсниками.

Итак, мотивирующими факторами у нас являются способность проявить творческий потенциал и обязанности, выполнение которых потенциально ведет к повышению в должности. Забавно то, что работа над проектом тоже *далеко не всегда* предоставляет подобные возможности.

Работы по техническому сопровождению, как правило, тесно связаны со старыми, плохо функционирующими системами и надоедливыми

конечными пользователями. Так как считается, что в таких случаях программное обеспечение уже существует, отдел информационных технологий сосредоточивается на удешевлении сопровождения системы и ищет максимально дешевые способы поддержания ее работоспособности.

Обычно это означает выделение крайне незначительных ресурсов на надзор за этими системами и отсутствие финансирования на обновление оборудования.

В то же время хорошая работа — это работа, которая начинается с прекрасного, чистого, совершенно нового листа. В хорошо работающей компании каждый проект способствует заработку или экономии денег, поэтому обычно на их внедрение выделяются достаточные фонды (хотя случаи бывают разные). При этом отсутствует минное поле старого кода, по которому программисту приходится ходить на цыпочках, а значит, «корректно» разрабатывать новые подсистемы можно с куда меньшим числом помех, чем было бы в уже существующей системе. Короче говоря, внедрение новых проектов происходит в куда более благоприятных условиях.

Допустим, что я даю тебе тысячу долларов и прошу принести чашку кофе. Меня сильно расстроит, если ты вернешься с меньшей суммой и без кофе. Не вызовет у меня восторга и ситуация, когда ты принесешь мне прекрасный кофе, но через два часа. С другой стороны, если я не дам тебе денег и попрошу принести кофе, то достаточно высоко оценю выполнение моей просьбы, но пойму, если ты этого не сделаешь. Работа над проектом соответствует первому сценарию. Работа по технической поддержке — второму.

Когда стандартных ограничений — устаревшего кода или недостатка финансирования — нет, руководство и заказчики с полным правом ожидают от нас большего. Предполагается, что любой новый проект положительно скажется на деятельности фирмы. Если так не происходит, это означает, что ты не справился с задачей. Так как фирмы рассчитывают на увеличение доходов, зачастую они строго контролируют, что будет создаваться, каким образом и в какие сроки. И внезапно вместо полигона для творчества мы оказываемся в центре военной операции — каждое наше движение регламентируется сверху.

В режиме сопровождения надеются лишь на бесперебойную работу программного обеспечения при максимально низких затратах. Никто не ожидает от группы технической поддержки эффектных свершений. Пока все идет хорошо, клиенты воздерживаются от вмешательств в повседневное управление работой технических специалистов. Исправляй ошибки, делай небольшие доработки, о которых тебя попросили, и обеспечивай бесперебойную работу системы. Вот и все твои обязанности.

А как быть, если ошибка приводит к необходимости модернизации подсистемы приложения? Но то же самое относится к исправлению ошибок, не так ли? Проект может оказаться старым и допотопным, а вся система заполненной разбитыми окнами[1]. Это возможность испытать твои навыки переделки. Насколько элегантной можно сделать эту систему? Насколько быстрее можно будет исправить или улучшить этот раздел в следующий раз после выполненной сейчас переделки?

Пока ты обеспечиваешь работоспособность системы и достаточно быстро реагируешь на запросы пользователей, служба поддержки является местом свободы и творчества. Ты одновременно руководитель проекта, архитектор, дизайнер, кодер и тестировщик. Можно сколько угодно проявлять свои творческие способности, самостоятельно отвечая за все видимые успехи и неудачи.

> Отдел технической поддержки также может стать местом свободы и творчества.

Поддерживая систему, можно планировать заметные улучшения. Например, в созданной три года назад системе могут отсутствовать некоторые новые полезные элементы пользовательского интерфейса, доступные в современных браузерах. И если ты умеешь исправлять недостатки, сохраняя работоспособность системы, можно значительно повысить комфорт конечных пользователей. Можно неожиданно для заказчика корректно добавить дополнительные функциональные возможности — это все равно, что принести жене цветы без повода или убрать квартиру родителей,

[1] Подробно «разбитые окна» описываются в книге Дэйва Томаса и Энди Ханта *The Pragmatic Programmer: From Journeyman to Master* («Программист-прагматик. Путь от подмастерья к мастеру»).

пока их нет дома. Не сталкиваясь с бюрократией, неизбежной для полномасштабных проектов на стадии реализации, ты получаешь удивительное количество возможностей. И вполне можешь удивить заказчика.

В отличие от многих современных проектных групп, работающих по контракту, у службы поддержки есть скрытое преимущество — возможность непосредственно общаться с заказчиками. Это позволяет заводить многочисленные знакомства, формируя собственную группу поддержки. Кроме того, такая должность дает прекрасные возможности изучить, как функционирует отрасль, в которой ты занят, изнутри. Полностью отвечая за сопровождение бизнес-приложения и реагируя на проблемы и запросы конечных пользователей, можно без особых усилий понять, какую же нагрузку несет это приложение. Правила ведения бизнеса закодированы в логической схеме приложения, просчитать которую обычно не так-то просто. Я сталкивался с ситуациями, когда полностью специфику бизнес-процессов в компании понимал только программист из службы поддержки. Больше ни у кого не было непосредственного представления о кодировании бизнес-логики.

Самая большая ирония при противопоставлении работ над проектом и в службе поддержки состоит в том, что их *нельзя* разделить. После появления первой строки кода все следующие пишутся уже на ее базе. Разумеется, такой код чище и вызывает меньше проблем, чем код устаревшего приложения, но, по сути, действия ничем не отличаются друг от друга. Новые функциональные возможности добавляются к уже существующему коду и в этом же коде исправляются ошибки. А кто может справиться с этим лучше, чем человек, полюбивший техническую поддержку и поставивший себе цель научиться обеспечивать ее на отлично?

### Действуй!

1. *Измерь*, *исправь*, *измерь*. Для большинства важных приложений или кода, поддержкой которого ты занимаешься, составь список *измеримых* показателей, дающих представление о качестве работы. Это может быть время реакции приложения, количество необработанных исключений, выбрасываемых

во время функционирования, или время безотказной работы программы. Если ты занимаешься непосредственно сопровождением, не оценивай качество приложения напрямую. Важной частью работы с приложением является скорость твоей реакции на запросы пользователей.

Выбери наиболее важные из перечисленных атрибутов и приступай к их измерению. Измерив базовый уровень, поставь реальную цель повысить производительность приложения (или собственной работы). Внеся усовершенствования, снова выполни измерения, чтобы убедиться в достижении намеченной цели. Если цель достигнута, поделись этим с коллегами и заказчиками.

Выбери следующий показатель и повтори процедуру. После первого раза ты обнаружишь, насколько интересным и похожим на игру является этот процесс. Улучшение измеримых показателей вполне может стать твоей привычкой.

## Совет 28
## Восьмичасовое пламя

Одним из многочисленных поводов для полемики вокруг движения «экстремального программирования» является исходное утверждение о том, что члены группы должны работать не более сорока часов в неделю. Такие разговоры сильно расстраивают руководителей — поклонников рабского труда, жаждущих добиться от подконтрольных групп максимальной продуктивности. Порой это расстраивает и самих программистов. Количество непрерывно отработанных часов становится своего рода мужской гордостью разработчиков, напоминающей гордость за количество выпиваемых залпом кружек пива в студенческие годы.

Боб Мартин[1], один из корифеев сообщества экстремального программирования, перевернул эту фразу таким образом, что умудрился примирить с ней обе партии, не отступая от первоначальных постулатов Кента Бека. Мартин переименовал *сорокачасовую рабочую неделю* в «восьмичасовое пламя». Основная идея состоит в том, что человек

[1] http://www.objectmentor.com

должен работать настолько интенсивно, что просто не сможет работать больше восьми часов.

Прежде чем перейти к рассмотрению процесса горения, ответим на вопрос, почему акцент делается на уменьшении числа рабочих часов? Этот раздел посвящен работе над какими-то вещами. Имеет ли смысл разговор об *увеличении* продолжительности рабочего дня?

Когда дело доходит до работы, меньшее может превратиться в большее. Экстремальные программисты любят повторять, что уставший человек не может работать с той же эффективностью, что и отдохнувший. Когда мы доходим до предела, творческий подход угасает, а качество работы существенно снижается. Мы начинаем делать глупые ошибки, которые стоят нам времени и денег.

Большинство проектов являются долгосрочными. Невозможно пробежать марафон в том же темпе, что и спринт. Если ты начнешь засиживаться допоздна на работе, краткосрочная продуктивность повысится, но в долгосрочной перспективе ты рано или поздно надорвешься настолько серьезно, что затраченное на восстановление время не оправдает результатов, достигнутых за восьмидесятичасовые рабочие недели.

Ко времени можно относиться как к деньгам. Юношей я работал неполный рабочий день за минимальную плату и был бы счастлив, будь у меня столько же денег, сколько сейчас я *трачу на всякую ерунду*. Сейчас у меня так много денег, что я уже не обращаю внимания на разнообразные мелкие расходы. Тем не менее в прошлом я каким-то образом умудрялся жить. У меня были квартира и машина, я не голодал.

> Проект — это не спринтерская дистанция, а марафон.

Все это есть у меня и сейчас. Но нельзя сказать, чтобы я вел роскошную жизнь. Просто в те времена, когда мне не хватало денег, я пытался оптимизировать свои расходы. Получая в итоге, по сути, аналогичный результат.

Скудные ресурсы мы считаем более ценными и стараемся использовать их эффективно. Этот подход применим не только к деньгам, но и ко времени. Представь четвертый день своей семидесятичасовой

рабочей недели. Без сомнения, ты будешь прилагать героические усилия. Но с четвертого дня ты, непременно, начнешь расслабляться. *Стрелки показывают всего 10:30 утра, а ты знаешь, что останешься на рабочем месте еще на несколько часов после того, как все остальные уйдут домой. Что мешает некоторое время уделить чтению информации о последних технологических новинках?*

Если рабочего времени слишком много, его реальная ценность заметно снижается. При семидесяти часах ценность одного часа куда меньше, чем в случае, когда их у тебя всего сорок.

Если стоимость доллара падает вследствие инфляции, для покупки такого же, как и раньше, набора вещей потребуется бóльшая сумма. При снижении ценности часа на выполнение одной и той же работы будет уходить больше времени. Восьмичасовое горение Боба Мартина налагает на тебя ограничение и дает тебе стратегию действий в новых условиях. По дороге на работу ты думаешь: *у меня всего восемь часов! Быстрее, быстрее, быстрее*! Наличие жестких временны́х рамок естественным образом позволяет распределить время более эффективно. Можно начать с набора заданий на сегодня, определить их приоритеты и приступить к выполнению по одной за раз.

Восьмичасовое горение создает среду, напоминающую крайне продуктивные выходные в студенческие годы, когда тебе нужно быстро подготовиться к зачету по лекциям, которые ты прогуливал, или написать наконец курсовую работу, павшую жертвой постоянных проволочек. Разница — в ограниченной по времени *зубрежке*. Время зубрежки крайне продуктивно, потому что его не хватает, а значит, оно очень ценно. Восьмичасовое горение представляет собой метод изначально плотной работы с регулярным графиком, при котором тебе не потребуется бодрствовать ночами, вливая в себя кофе и энергетики.

Как работники умственного труда мы в состоянии трудиться вне офиса, не имея перед собой компьютера. Это происходит во время поездки на обед с супругой или во время просмотра фильма. Работа постоянно не дает тебе покоя.

Моя работа обычно не дает мне покоя, когда я уделяю ей недостаточно внимания. Когда я спускаю какие-то дела на тормозах или позволяю им накапливаться. В этом случае работа отправляется со

мной домой и не дает мне спокойно расслабиться. При каждодневной интенсивной трудовой деятельности ты обнаруживаешь, что перестал тащить домой рабочие проблемы. И дело не в том, что ты намеренно не даешь себе работать сверхурочно. Твой мозг просто *не позволяет* этого делать.

Распоряжайся своим рабочим временем аккуратно. Работай меньше, и ты начнешь больше успевать. Работа всегда приносит больше удовольствия, когда ты можешь от нее отдохнуть.

**Действуй!**

1. Постарайся как следует выспаться сегодня. На следующее утро позавтракай и приступи к работе в определенное время (лучше чуть раньше, чем обычно). Интенсивно работай в течение четырех часов. Отведи час на обеденный перерыв. Затем поработай еще четыре часа с такой интенсивностью, чтобы в конце концов почувствовать себя совершенно изможденным. Возвращайся домой, расслабься и развлекайся.

## Совет 29
## Научись проигрывать

Как программисты мы знаем, что чем скорее в процессе разработки будут выявлены ошибки, тем более надежной выйдет программа. Модульное тестирование помогает обнаружить странные ошибки на ранних стадиях. Если нам удается быстро выявить в собственном коде причудливые неполадки, мы счастливы. Хотя сами ошибки указывают на некоторые недостатки нашей работы, их своевременное выявление является хорошим признаком, позволяющим надеяться на корректно функционирующую программу.

Нас учили, что в начале работы можно совершать ошибки. Ты хочешь узнать, когда они возникают, чтобы внести исправления или пометить проблемный фрагмент кода. Когда ты пишешь код, то не прилагаешь особых усилий для устранения небольших программных неполадок, неизбежно сопровождающих процедуру разработки.

Ошибка — это попытка кода донести до тебя информацию. Часть процесса стабилизации. Поэтому мы специально добавляем инструкции, приводящие к падению нашей программы, когда что-то идет не так, или модульные тесты, в случае нашего промаха показывающие красный флаг.

Эти небольшие неполадки кроме всего прочего дают нам понять, проблем какого типа можно ожидать. Если ты никогда не ходил по минному полю, откуда тебе знать, на какие куски грязи лучше *не* наступать? Если программа не подает никаких сигналов, откуда тебе знать, в каком месте притаилась опасность? При написании кода вслепую приходится постоянно быть начеку.

Программировать безопасно крайне важно. Когда что-то идет не так, качество программного обеспечения подвергается испытаниям. Неизбежно возникают *ситуации*, реакцию приложения на которые вы не запрограммировали. Аварийные завершения и синие экраны в окончательной версии кода означают, что программист не поработал над ошибками, которые не смог предвидеть.

Те же принципы применимы к работе. Ремесленник подвергается испытаниям, когда возникают ошибки. Умение с ними работать является ценным и сложно приобретаемым навыком. Как человек, увлекающийся джазовой импровизацией, я понял, что любая фальшивая нота находится по меньшей мере в одном шаге от верной. Импровизация никогда не будет хорошей, если музыкант, взяв фальшивую ноту, не знает, что ему делать дальше. Когда рядом твоя группа, а впереди публика, эта гадкая нота способна вогнать тебя в полный ступор. Фальшивые ноты бывают даже у мастеров. Но они лихо выходят из положения, и слушатели даже на подозревают, что все произошло спонтанно.

> Любая фальшивая нота находится рядом с верной.

Мы все обречены допускать ошибки на работе. В конце концов, мы всего лишь люди. Мы делаем ошибки в коде, заставляющие клиентов выполнять трассировку стека. Мы загоняем себя в угол критическими дизайнерскими ошибками. Или еще хуже — мы говорим неправду членам нашей группы, начальству и заказчикам. Мы даем невыполнимые обязательства или претендуем на отсутствующие у нас умения.

Мы можем случайно дать коллегам неудачный совет по поводу решения технической проблемы, вынудив их впустую потратить время.

Так как все мы делаем ошибки, мы даем право на ошибку другим. И в разумных пределах не судим друг друга за оплошности. Суждение о человеке выносится по тому, как он разбирается с этими неизбежными ошибками.

В случае технической, информационной или проектной ошибки применимы следующие правила.

- Озвучивай проблему сразу же, как только ты о ней узнал. Не пытайся ее скрыть. В разработке и тестировании программного обеспечения обнаруженные на ранних стадиях ошибки создают куда меньше проблем, чем ошибки заключительного этапа. Чем раньше ты озвучишь свой промах, тем менее негативные последствия он будет иметь.
- Бери вину на себя. Не пытайся найти козла отпущения, даже если у тебя есть хорошая кандидатура на эту роль. Даже если ты не полностью виноват в ситуации, возьми ответственность на себя и двигайся дальше. Важно миновать эту точку как можно быстрее. Проблема должна быть решена. Затянувшийся поиск виноватых только отодвинет этот момент.
- Предлагай решение. Если ты его не знаешь, предлагай план поиска решения. Оперируй реальными и конкретными сроками. Если по твоей вине работа группы зашла в тупик, ограничь временны́е рамки решения проблемы и дай оценку требуемых на устранение проблемы усилий. На этой стадии важны конкретные, достижимые цели, пусть даже небольшие. Они не только инициируют процесс перехода от плохого к хорошему, но и помогут восстановить доверие.
- Проси помощи. Даже если вина за возникшую проблему лежит целиком на твоих плечах, не позволяй собственной гордости усугубить ситуацию, отказавшись от помощи. Коллеги, руководство и заказчики увидят тебя в более позитивном свете, если ты сможешь сохранить здравый подход, отставив в сторону эгоистические соображения, пока группа помогает тебе с поиском выхода. Зачастую наше чувство ответственности заставляет взвалить себе

на плечи слишком тяжелую ношу и непродуктивно тратить силы, пока кто-нибудь не вмешается в ситуацию.

Вспомни последний случай, когда тебя плохо обслужили в ресторане. Например, ты долго ждал заказ и в конце концов официант принес *не то блюдо*. Вспомни, как официант отреагировал на твою жалобу.

В данном случае оправдания или попытки обвинить повара являются неверной реакцией. Не стоит ему и уходить, чтобы повторить заказ, после чего стараться не попадаться тебе на глаза, пока ты умираешь с голоду и недоумеваешь, когда же, черт возьми, тебе принесут еду. И уж совершенно точно официанту не нужно ставить на стол не то блюдо, заранее зная, что заказ выполнен неверно, но надеясь, что ты либо не заметишь, либо не станешь жаловаться.

> Стрессовая ситуация дает прекрасную возможность для формирования лояльности.

Реакция фирмы в случае допущенной ошибки порой оказывается ключевым фактором в деле формирования (или разрушения) лояльности. Правильная реакция на ошибку может сделать нас более лояльными клиентами, чем мы могли бы быть при отсутствии проблем с сервисом. Вспомни об этом правиле при работе со *своими* заказчиками, когда тебе случится сделать ошибку.

## Совет 30
## Умей говорить «нет»

Самый быстрый путь к невыполнению обязательств — непосильные обязательства. Несмотря на очевидность этого утверждения, мы берем их на себя каждый день. Попав в затруднительное положение, мы не хотим разочаровывать начальство и соглашаемся сделать нереальную работу за нереально короткий срок.

Говорить «да» — затягивающая и вредная привычка. Причем вредная привычка, выдающая себя за полезную. Но есть большая разница

между установкой на выполнение задания и введением в заблуждение по поводу своих возможностей. Последнее создает проблемы не только тебе, но и тем, кому ты раздаешь обещания. Если я (как твой начальник) интересуюсь, можешь ли ты к концу месяца модернизировать подсистему мониторинга грузов в программе учета заказов нашей фирмы, конец месяца, скорее всего, назван не случайно. Возможно, кто-то попросил *меня* сделать это к указанному сроку. А, возможно, на систему учета заказов завязаны важные изменения в деловой сфере, которые мы пытаемся внедрить. Воодушевленный твоей уверенностью в реальности выполнения работы к концу месяца, я, в свою очередь, уверяю заказчиков, что все будет готово.

> Говоря «да», чтобы избежать разочарования, мы попросту врем.

В подобной ситуации «да» равноценно лжи. Я не считаю такую ложь преднамеренной. Мы лжем себе ровно в той же степени, что и людям, которым мы раздаем обещания. В конце концов, отказывая, мы чувствуем себя неловко. Мы запрограммированы на постоянный успех. И признаваясь, что мы чего-то *не можем*, мы чувствуем себя неудачниками.

Но мы не в состоянии усвоить, что ответ «да» далеко не всегда правильный. А ответ «нет» редко бывает неверным. Я говорю о необходимости усвоить этот факт, потому что внутри себя мы *понимаем* его истинность. Если уж на то пошло, вряд ли кто-то хочет сам пострадать от фиктивных обязательств.

Неумение отказывать является частью индийской культуры. С этим сталкиваются фирмы, не имеющие опыта в переносе рабочих процессов за границу с применением местной рабочей силы. Со временем у них появляется нюх на неопределенность и вырабатывается способность задавать правильные вопросы. Многократное «нужен еще один день, и все будет готово» естественным образом заставляет копать глубже. И с подобным приходится сталкиваться не только в области информационных технологий. В Бангалоре мне пять раз пришлось ждать установки кабельного модема, и я ее так и не дождался. Оказалось, что первые три раза компания, которая принимала мой заказ, не имела даже нужных комплектующих. Но они не хотели меня разо-

чаровывать. Я сообщал о своих надеждах получить кабельный модем на следующей неделе, и мне обещали сделать к указанному сроку, прекрасно зная, что это физически невозможно.

В основе такого поведения лежало вполне положительное намерение, но его последствия были негативными. В конечном счете я высказал монтажникам все, что о них думаю, и даже заставил прийти ко мне в воскресенье для выполнения работ. Я уже не верил обещаниям, что все будет установлено «завтра, после праздников». Постоянно нарушаемые обязательства не оставили ни единого шанса, что я в них поверю. Более того, у меня появилось по отношению к ним враждебное чувство.

А что случится, если тебя попросят выполнить важную работу, а ты ответишь отказом? Как руководитель местной и офшорной групп могу сказать, что приму такой ответ с чувством облегчения. Умение подчиненного сказать «нет» дает мне уверенность, что когда он говорит «да», он на самом деле это подразумевает. Обязательства такого человека заслуживают большего доверия и имеют больший вес. Если он действительно выполнит взятые на себя обязательства, в следующий раз я не буду задавать дополнительных вопросов, если он ответит мне *отказом*.

Человек, всегда говорящий «да», либо обладает невероятными способностями, либо лжет. Чаще всего действительности соответствует второй вариант.

Бывают ситуации, когда уместно ответить «я не знаю». Это может быть ответом на вопрос о сроках выполнения задания, если перед тем как взять на себя обязательства, тебе требуется время для знакомства с фронтом работ. Кроме того, тебя могут спросить о принципе, лежащем в основе технологии, или о способе реализации какого-то фрагмента кода. Как и в случае с обязательствами, незнание ответа заставляет чувствовать себя непрофессионалом. А претензии на осведомленность в рассматриваемой области заставляют коллег и начальство сильнее верить в тебя. Но при встрече с настоящим специалистом обрати внимание, что он не боится признавать ограниченность собственных познаний. Фраза «я не знаю» не означает неуверенности в себе.

Подобное же мужество требуется в ситуации со спускаемыми сверху решениями. Сколько раз нам приходилось сталкиваться с продиктованными начальником технологическими решениями, заставляющими членов группы тихо сидеть за столом, разглядывая собственные ногти, и ждать возможности покинуть зал совещаний, чтобы тихо поплакаться друг другу? Начальники часто становятся жертвами феномена «голого короля». Все понимают, что решение плохое, но никто не решается возразить. Как руководитель, я все время принимаю решения или даю настоятельные рекомендации. Но я нанимал работников не для слепого исполнения моей воли. И моим надежным помощником может стать только тот, кто не боится выступить и предложить лучший вариант.

Но не переусердствуй с количеством отказов. Исполнительность по-прежнему в цене, и здорово, когда человек готов взять на себя повышенные обязательства. Если ты не уверен, что справишься, но хотел бы попробовать, так и скажи. «Это будет сложная задача, но я хотел бы попробовать» — великолепный ответ. Хотя порой лучше ограничиться простым «да».

Имей смелость быть честным.

### Действуй!

1. Наш рецензент Карл Брофи предложил вести список взятых на себя обязательств.
   - Что тебя попросили сделать к определенному сроку?
   - Что ты пообещал?
   - Если что-то не получилось выполнить, запиши, что ты думал и на что тебе пришлось согласиться.
   - Запиши дату выполнения.

   Читай этот список каждый день. Если станет понятно, что такие-то обязательства выполнить в срок не удается, сразу же проинформируй об этом. Каждый месяц оценивай свой коэффициент успеха. Как часто ты выходишь победителем?

# Совет 31
# Не паникуй

Карьеру программиста я начал из-за видеоигр. Интерактивные приключения с эффектом присутствия очаровали меня еще во времена Commodore 64 с его видеолентами. Я стеснялся своего пристрастия, но теперь понимаю, что нечего было стыдиться. Компьютерные игры превращали невыразительные картинки, которые я видел на экране (полагаю, это был интерфейс операционной системы), в среду, где мне было комфортно и интересно.

Моя любимая игра на все времена — Doom от *id Software*. Я был в восторге от одиночного прохождения, дуэлей с реальными игроками и «боев насмерть». Компьютеры соединялись по модему или через последовательный порт, и игроки сражались друг с другом на небольших, быстро меняющихся площадках. Я достиг больших успехов в «боях насмерть». Я часто шучу, что, может быть, это и сейчас единственное, в чем я достиг максимального успеха. Игра в этом режиме удивительно сложна. Как с технической, так и с психологической точки зрения она напоминает безумную смесь шахмат с фехтованием на повышенной скорости.

Для повышения своих навыков полезно наблюдать за работой мастеров. Во времена моего активного увлечения Doom был один такой мастер, выступавший под ироническим псевдонимом Noskill (Неумеха). По факту он был главным чемпионом в Doom. Игроки из Северной Америки платили за междугородные телефонные соединения, чтобы попытать счастья в бою против него. Эти матчи записывали при помощи встроенной в Doom программы. И я посмотрел их все.

Очень скоро я понял его секрет. Разумеется, он был очень сильным игроком, но ключом к его успеху была способность никогда не паниковать. Специфика Doom такова, что матч может закончиться буквально за несколько секунд. Все происходит очень быстро. Я помню мой первый «бой насмерть». Спаун, смерть, спаун, смерть, спаун, смерть. Когда у меня наконец получилось выжить больше, чем две секунды, оказалось, что я бесцельно бегу, не в силах понять, где нахожусь.

Noskill никогда так не поступал. Просматривая записи его матчей, понимаешь, что какой бы сложной ни была окружающая обстановка, он всегда оставался спокойным и продумывал свой следующий шаг. Казалось, он всегда знал, как текущая ситуация вписывается в общую канву матча.

Если посмотреть другие игры, и в частности на спортивные состязания, ты увидишь, что лучшие игроки тоже обладают этим качеством. Более того, оно свойственно даже восхищающим нас персонажам книг, фильмов и телепередач. Герои никогда не паникуют. Это люди, которые даже при падении на их город атомной бомбы или при крушении самолета смогут собрать всех в группу, помочь выжившим, перехитрить врага или по крайней мере не будут тратить время на бесполезные рыдания.

Герои никогда не паникуют.

Это распространяется и на реальную жизнь. Вопреки самым лучшим планам моя профессиональная жизнь была чередой чрезвычайных ситуаций и бедствий. Проекты запускались слишком поздно. Приложения падали, что стоило моим работодателям денег и доверия. Я сказал не те слова не тому вице-президенту и нажил политического врага. Все эти неприятности в основном шли волнами, а не по отдельности.

Хуже всего было, когда я впадал в панику. Я запирался и в лучшем случае мог размышлять тактически. Я реагировал на каждый небольшой сигнал, не пытаясь рассмотреть картину в целом.

Оглядываясь на *все* эти неприятности, я должен признать: ни одна из них не оказала значительного влияния на мою карьеру. Хотя я воспринимал все эти ситуации через призму моей паники, депрессии и расстройства как катастрофические, ни одна из них не привела к *реальной* катастрофе.

Что мне дала паника? В чем было преимущество негативной реакции во всех этих случаях? Да ни в чем. На самом деле паника лишала меня возможности проявлять свои лучшие качества в моменты, когда это было действительно *нужно*.

Но должен признать, что совет не терять головы в стрессовых ситуациях проще дать, чем ему последовать. Все равно что сказать чело-

веку «просто будь счастливым». Разумеется, это хороший совет, но как им воспользоваться? Как не впасть в панику, когда кажется, что все вокруг рушится? Чтобы ответить на этот вопрос, нужно немного подумать, *почему* мы паникуем.

Мы паникуем, потому что теряем перспективу. Когда что-то идет не так, сложно сосредоточить на проблеме все свое внимание. До определенной степени такая концентрация помогает в поиске решения. К сожалению, она заставляет проблему, вне зависимости от ее реального размера, казаться более важной, чем она есть на самом деле. И по мере того, как проблема раздувается, растет уровень стресса и наши мозги отключаются.

Кто из твоих знакомых является самым худшим пользователем компьютера? У меня это один из моих родственников или родственников жены (это вполне конкретный человек, но у меня хватает ума обойтись тут без имен). Представь, что этот человек сидит за компьютером и пытается закончить работу, но вдруг каждое его действие начинает сопровождаться появлением сообщения об ошибке. Думаю, у каждого бывала подобная ситуация. Неопытные пользователи сразу начинают нервничать и терять самообладание. Они беспорядочно щелкают мышкой и двигают что-то по экрану, игнорируя потенциально полезную информацию из сообщения об ошибке, которая появляется снова и снова. В конечном счете они начинают паниковать и звонить с просьбой о помощи, но обычно к этому моменту они успевают сломать еще что-нибудь.

Не подумай обо мне нехорошо, я просто предлагаю представить, что это случилось с кем-то, кого ты хорошо знаешь, и посмеяться над этим. Ведь такое поведение выглядит нелепо. Это просто анекдотично.

Но по-настоящему смешно то, что описанная ситуация является реальным жизненным сценарием. Человек оказывается вне своей зоны комфорта, сталкивается с проблемой и впадает в панику. Это ничем не отличается от моей реакции на отложенный запуск проекта, случайное падение системы или недовольного заказчика. Это всего лишь другие внешние условия.

И вот как я научился не паниковать. Если происходит что-то плохое и я начинаю ощущать это поглощающее чувство напряжения, веду-

щее к панике, то я сравниваю себя с расстроенным компьютерным «чайником» и начинаю смеяться над собой. Я анализирую ситуацию с точки зрения третьего лица, как будто помогаю своему родственнику справиться с переставшим работать текстовым редактором. И неразрешимые проблемы внезапно становятся проще. Сложные ситуации вдруг оказываются не такими уж запутанными. А решение зачастую оказывается простым и лежащим у меня под носом, как окно с сообщением об ошибке, информирующее, какие действия следует предпринять. Если у тебя хватит силы воли прочитать сообщение, проблема будет решена.

### Действуй!

1. Заведи «тревожный» журнал. Способ предотвратить паническое состояние — внимание к собственным ощущениям и эмоциям в стрессовых ситуациях. Я хорошо анализирую свои реакции на происходящее после того, как все заканчивается. Мне не хватает ресурсов на поддержание фонового программного потока с одновременным копанием в природе собственных мыслей, но оказалось, что, рассматривая причины стресса постфактум, я все лучше и лучше начинаю анализировать происходящее в реальном времени.

   Сказать, что ты собираешься научиться анализировать собственные реакции, и на самом деле научиться этому — это разные вещи. Ведение журнала сделает этот процесс более структурированным. Каждый день в определенное время (используй календарь с напоминалками!) открывай текстовый файл и записывай ситуации, заставившие тебя хотя бы немного запаниковать. Раз в неделю просматривай список и критически оценивай влияние паники на каждую ситуацию. Стоило ли вообще паниковать? Какой могла бы быть наиболее продуктивная реакция? Что сделал бы герой драматического фильма, посвященного твоей жизни, чтобы не впасть в панику?

   Через некоторое время ты обнаружишь способность проводить подобный анализ непосредственно перед приступом паники. Рассматривая причины паники с рациональной точки зрения и в реальном времени, ты почувствуешь, как паника отходит на второй план и постепенно уходит.

# Совет 32
# Скажи это, сделай это, покажи это

Чтобы не доводить дела до логического конца, достаточно не брать на себя никаких обязательств. Если срок исполнения не указан, ничто не заставляет и не мотивирует тебя на трудовые подвиги. Это особенно верно, когда приходится делать *нечто* не особо интересное.

Даже плохим руководителям инстинкт говорит о важности планирования. У некоторых разработчиков магическое слово *планирование* вызывает тревогу. Бесконечные совещания со слабоумными начальниками, создающими в Microsoft Project многочисленные никому не понятные и никому не нужные планы, являются вполне реальным поводом для беспокойства. В итоге зачастую технари компенсируют протест против излишнего планирования, в пику ему постоянно действуя по наитию.

Однако планирование — не горькое лекарство, принимать которое можно, только задержав дыхание. Оно может дать вам освобождение. При наличии слишком большого количества дел именно планирование позволяет перейти от неопределенности, приводящей в замешательство в начале рабочего дня, к четкой уверенности в очередности решения задач.

План вовсе не обязан представлять собой большой чертеж. Достаточно списка в текстовом файле или в электронном сообщении. Не имеет смысла планировать на много дней вперед. Достаточно, чтобы план отвечал на вопрос: «Чем я буду сегодня заниматься?» Я знаю множество людей, дни которых столь сумбурны, что им практически никогда не удается пройти этот тест. В качестве первого шага рекомендую тебе найти время сегодня днем и выписать все дела на завтра в порядке убывания их приоритета. Пытайся реально оценивать свои силы, хотя не думаю, что у тебя есть склонность намеренно брать на себя повышенные обязательства.

План на один день может быть сколь угодно подробным или свободным. У меня был сосед по комнате, которого звали Крис. Каждое утро он просыпался и, даже рискуя опоздать на занятия, тщательно планировал свой день, уделяя особое внимание отработке упражне-

ний на фортепиано (его специализацией было джазовое фортепиано). Учитывая количество лекций, которые он выбрал для посещения, у него был достаточно жесткий график. Он планировал даже то, что будет делать в 15-минутных перерывах *между* занятиями, вставляя в эти промежутки быстровыполнимые практические процедуры. Многие лекции проходили в одном здании, и зачастую в перерывах у студентов была масса свободного времени, которое они тратили на общение. Пока остальные сидели в ожидании начала следующей лекции, Крис зубрил звуковые ряды или занимался сольфеджио. Он даже делил свое расписание на многочисленные промежутки, занимавшие от трех до пяти минут, успевая за 10-минутный перерыв выполнить несколько практических упражнений. В итоге он стал одним из наиболее уважаемых музыкантов в нашем городе. Конечно, доля успеха тут приходится на его природный талант, но лично я убежден, что именно планирование и следование собственным планам обеспечило ему билет в ряды музыкальной элиты.

Итак, твой план готов. Он может быть не таким детальным, как у Криса, но его должно быть достаточно для ответа на вопрос, чем ты собираешься сегодня заниматься. На следующий день, придя на работу, достань список и начни с первого пункта. Действуй в соответствии со списком, пока не настанет время обеденного перерыва, а затем начни с прерванного места и постарайся завершить все намеченные дела.

Завершив дело из списка, напиши рядом ГОТОВО, используя прописные буквы. Затем произнеси *готово* и почувствуй себя счастливым. В конце дня посмотри на список завершенных дел и ощути, что ты достиг некоего результата. Ты не только знал, чем ты будешь сегодня заниматься. Теперь ты знаешь, что все *готово*.

Если у тебя не получится целиком выполнить запланированное, не переживай. В конце концов, никогда нельзя заранее предсказать, сколько удастся сделать за день. Просто перенеси незавершенные дела из сегодняшнего списка (если они еще актуальны) в список дел на завтра и повтори весь процесс заново. Это очень стимулирует. Это позволяет войти в ритм. Это дает возможность разделить свои дни и недели на серию маленьких побед, каждая из которых толкает тебя к следующей. Ты обнаружишь, что эти списки не только делают твои

достижения заметными, но и позволяют достигать куда *большего*, чем во времена, когда ты не визуализировал свои успехи.

Если у тебя появилась привычка к определенному ритму планирования и выполнения, можно начинать думать в категориях недель и даже месяцев. Разумеется, чем больший промежуток времени нужно распланировать, тем менее подробным должен быть план. Расписание на неделю и на день можно сравнить с тактическим планированием битвы, в то время как расписания на тридцать, шестьдесят и девяносто дней лучше сконцентрировать на стратегических целях.

Для бойцов фронта программного обеспечения планировать дела на девяносто дней вперед не очень естественно. Мы — оперативная группа. Заставляя себя вообразить состояние разрабатываемой системы, происходящих в группе процессов и собственной карьеры через девяносто дней, ты столкнешься с вещами, о которых даже не подозревал. Вид на поле боя с высоты сильно отличается от вида снизу. Сначала тебе будет сложно, но продолжай усердно работать. Все полезные навыки нарабатываются только практикой, а преимущества скоро увидишь и ты сам, и те, кто с тобой работает (даже если они не в курсе, чем ты занимаешься).

Начни сообщать о своих планах руководству. Лучше всего это делать после выполнения хотя бы одного цикла своего плана. И — что крайне важно — начни это делать до того, как тебя об этом попросят. Ни один пребывающий в здравом уме начальник не выскажет недовольства, получив по электронной почте от своего подчиненного *лаконичный* отчет о сделанном за прошедшую неделю с планом на следующую. Еженедельное получение таких непрошеных писем — мечта каждого руководителя.

> Отчеты о проделанной работе помогут продвигаться на рынке труда.

Начни с еженедельных отчетов. Привыкнув, переходи к планам на тридцать, шестьдесят и девяносто дней. В более долгосрочных перспективах концентрируйся на обобщенных, эффективных улучшениях, которые ты наметил для своих проектов или поддерживаемой системы. Всегда формулируй эти планы как предложения к руководству

и проси прислать комментарии и пожелания. Со временем эти предложения будут требовать все меньших согласований, потому что ты на практике узнаешь, какие вещи обычно принимаются без вопросов, а какие требуют дальнейшей проработки.

Самой важной вещью, которую следует все время иметь в виду, является необходимость проработки всех пунктов плана. Задача может быть завершена, отложена, удалена или замещена. Ничто не должно оставаться неучтенным. Если задача появилась в твоем списке, а потом ни разу нигде не упоминалась, люди перестанут верить твоим планам и эффективности планирования. Сообщать следует даже о *негативных* результатах. Мы все допускаем ошибки. Но ты можешь выделиться из общей массы, публично признавая свои ошибки и несостоятельность и не стесняясь просить помощи. Постоянный мониторинг состояния запланированных дел создаст тебе заслуженную репутацию человека, у которого ни одна важная работа не теряется в общей массе дел.

Отладь этот процесс, и начальство обязательно обратит внимание на твой стратегический подход. Написание и выполнение планов демонстрирует, что ты не робот для написания кода, а *лидер*. Именно такой независимой отдачи компании ждут в рамках мероприятий по снижению накладных расходов.

Окончательным преимуществом общения посредством планов является упрочение доверия к обязательствам, которые ты на себя берешь. Если ты говоришь, что собираешься что-то сделать, а затем делаешь это и показываешь готовый результат, ты создаешь себе репутацию *человека дела*. Вместе с доверием к тебе растет и твое влияние. Представь, что ты хочешь внедрить в своем отделе гибкую методологию разработки[1] или новую технологию. Общепризнанная способность брать на себя и выполнять обязательства обеспечит тебя большей свободой действий в этих экспериментах.

В Бангалоре, в нашем центре программного обеспечения, была группа, которая более года работала в ночь. Из семи входящих в нее человек двум всегда доставалась ночная смена. Еженедельно они ме-

[1] http://www.agilemanifesto.org

нялись, в итоге каждую третью или четвертую неделю каждому члену группы приходилось работать с 7 вечера до 3 утра. Группа постепенно выдыхалась, страдая от постоянной смены суточных ритмов. Но эта группа играла важную вспомогательную роль, и заказчики из США считали, что не справятся без специалистов из Бангалора, помогающих в режиме реального времени.

В итоге был разработан план действий. Группа рассмотрела различные процедуры поддержки и связанные с ними измерения и придумала, как *одновременно* и избавиться от ночных смен, и улучшить качество обслуживания клиентов. Как действующий руководитель проектов я помог оптимизировать их план и выступил (с моральной поддержкой) в рамках официального предложения руководству в США.

Они знали, что для начальника, который лично отвечал перед американскими заказчиками, это будет щекотливая тема. Когда встреча началась, многих членов группы в буквальном смысле слова охватила дрожь. Но руководитель группы был столь впечатлен, что немедленно поставил свою подпись под предложением, и группа перешла к реализации плана. Через несколько недель ночные смены канули в прошлое и все вернулись к нормальному расписанию.

Основательность их плана, позволявшего не только поменять рабочие часы, но и стратегически повысить производительность работы, внушила большое доверие не только руководству, но в конечном счете и заказчикам. Руководитель группы пользовался этим планом, обсуждая изменения с заказчиками. И группа довела дело до конца. За несколько месяцев она вышла на новый уровень эффективности. И с этого момента доверие к ним настолько возросло, что постепенно они получили возможность работать независимо.

Группа воспользовалась собственным планом для решения конкретной проблемы. Они пошли к начальству не с жалобами, а с предложением по разрешению ситуации.

Твои руководители хотят, чтобы ты мог работать независимо и ощущал вовлеченность в процесс. Составление планов, их воплощение и отчет о достигнутых результатах помогут тебе получить то и другое.

## НЕУДАЧИ И КОПИРОВАНИЕ

*Патрик Коллисон, студент Массачусетского технологического института*

Ларри Уолл писал, что типичными чертами великих программистов являются лень, нетерпение и гордыня. Я не знаю, врожденные это особенности или они приобретаются прилежной работой над собой. В любом случае, непонятно, как воспользоваться этой информацией, чтобы стать более квалифицированным программистом. Поэтому будем смотреть не на характеристики, а на показатели, которые помогут нам самосовершенствоваться.

Если бы мне нужно было выбрать только два показателя, я бы остановился на неудачах и копировании.

Знаю, что ошибаюсь чаще других программистов. Большинство моих проектов заканчиваются провалом. В папке ~/Projects можно найти целый ворох забытых попыток сделать нечто интересное. Шансы на успех у каждой из них были примерно такими же, как шансы лобстера уплыть из кастрюли в океан. Кое-чем они привлекают внимание. Успешные проекты, как и счастливые семьи, похожи друг на друга, а вот неудачные проекты завершаются по-разному.

Хотя утверждение, что если человек был владельцем обанкротившейся фирмы, то это указывает на его большой опыт, уже набило оскомину, я не слышал, чтобы подобные идеи распространялись на программирование.

(Если что, у меня есть опыт в обеих сферах. Мои попытки заниматься бизнесом проваливались так же часто, как программные проекты.)

Коммерческие провалы обычно дают тебе вполне конкретный опыт. Ты понимаешь важность экономии или становишься более решительным. Но в программировании ценен не столько опыт неудач, сколько знания, полученные во время работы над проектом, который, скорее всего, провалится.

Когда я начинал программировать, много времени тратилось на бесплодные попытки написания самых разных замечательных вещей: операционных систем, файловых систем, виртуальных машин, дополнительных реализаций сетевых протоколов, интерпретаторов, JIT-компиляторов. Большинство моих творений так и не заработало, а то, что заработало, справлялось со своими задачами крайне посредственно. Даже если игнорировать технические аспекты, большинство моих попыток с самого начала было обречено на неудачу. Я не знаю, насколько велика вероятность написать новую операционную систему, но она крайне мала.

**НЕУДАЧИ И КОПИРОВАНИЕ (продолжение)**

Однако для меня эти проекты являются самыми интересными в программировании. Это фундаментальные задачи, касающиеся разработки программного обеспечения, причем очищенные от всего постороннего. Все они связаны с поиском компромисса между пространством, быстродействием, надежностью и сложностью, без сглаживания углов или некорректного API.

Это теоретические задачи, в решение которых можно погружаться месяцами, так и не получив реального результата, — именно это я регулярно демонстрировал.

Точно не знаю, по каким причинам, но люди, в настоящее время изучающие программирование, обычно не занимаются подобными вещами.

Возможно, это связано с увеличением количества сетевых приложений. Несколько дней назад на сайте Hacker News кто-то поинтересовался, нужны ли в настоящее время хоть кому-нибудь программы на стороне клиента. Это некоторое преувеличение, но оно недалеко от истины. Да-да, я тоже считаю, что веб-приложения — это очень круто.

Но с точки зрения программирования такая тенденция имеет свой минус. При написании веб-приложений практически никогда не приходится сталкиваться с серьезными техническими проблемами, пока дело не доходит до реально больших масштабов (мы не берем в расчет совместимость с Internet Explorer 6).

Другими словами, барьер, за которым программиста подстерегают неудачи, стал выше. И на первых порах человек работает вполне успешно.

И вот из-за этой направленности на программы, ориентированные на работу в Сети, я считаю, что нужно специально искать проекты с высокой вероятностью неудачи.

А что с копированием? Любой вам скажет, что для превращения в хорошего программиста нужно читать по-настоящему хороший код. Допускаю, что никто не подразумевает чтения в буквальном смысле (это слишком скучное занятие), но все равно такой подход остается, по сути, неверным: ведь он пассивен. Вместо него я предлагаю активно, широко и беззастенчиво заниматься копированием.

Разумеется, это относится ко многим вещам. Хантер С. Томпсон не просто читал хорошие книги; он перепечатывал Хемингуэя и Фитцджеральда. А старейшие из известных рукописей Баха являются переложениями произведений других органистов. Возможно, более известным является тот факт, что Гейтс в Гарварде доставал чужие программы из мусорной корзины.

**НЕУДАЧИ И КОПИРОВАНИЕ (продолжение)**

Понять, как это может помочь, не так уж сложно. Копирование нарабатывает мышечную память. Ты начинаешь чувствовать нюансы и форму оригинала — детали, которые при быстром сканировании теряются.

В случае с кодом есть также менее очевидная — но значительная — выгода. Копирование позволяет продвинуться в проектах, которые с большой вероятностью должны были потерпеть неудачу. Это может быть как непосредственное переписывание реализации хэш-таблицы (что позволило повысить качество первого написанного мною интерпретатора), так и разработка дизайна на основе чьей-то работы (как, скажем, операционная система Linux написана на основе Minix).

В своем лучшем проявлении заколдованный круг из неудач и копирования ведет к постепенному медленному самосовершенствованию. Ты берешься за что-то сложное, сталкиваешься с неразрешимой задачей, копируешь чужое решение и в итоге получаешь представление о том, как следует действовать в подобных случаях.

Среди этого безудержного грабежа, во время которого ты без оглядки поглощаешь различные приемы, зачастую обнаруживается возможность совместить их друг с другом совершенно непредсказуемым образом. Я не совсем понимаю, что имел в виду Пикассо, говоря: «Хорошие художники копируют, в то время как великие художники крадут». Возможно, он сознательно пытался быть порочным, но основной смысл этой фразы именно тот, который я всегда подразумевал.

Программирование переполнено странными идеями. Использование более коротких и менее осмысленных имен часто дает в целом более читабельный код. Наиболее мощные языки обычно поддерживают намного меньше концепций, чем более простые. А неудачи и копирование порой лучше всего ведут к созданию успешной и оригинальной работы.

# ЧАСТЬ IV
# МАРКЕТИНГ… НЕ ТОЛЬКО ДЛЯ БИЗНЕСМЕНОВ

Ты — самый талантливый из известных тебе разработчиков программного обеспечения. Твой творческий потенциал становится источником практически бесконечного потока элегантных проектов. Твоя архитектурная прозорливость не имеет себе равных в рабочем коллективе. Ты пишешь код быстрее и аккуратнее всех когда-либо работавших в этой фирме.

*Ну и что?*

Многие программисты — особенно наиболее тщеславные — пребывают в заблуждении, считая, что их способности должны быть очевидны любому руководителю или работодателю. Эти ошибочные убеждения они с легкостью прикрывают выдуманными нравственными критериями: мол, они слишком *скромны*, чтобы продавать свой талант. Лезть из кожи вон, демонстрируя свои способности, они считают *подхалимством*. Ни один уважающий себя программист в жизни не опустится до подхалимажа перед начальником.

Но это всего лишь отговорка. На самом деле они опасаются.

Большинство программистов — ботаники. В школе они были последними, кого брали в любую команду. Они избегали социальных контактов и с треском проваливались, когда обстоятельства к ним вынуждали. Неудивительно, что такие люди боятся стать объектом критики, демонстрируя кому-либо свои способности.

На минуту отбросим свое неверие и представим, что вся эта чепуха насчет моральных норм не так уж бессмысленна. Какими бы ни были изначальные побуждения, скрывать информацию о своих способностях и навыках глупо. Посмотри на это таким образом: тебя наняли для разработки программного обеспечения, повышающего доход компании. Задача начальника — сформировать рабочие группы, способные принести максимальную выгоду. А как это сделать, не зная, какую работу способен выполнять каждый из подчиненных?

Как мне недавно сказал один начальник, если кто-то делает нечто совершенно фантастическое, но об этом никто не знает, можно считать, что он не делает ничего. Это может прозвучать жестоко, но с точки зрения компании такой подход является вполне осмысленным. Начальник не может пристально следить за всеми действиями своих

подчиненных в течение каждого дня. И ни компании, ни их сотрудники не хотели бы, чтобы начальство тратило свое время подобным образом. Компании нужно, чтобы руководитель концентрировался на ситуации в целом, а не на мониторинге отдельных заданий. А сотрудники (особенно программисты) ненавидят, когда каждый их шаг контролируется.

Короче говоря, ты можешь производить лучший в мире продукт, но если ты не сделаешь ему хоть какую-то рекламу, его никто не купит. Мы все знаем — особенно в мире информационных технологий, — что далеко не всегда выигрывает самый лучший продукт. Помимо качества успех на рынке обеспечивается еще массой факторов. Не стоит забывать эту истину, когда речь заходит о рынке труда.

*Достаточно... а мне-то что делать?*

На первый взгляд маркетинг — это просто. У тебя всего две цели: рассказать людям о своем существовании и дать им понять, что ты тот, кто может решить проблемы, мешающие им спать ночами. Это относится не только к рынку труда в целом, но и к фирме, в которой ты уже работаешь. Не стоит думать, что раз ты уже трудоустроен, руководство знает, что ты собой представляешь. Более того, не стоит полагать, что если начальник помнит твое имя, то он имеет хотя бы отдаленное представление о твоих способностях.

Эта часть книги поможет тебе не только показать начальству, на что ты способен, но и расширит твои горизонты до отрасли в целом. В конце концов, задача этой книги — повысить спрос на твою персону на рынке труда. Поговорим же о том, как на практике поднять свою конкурентоспособность.

## Совет 33
## Восприятие и репутация

Играть в идеалиста, делая вид, что тебе все равно, что о тебе думают другие, очень удобно. Но это игра. Нельзя позволять себе поверить в нее. Тебя *должно* заботить чужое мнение. Репутация — это реальность. Смирись с этой мыслью.

Возможно, ты слышал старый философский вопрос: «Если дерево падает в лесу, где никого нет, производит ли оно шум?» Правильный ответ: «А какая разница?»

Я считаю, что падение сопровождается шумом. На метафизическом уровне это не очень впечатляющий ответ, но здравый смысл подсказывает, что шум при этом будет. Однако если звук никто не слышит, то сам факт его возникновения по большому счету не имеет значения.

То же самое касается твоей работы. Если ты сделал нечто сногсшибательное, но этого никто не заметил, можно ли засчитать твое достижение? А какая разница? Никакой.

Вращаясь в субкультуре индийской бюрократии в области информационных технологий, я удивлялся, почему люди *не понимают* эту простую истину. Почти все, с кем мне приходилось иметь дело, не понимали, почему имеет значение, к примеру, осведомленность руководства об их деятельности. Если *ты* знаешь, что работаешь лучше, чем имярек, это должно отражаться на оценке твоих показателей, рейтинге и зарплате. Они обманывали себя, думая, что то, как их воспринимают другие, имеет отношение к *истине*, какой бы она ни была.

Оценка деловых качеств никогда не бывает объективной.

Истина… что это такое? Кто ее определяет? Что такое хорошо, и что такое плохо в широком понимании?

Не бывает абсолютно хорошего и абсолютно плохого, по крайней мере в ситуации выбора лучшего среди творческих личностей. Как определить, что делает хорошей песню? Или картину? У тебя могут быть собственные критерии, но, возможно, я с ними не соглашусь. Они субъективны.

Ужасные отделы кадров, боящиеся рискнуть, в ужасных фирмах, не желающих идти на риск, впустую тратят время, определяя объективные показатели нанимаемых работников. Порой они даже внедряют «объективные» системы оценки. Все члены моей группы в Индии

думали, что *они* хотят быть аттестованными таким способом. Просто потому, что никогда раньше с подобным не сталкивались.

Не существует объективной меры квалификации работника умственного труда и качества его работы. Давай, попробуй со мной поспорить. А потом как следует подумай над своими аргументами. Видишь их слабые места?

Итак, если измерение твоих положительных качеств в компании (в отрасли или вообще на рынке труда) субъективно, что из этого следует? Из этого следует, что о тебе всегда будут судить исходя из чужого *восприятия* твоей персоны. Твое потенциальное повышение в должности или увеличение зарплаты — и даже решение, должно ли твое имя и дальше оставаться в платежной ведомости, — полностью зависит от восприятия тебя другими людьми.

Субъективность, основанная на *личных* предпочтениях, означает, что ты никогда не можешь рассчитывать на совпадение мнений. Впечатление на людей производят самые разные факторы. Некоторым нравится жесткая структурированность, в то время как другие предпочитают свободное творчество. Кто-то любит общаться посредством электронной почты, а кому-то нужны личные беседы или отчеты по телефону. Одни руководители с благосклонностью смотрят на инициативных сотрудников, а кто-то хочет полного подчинения. Я говорю «трах-тарарах», а ты — «трам-тарарам».

И дело не сводится только к персональным предпочтениям. Восприятие людей, находящихся с тобой в разных отношениях, базируется на качествах, обеспечивающих успешность этих *конкретных* отношений. Если я руководитель проекта, то твоя способность писать код будет для меня менее важной, чем твои коммуникативные навыки. Если я твой коллега-программист, твои способности и творческий потенциал впечатлят меня куда больше, чем выполненная работа. Но с позиции твоего непосредственного начальника могу сказать, что твои способности не имеют для меня практически никакого значения, если ты не направляешь их на *решение конкретных задач*.

Наши культурные особенности приучили нас *полагать*, что управление чужим восприятием — деятельность до некоторой степени

недостойная и нечестная. Но, как легко заметить, это всего лишь способ достичь нужного результата. Отчетливо поняв, какие факторы влияют на восприятие тебя другими людьми, ты получишь ключ к тому, чтобы превратить этих людей в счастливых заказчиков. Ты не будешь пытаться впечатлить заказчика, не имеющего технического образования, своими способностями в объектно-ориентированном проектировании. Может быть, ты гений проектирования, но без умения общаться по существу дела и без способностей сделать работу в срок тебя будут воспринимать как отвратительного специалиста. И это не вина заказчиков. Ты и *в самом деле* отвратительный специалист.

Наша репутация имеет большое значение. Именно она обеспечивает нам трудоустройство (или его отсутствие). Она позволяет как сделать карьеру, так и застрять в одной должности на долгие годы. Она обеспечивает повышение в должности и увеличение зарплаты. Чем скорее ты поймешь это и научишься управлять своей репутацией, тем скорее ты окажешься на верном пути.

### Действуй!

1. Факторы, влияющие на репутацию, зависят от целевой аудитории. Вряд ли мама по достоинству оценит твои навыки ООП, а вот коллеги вполне могут это сделать.

   Умение определить важные для конкретных отношений факторы важно для построения надежной репутации среди коллег. Подумай, на какие типы делятся твои отношения с другими сотрудниками фирмы. Скорее всего, у тебя есть пара коллег, выполняющих ту же работу, что и ты, у тебя есть начальник, кроме того, ты можешь общаться с одним или двумя заказчиками и руководителем проекта.

   Составь список этих групп (или других структур, с которыми тебе приходится иметь дело на рабочем месте). Рядом с каждым пунктом списка перечисли качества, которые определяют твою репутацию среди членов этой группы. Вот пример:

| Группа | Качества, определяющие репутацию |
|---|---|
| Коллеги | Технические навыки, социальные навыки, работа в команде |
| Начальник | Лидерские способности, ориентированность на заказчика, коммуникационные навыки, умение завершать начатое дело, работа в команде |
| Заказчики | Ориентированность на заказчика, коммуникационные навыки, умение завершать начатое дело |
| Руководитель проекта | Коммуникационные навыки, умение завершать начатое дело, продуктивность, технические навыки |

Составь по такому же принципу собственный список и посмотри на результат. Насколько тебе нужно изменить поведение? В чем ты уже скорректировал взаимодействие с каждой из групп? А в чем твое поведение пока не изменилось нужным образом?

## Совет 34
## Проводник в неведомое

Рискну напомнить очевидный факт. Наиболее важным качеством, позволяющим проявить себя на рабочем месте, является способность к коммуникации. Времена взлохмаченных хакеров, припавших к терминалам в глубине серверной комнаты и набирающих код при свете монитора, давно прошли. Периодического односложного отклика в промежутках между проявлением своих поразительных способностей уже недостаточно.

Возможно, мое предложение тебя шокирует, но попробуй взглянуть на себя с точки зрения начальника или заказчика (далее в этом разделе и того и другого я буду называть *заказчиком*).

Они отвечают за что-то крайне важное, но в конечном счете вынуждены доверить внедрение этого важного каким-то неопрятного вида парням, занимающимся информационными технологиями. Они делают все, что могут, чтобы поспособствовать работе, но в целом зависят от

милости этих программистов. Более того, они не имеют представления о том, как воздействовать на этих программистов, и даже о том, как грамотно обсудить с ними проводящиеся мероприятия. Какой навык члена группы будет больше всего интересовать их в подобной ситуации? Готов спорить на гонорар от этой книги, что им будет совершенно все равно, помнит ли этот человек о новейших образцах проектирования и сколько языков программирования он знает.

*Они будут искать того, кто поможет им не переживать по поводу их проекта.*

У всех этих начальников и заказчиков есть маленькая тайна: они тебя *побаиваются*. И не просто так. Они умны. А ты говоришь таинственным языком, которого они не понимают. Твои саркастические комментарии (порой их ты отпускаешь даже непреднамеренно) заставляют их чувствовать себя глупо. А твоя работа зачастую представляет собой последнее и наиболее важное препятствие между концепцией проекта и его рождением.

Твои заказчики тебя боятся.

Для клиента ты должен стать проводником по суровым землям мира информационных технологий. Ты обеспечиваешь комфорт заказчиков во время прогулки по незнакомым местам. Ты показываешь им достопримечательности и отводишь их туда, куда они хотят, избегая сомнительных уголков города, местонахождение которых ты заранее разведал.

Непрограммисты в среднем так же умны, как и программисты. (Другими словами, хотя большинство из них не поражает нас уровнем интеллекта, попадаются и довольно умные люди.) Вполне возможно, твой заказчик не глупее тебя, просто он не знает, как написать компьютерную программу. В этом нет ничего страшного. Ведь, скорее всего, ты не имеешь представления о многих вещах, которыми он занимается ежедневно. Поэтому вас двое и вам обоим платят, чтобы вы приходили на работу.

Я упомянул про интеллект, потому что компьютерщики слишком часто считают тех, кто не умеет работать с компьютером, не слишком умными людьми. Открыто высказывая подобную точку зрения, че-

ловек выглядит глупо, как и в случае с любым другим предрассудком. Тем не менее это чувство так прочно укоренилось во многих из нас, что мы его попросту не замечаем. Попытки напрямую его контролировать не работают.

Я советую посмотреть на подобное отношение с противоположной стороны. Вместо того чтобы воображать себя компьютерным гением, спускающимся с небес, чтобы вытащить бедного заказчика из болота, поменяйся с ним местами. Если ты трудишься, например, в области страхования, представь заказчика крупным специалистом в страховании, который даст *тебе* информацию, необходимую для выполнения *твоего* задания.

Исходя из сказанного ты должен помнить, что при обсуждении связанных с программным обеспечением вопросов важно выбрать правильный тон. Существует деликатный баланс между избыточными техническими подробностями и чрезмерным упрощением.

Ты можешь удивиться, почему мы так много говорим об отношении к заказчикам, если в этой главе рассматриваются способы продвинуть себя на рынке труда. Но если ты трудишься в типичной фирме, связанной с информационными технологиями, бóльшая часть средств, позволяющих держать тебя на рабочем месте, появляется благодаря коммерческой деятельности — той самой, которой занимаются твои заказчики. Именно заказчик, который не представляет работы ни с кем, кроме тебя, является твоим лучшим адвокатом во время принятия решения о продвижении в должности и кадровых перестановках. А теперь представь, как на твою судьбу может повлиять заказчик, считающий, что ты слишком важничаешь. Заказчик представляет потребности бизнеса, а тебе платят за удовлетворение этих потребностей. Не забывай об этом.

### Действуй!

1. *Проверяй себя*. Ты сварливый, старый, погрязший в коде человек, которого все боятся? Ты *уверен*? Люди не осмеливаются тебе в этом признаться?

   Найди в своем почтовом ящике примеры писем, отправленных менее подкованным в техническом отношении коллегам, руководителям и заказчи-

кам. Перечитай их и постарайся оценить с точки зрения получателей. Если с момента отправки письма прошло некоторое время, пора взглянуть на себя со стороны.

Более того, покажи эти письма маме. Скажи, что кто-то из твоих коллег отправил их заказчику, и спроси, какие эмоции она испытывает при чтении.

2. *Взгляд с другой стороны*. Попробуй поставить себя в ситуацию, когда ты ничего не понимаешь, а значит, целиком зависишь от сторонних экспертов.

   Представь, что у тебя две левые ноги и ты оказался в футбольной команде. Или у тебя два левых указательных пальца, а ты — член национальной сборной по вязанию. Какого отношения со стороны товарищей по команде ты ждешь?

## Совет 35
## Велик могучим русский языка

Дни немногословных угрюмых программистов прошли. Если фирма хочет проблем с программистами, она переводит их на другой континент, в другую временну́ю зону и общается с ними только по электронной почте и телефону.

Итак, вопрос передачи информации является одним из самых важных. В списке задач, которые тебе нужно решить, чтобы сохранить за собой рабочее место, эта может показаться несколько надуманной, глупой или тривиальной. Возможно, у тебя возникнет чувство, что ты вернулся в школу на урок родного языка. Ничего страшного. На этот раз ты сможешь как следует сконцентрировать свое внимание.

Начнем с самого скучного: с важности грамматики и орфографии. Скорее всего, ты защитил диплом по какой-нибудь зубодробительной специальности, а я завожу речь об орфографии. *Какое нахальство!*

Но, к примеру, в Соединенных Штатах это настоящая проблема.

Согласно отчетам Национальной комиссии по грамотности, более половины участвовавших в опросе компаний учитывают способность грамотно излагать, принимая решения о найме на работу и повышении по службе. А сорок процентов опрошенных компаний из сектора

услуг ответили, что только треть их новых сотрудников обладают необходимым уровнем грамотности[1].

Отступив назад и посмотрев на общую картину, мы констатируем, что навыки письма являются необходимыми *и* достаточно редкими.

Как ты знаешь, рабочая сила сейчас распределена по всему земному шару. И если эта тенденция продолжится, настанет время — а для некоторых оно уже настало! — когда *бóльшая часть* делового общения будет осуществляться через службы мгновенных сообщений или по электронной почте.

Тебе придется *много* писать. И если изрядная часть твоей работы будет связана с писаниной, лучше научиться делать это хорошо. Более чем когда бы то ни было твоя репутация будет формироваться на базе твоих способностей грамотно писать. Ты можешь быть замечательным кодером, но без умения выразить свои мысли словами у тебя не получится результативно функционировать в «распределенной» команде.

Умение писать дает как внешнее впечатление о тебе, так и реальное понимание того, как работает твой ум. Если ты не можешь выразить свои мысли на родном языке таким образом, чтобы другие их без проблем поняли, как поверить в то, что ты в состоянии это сделать на языке программирования? Умение сформировать идею и привести читателя через мыслительный процесс к логическому заключению немногим отличается от умения разработать четкий проект и реализовать систему, доступную для понимания тому, кто будет заниматься сопровождением.

> Ты представляешь собой только то, что можешь объяснить.

И дело даже не в том, что о тебе подумают. Если члены рабочей группы находятся в разных временны́х зонах, на большом расстоянии друг от друга, электронные письма могут быть единственным средством объяснить, что ты сделал, каким образом ты что-то спроектировал или над чем должны поработать твои коллеги.

Коммуникация, особенно посредством писем, является тем бутылочным горлышком, через которое должны пройти все твои велико-

[1] http://www.writingcommission.org/report.html

лепные идеи. И их воплощение зависит от твоего умения все четко объяснить.

**Действуй!**

1. Начни дневник разработчика. Ежедневно делай короткие записи, объясняя, над чем ты работаешь, обосновывая собственные проекты и подвергая тщательной проверке технические или профессиональные решения. Хотя этот дневник предназначен только для тебя, уделяй пристальное внимание качеству текста и способности четко выражать мысли. Время от времени критически перечитывай старые записи. В зависимости от того, что тебе понравилось или не понравилось в старых записях, корректируй более новые. Это не только улучшит твои навыки письменной речи, но и позволит понять причины принятия решений, а также даст материал, к которому можно будет обратиться, когда потребуется уточнение, как или почему ты что-то сделал раньше.
2. Научись быстро набирать текст. Если ты пока не умеешь печатать вслепую, запишись на курсы или загрузи обучающую программу. Если набор текста не вызывает затруднений, тебе будет проще выражать свои мысли в письменной форме. И разумеется, научившись быстро печатать, ты сэкономишь время при рабочей переписке.

## Совет 36
## Будь рядом

У тебя есть возможность лично пообщаться с руководством и коммерческими заказчиками. Не упускай ее.

В Бангалоре, когда я занимал должность главного технического директора, у меня был неприятный опыт докладов начальнице, которую я не любил (она платила мне той же монетой) и которая к тому же находилась в США. Мы вели напряженные разговоры по телефону поздно ночью или рано утром, раздражаясь из-за фоновых шумов и внезапных обрывов связи. Я писал длинные письма, пытаясь сгладить проблемы, порожденные расстоянием и разницей во времени, но они игнорировались. Стоило пожаловаться на недостаток внима-

ния, и в ответ я получал критику моих писем. Положение казалось безвыходным.

Как раз в это время в фирме оценивали годовую производительность: руководители приводили сильные стороны своих подчиненных и (так называемые) потребности развития. Первой в списке моих потребностей развития за этот год стала такая вещь, как *присутствие*.

В этом контексте присутствие означает некое корпоративное понятие, описывающее несколько расплывчатое лидерское качество. Это не поддающееся измерению качество особенно ощущается в ситуациях, требующих личного общения. Сюда относится аналогично не поддающееся измерению умение вести себя как лидер.

Обсуждая по телефону обзор эффективности моей работы с моей любимой начальницей, я выключил звук, как только она произнесла слово «присутствие». Я не хотел, чтобы она слышала мой смех. Хотел бы я знать, расслышала ли она интонацию, порожденную гримасой, которую я не мог стереть со своего лица весь остаток нашего разговора. Мы оба знали, что *реальная* проблема была связана с присутствием в более общем смысле: меня просто не было в США, где находились все остальные.

Большинство из тех, кто делился со мной своими мыслями, не любили эту начальницу. Это было неудивительно, так как она практически ничего не делала, чтобы заслужить уважение. Вырисовывалась интересная картина. Действительно *плохие* отношения у нее были только с теми сотрудниками, которые работали в другой стране.

Ее отношения с сотрудниками из Индии, Венгрии и Великобритании (в порядке убывания) были натянутыми, так как мы не только физически находились далеко от нее, но и жили в другом часовом поясе. Кроме того, влияние оказывали инфраструктурный, культурный и языковой барьеры.

Казалось, что для персонала из США, который делал все, чтобы избежать контактов с этой начальницей, физическая досягаемость и периодические личные встречи изрядно смягчали ситуацию. Плюс ко всему, ступив на индийскую землю, я на собственном примере убедился в справедливости поговорки «с глаз долой, из сердца вон».

Я не просто рассказываю историю своих злоключений, но и пытаюсь научить тебя на собственном опыте. Возможность личного контакта на рабочем месте является преимуществом.

Вспомни последний случай, когда не разбирающийся в компьютерах родственник или друг попросил у тебя помощи. Ты попытался решить вопрос по телефону, но его неспособность следовать инструкциям все больше действовала тебе на нервы. *Если бы у тебя была возможность показать…* Думаю, теперь ты понимаешь удивительную эффективность личного взаимодействия. Ты можешь намного четче услышать своего оппонента. Можешь пользоваться наглядными пособиями, помогать жестами или рисунками на доске. Кроме того, мы, *сами об этом не подозревая*, даем дополнительную информацию даже выражением своего лица.

При личном общении мы не только обеспечиваем бóльшую продуктивность информационного взаимодействия, но и формируем более тесные связи. Возникновение такого явления, как дружба, с человеком, которого ты никогда не встречал, требует достаточно долгого времени. Пятнадцать лет назад это вообще было неслыханно. В наши дни с засильем интернета подобная дружба просто чуть меньше распространена, чем обычные очные отношения. По тем же причинам, которые снижают эффективность общения по телефону, электронной почте или через чат, мы намного медленнее заводим подобные отношения. Добавь к этому дискомфорт от неестественных переговоров по электронной почте (возможно, следующее поколение об этом уже не вспомнит), и станет понятно, что в большинстве случаев отношения, возникающие при удаленном общении, сосредоточиваются вокруг выполнения текущих рабочих заданий.

Сильные групповые отношения с эффективным высокоскоростным общением позволяют ускорить создание новых программ. В предпринимательской среде большинство важных решений принимается в личном порядке, во время обеденных перерывов и неформального общения. Это достаточно очевидные наблюдения и преимущество от принадлежности к определенным кругам также достаточно очевидно. Далеко не такой тривиальной — особенно для нас, компьютерных фанатиков, — является важность нахождения *на виду*.

Я *никогда* не хожу в банк. Все нужные операции я выполняю через интернет или банкомат. Иначе обстоят дела у моих бабушек. Практически все свои вопросы они решают непосредственно *в банке*, разговаривая с *живыми людьми*. Они не любят вести дела даже по телефону. Им попросту неудобно. Еще они знакомы с продавцами магазинов, в которых покупают продукты. Они приходят туда снова и снова и болтают с кассиром, пока тот проводит их товары через кассу. Они не думают о возможности поменять магазин (или банк), потому что выбор в данном случае обусловлен не только прагматическим расчетом затрат и удобств. В дело вступает личный фактор.

Пока мы не делегируем свою работу роботам и компьютерным программам, вся деловая активность будет зависеть от личного фактора. Мы, люди, любим общаться с другими людьми. По крайней мере некоторые из нас.

Естественный режим работы компьютерщика выглядит так: он забивается в свой закуток, надевает наушники и исчезает в собственном «пространстве», пока не наступает время обеденного перерыва. Дуглас Коупленд в романе *Microserfs* («Рабы Микрософта») рассказывает забавную историю про людей, которым пришлось покупать еду в плоской упаковке, чтобы просовывать ее под дверь офиса, в котором работал программист. Подобная целенаправленная изоляция стала частью культуры и фольклора индустрии программного обеспечения.

К сожалению, карьерному росту такое поведение не способствует. Если ты запираешься в своем офисе, доступный только по телефону (если ты соизволяешь взять трубку) или по электронной почте, работая до поздней ночи, для своего начальника и заказчиков ты ничем не отличается от сотрудников, находящихся за границей. Ты упускаешь прекрасную возможность стать *неотъемлемой* частью своей фирмы. Напоминаю, что подобное можно сделать только при *личном общении*. Для этого тебе придется вспомнить про естественную человеческую склонность к взаимодействию с другими — не по голосовой почте, не по электронной почте, не через мгновенные сообщения, а с живыми людьми.

> Познакомься со своими коллегами.

В современной распределенной среде может оказаться, что твой коллега работает в другом городе. В подобной ситуации помогают совместные путешествия и развлечения, если, конечно, подобное практикуется в твоей компании. Но лучшее, что ты можешь сделать, — это поднять трубку и позвонить руководителям и коллегам. Не используй громкую связь и не полагайся на плановые совещания. Тебе нужно сымитировать непринужденный разговор, из тех, которые ведут во время обеденного перерыва люди, живущие и работающие в одном месте, поэтому выдели для таких с виду спонтанных бесед определенное время. По возможности переведи этот разговор в личную плоскость. Пусть вопрос «как дела?» превратится в «чем ты обычно занимаешься по выходным?» Попробуй реально познакомиться с *людьми*, которые являются твоими коллегами. Это не только закрепит твое положение в фирме, но и обогатит твою жизнь.

**Действуй!**

1. Выдели один день на следующей неделе и постарайся (в разумных пределах) обойтись без электронной почты. Каждый раз, когда тебе нужно отправить сообщение, позвони адресату или (что еще лучше) зайди к нему в офис и поговори лично.
2. Составь список коллег, руководителей и клиентов, с которыми ты недостаточно общаешься. Сделай в календаре напоминалки о необходимости поговорить с ними (по телефону или лично). Беседа должна быть краткой и предметной. Используй такие беседы для обсуждения рабочих вопросов, попутно пытаясь установить личные отношения.

## Совет 37
## Разговор с бизнесменом

Мои племянники постоянно используют компьютеры. Они, собственно говоря, достаточно хорошо в них разбираются. Компьютеры позволяют им общаться с друзьями по всему миру. Их не пугают ни сервисы мгновенных сообщений, ни электронная почта, ни интернет и, разумеется,

они знакомы с офисными приложениями и прочими вещами, совершенно необходимыми старшекласснику, чтобы делать уроки.

Но вздумай я похвастаться своим винчестером Serial ATA с его 10 000 RPM, они в лучше случае изобразят небольшой энтузиазм. Скорее всего, не впечатлят их и рассказы о гигабайтах RAM и о GPU, который намного быстрее любого процессора, использовавшегося всего пять лет назад.

А вот если сказать, что они смогут запустить в полном разрешении новейшую «стрелялку» и наслаждаться красивой графикой, они сразу навострят уши.

Среднестатистического тинейджера не интересуют гигагерцы и обороты в минуту. А вот компьютерные игры им понятны и близки.

Бизнесмены тоже не интересуются гигагерцами и частотами. Им нравится, когда приложение работает быстро, так как это избавляет их от необходимости ждать, общаясь с клиентом по телефону или закрывая квартальный отчет. Но им все равно, сколько запросов в секунду в состоянии обработать серверный процесс нового пользовательского приложения.

Коммерческие предприятия и те, кто там работает, заинтересованы в *результатах* своей деятельности. Поэтому рекламировать свои достижения на любом языке, отличном от языка бизнеса, бесполезно.

> Рекламируй свои достижения в терминах, понятных бизнесу.

Ты же не станешь рекламировать что бы то ни было американцам на немецком языке. Производитель прохладительных напитков не станет продавать свой товар, рассказывая, сколько красного красителя № 8 в нем содержится. Здравый смысл подсказывает, что для продажи продукции целевой аудитории нужно говорить с ней на языке, понятном обеим сторонам.

Так что разработчику программного обеспечения необходимо строить рассказ о своих достижениях, учитывая бизнес, на который он работает. Хорошо, *ты это сделал*, но *что* это вообще такое? Почему оно имеет значение? Докажи, что это так называемое достижение не является для компании потерей времени.

Мне кажется, что если попросить тебя вспомнить достижения последнего месяца, ты вряд ли сможешь ясно сформулировать, *в чем* их главная полезность. Разумеется, ты просто выполнял распоряжения начальства, но какую выгоду это приносит бизнесу?

В *General Electric* существует легенда, что ее бывший генеральный директор Джек Уэлш любил подниматься на лифте в одном из высотных офисов фирмы. Если с ним оказывался кто-либо из сотрудников, Уэлш спрашивал у напуганного такой встречей человека: «Над чем вы сейчас работаете?», а затем (вот она — опасная часть) «А какая от этого польза?» В итоге всем служащим приходилось заранее заготавливать *небольшую речь* для поездки в лифте. Просто на всякий случай.

Что ты ответишь, если твой босс нежданно-негаданно задаст тебе этот вопрос? Даже если у тебя будет пара минут на подготовку, сможешь ли ты объяснить, как повлияет на бизнес задание, над которым ты работаешь сейчас или которое завершил на прошлой неделе? Сможешь объяснить это словами, которые далекий от техники руководитель высшего звена сможет не только понять, но и *оценить*?

**Действуй!**

1. Перечисли свои недавние достижения. Напиши, какую выгоду бизнесу принесло каждое из них. Если для каких-то пунктов списка ты *не можешь* этого сделать, расспроси своего начальника или заслуживающего доверия знакомого.
2. Подготовь собственное выступление для «лифта» и запомни его.

## Совет 38
## Меняй мир

Худшее, что о тебе могут спросить на работе «А чем он (или она) вообще занимается?» Такая постановка вопроса подразумевает, что человек понятия не имеет о достигнутых тобой *результатах*.

Как ни грустно признаваться, но я понятия не имею о достижениях большинства сослуживцев, с которыми работал в крупной компью-

терной фирме. Люди просто не думают в подобных категориях. Они приходят на работу, выполняют предписанные дела и уходят домой. Все, что после них остается, — это фрагменты кода, документы и переписка по электронной почте.

Так происходит, когда ты появляешься на работе, не имея *цели*. Ты сидишь, ожидая, что тебе прикажут сделать. И после выполнения задания в курсе проделанной тобой работы оказываются только те, кто тебе ее поручил. Это нормально, если пределом твоих мечтаний являются розничные продажи или в крайнем случае работа программиста где-то в офшорах.

Но если ты хочешь стать разработчиком программного обеспечения в стране с высоким уровнем жизни, на работу следует приходить с определенной целью. Твои действия должны возыметь некий эффект, заметный не только для тебя или влияющий на твою работу (это само собой разумеется). Твои действия должны приводить к видимым переменам в рабочей группе, организации или фирме.

Изменение может быть небольшим. Ты можешь пропагандировать модульное тестирование, зарождая в душах немытых масс из программистского болотца твоей фирмы желание тоже этим заняться. Можно пойти дальше и попробовать ввести радикально новую технологию, которая позволит создавать более быстрые и дешевые системы.

> Имей цель. Убедись, что окружающие о ней знают.

Действуя так, ты чувствуешь внутреннюю *потребность*. Ты не можешь молча смотреть, как твои коллеги делают что-то неправильно. Ты знаешь, что бывает лучше и что ты *должен* все поменять.

Без сомнения, твои стремления изменить мир будут сопровождаться недовольством некоторых коллег. Если ты отстаиваешь правое дело, на них можно не обращать внимания. Не дергайся по этому поводу и не отказывайся от намеченного только из-за того, что это может задеть чьи-то чувства.

Даже если в конце концов некоторые из твоих коллег выйдут из себя, пусть тебя утешит тот факт, что эти люди больше никогда не зададут вопроса: «А чем *он* (или *она*) вообще занимается?»

Если ты не знаешь, в какой крестовый поход тебе отправиться, скорее всего, у тебя просто нет цели. Если ты уже *активно* не пытаешься оставить свой след, скорее всего, ты не оставишь его вообще.

**Действуй!**

1. Перечисли кампании, свидетелем которых ты был за время своей трудовой деятельности. Вспомни коллег, которые были их инициаторами и вели себя так, будто защищают дело всей своей жизни. Вспомни наиболее целеустремленных и результативных сотрудников фирм, в которых тебе довелось работать. Каковы были *их* цели?

   Можешь ли ты характеризовать какие-либо из этих целей как неуместные? Где проходит граница между мотивацией и фанатизмом? Ты встречал людей, которые пересекли эту границу?

## Совет 39
## Пусть твой голос услышат

Идеи, которые мы до этого рассматривали, были достаточно консервативными и вращались вокруг получения признания за выполняемую работу. Затронутые ранее темы особенно важны людям, которые хотят, чтобы их заметили, мечтают о продвижении или хотя бы о сохранении своего рабочего места в престижной фирме.

*Но как же это все скучно!*

Мир меняется. Если ты хочешь планировать свою жизнь, придется думать масштабнее, чем раньше. В качестве краткосрочной карьерной цели *вполне допустимо* мечтать о переходе с должности программиста уровня 23 на должность программиста-аналитика уровня 24, но как современный человек ты должен выходить за пределы следующего повышения и даже за пределы текущего места работы.

Пусть твои устремления будут амбициозны. Воспринимай себя не как программиста в определенной фирме — в конце концов, вряд ли ты проработаешь на одном месте до конца своих дней, — а как представителя целой отрасли. Ты творческая личность, художник. Тебе есть

чем поделиться кроме приложения для учета расходов, которое ты разработал для отдела кадров, или ошибок, найденных тобой в системе отслеживания выпусков ПО.

Компаниям нужны специалисты. И хотя резюме с длинным списком проектов прекрасно свидетельствует об уровне твоего опыта, лучше, если человек, проводящий собеседование, о тебе уже слышал. Особенно здорово, если он знает тебя как автора статей или книг или слышал твое выступление на конференции. Разве *ты* сам не хотел бы нанять на работу человека, написавшего книгу о технологии или методе, который в твоей фирме пытаются внедрить?

В предыдущей жизни я был профессиональным саксофонистом и много выступал в клубах Мемфиса и его окрестностях. Начав приспосабливаться к компьютерной индустрии, я обнаружил, что способы создания имени в музыке и информационных технологиях зачастую имеют много общего. Для пытающегося найти работу музыканта всегда верны следующие утверждения.

- (Первое и наиболее важное.) Ангажемент не всегда получает лучший саксофонист.
- Группа, в которой ты играешь, не менее важна, чем уровень твоей игры; музыканты *придают себе значительности, хвастаясь знакомствами*.
- Иногда лучших музыкантов не приглашают потому, что заранее считают, что те заняты, или просто боятся к ним подойти.
- В музыке важен сетевой эффект. Если в твою социальную/музыкальную сеть не входит конкретный человек, тебя, скорее всего, никогда не попросят вместе с ним выступить, пока в дело не вмешается какой-нибудь посредник.

В компьютерной отрасли дела обстоят аналогичным образом. Не существует объективной системы рейтинга разработчиков программного обеспечения, которая могла бы помочь при их найме на работу. Высокая квалификация — это здорово, но она не гарантирует непременного трудоустройства. В нашей отрасли, как и в музыке, существует большая разветвленная сеть объединенных друг с другом людей. И чем в большем количестве мест ты к ней подсоединишься, тем

выше будут твои шансы на идеальную работу. Ограничиваясь фирмой, в которой работаешь, ты серьезно снижаешь количество доступных для формирования связей.

Сделать свое имя известным и заставить о себе говорить проще всего с помощью публикаций и выступлений. Но как превратиться из рядового программиста в автора, а затем и в докладчика? Начни с интернета.

Во-первых, читай блоги. Узнай, что такое синдикация блогов и настрой агрегатор. Если ты не знаешь, с чего начать чтение, вспомни авторов твоих любимых технических книг и выполни поиск по их фамилиям. Вполне вероятно, что кто-то из них ведет блог. Подпишись на их ленту новостей и на новостные ленты людей, с которыми они связаны. По мере обнаружения ссылок на блоги других авторов твой список новостных лент будет расти.

Затем заведи собственный блог. Это можно сделать на базе одного из многочисленных бесплатных сервисов. Дальше все очень просто. Начни с написания своего мнения о постах из агрегатора (разумеется, со ссылками на них), которые показались тебе интересными. Постепенно ты обнаружишь, что блогосфера представляет собой все ту же социальную сеть — миниатюрную копию карьерной сети, которую ты сейчас начинаешь строить. В конечном счете твои записи окажутся в агрегаторах новостных лент других людей, которые, в свою очередь, начнут писать о тебе и распространять твои идеи.

Блог в данном случае является тренировочным полигоном. Пиши, как будто это колонка в твоем любимом журнале. Отрабатывай навык письменной речи. Постепенно он начнет улучшаться, что придаст тебе дополнительную уверенность.

Записи в блоге, по сути, являются примерами твоих работ, которые можно использовать на следующем этапе. Например, предлагая свои услуги специализированным сайтам и журналам или даже попытавшись написать книгу. Материала из сетевого портфолио, демонстрирующего твое умение писать тексты, будет вполне достаточно для включения в заявку на книгу. Опубликуй свои труды, и твоя сеть вырастет. Чем больше ты пишешь, тем больше у тебя будет возмож-

ностей писать. А все это, в свою очередь, ведет к возможности выступать на конференциях.

Карьеру писателя можно начинать с сетевых публикаций, а карьеру оратора — с выступлений на собраниях групп разработчиков. Если твоя работа связана с .NET, подготовь презентацию для разработчиков *Microsoft*, которые живут или работают недалеко от тебя. Если ты пишешь программы под Linux, выступи на встрече пользователей Linux. Когда дело доходит до выступлений, основным фактором успеха становится практика. Обязательно как следует продумай свою речь. Не относись к этому несерьезно. Пусть тебе предстоит выступить перед небольшой группой людей из твоего собственного города, но ведь именно здесь ты живешь и работаешь. *Хорошо* сделанная работа никогда не останется без награды. Ты обнаружишь, что при внимательном подходе к делу небольшие выступления ничем не отличаются от выступлений на крупных отраслевых конференциях. А ведь именно они станут следующим логичным шагом.

Все эти советы, как сделать свое имя известным, имеют небольшой, но крайне важный аспект. Нужно начинать раньше, чем почувствуешь себя готовым. Большинство людей недооценивает себя. Но у тебя *есть* чем поделиться с другими. Ты никогда не почувствуешь себя готовым на 100 %, поэтому ничто не мешает начать прямо сейчас.

### Действуй!

1. Если у тебя еще нет блога, заведи его прямо сейчас. Выбери одну из многочисленных бесплатных служб, предоставляющих площадки для блогов, и зарегистрируйся в ней.

   Создай на своем компьютере новый текстовый файл и составь список возможных тем. Все это со временем должно превратиться в статьи. Не ограничивайся глобальными вопросами. Подумай о том, что можно написать за десять-двадцать минут. В списке должно быть не менее десяти пунктов (но если тебя посетило вдохновение — не останавливайся на этом).

   Сохрани файл, но не закрывай его. Если тебе нужно будет перезагрузиться, открой файл снова. У тебя есть три недели. Каждый день выбирай по одному

пункту из списка и пиши статью. Не думай слишком напряженно. Просто записывай и публикуй свои мысли. Давай ссылки на другие блоги со статьями сходной тематики. При выборе темы статьи не стесняйся добавлять в список новые пункты.

Через три недели выбери две свои лучшие статьи и отправь их на какой-нибудь модерируемый пользователями сайт, например Хабрахабр. Если в твоем списке еще остались неосвещенные темы, продолжай писать.

## Совет 40
## Раскрути свой бренд

Раскрутка бренда состоит из двух частей: создания собственной узнаваемой марки и действий, направленных на то, чтобы эта марка вызывала положительные ассоциации. Признание и уважение!

В наши дни при виде свастики мы вспоминаем Гитлера и нацистскую Германию. С точки зрения раскрутки бренда нацистам можно только позавидовать. Первая часть задания выполнена — узнавание есть. Но у всех нормальных людей при этом возникают крайне негативные ассоциации, связанные, например, с холокостом. Поэтому вторая часть задачи с треском провалена. На самом деле, украв свастику у индуистов, Гитлер совершил преступление, которое стараются предотвратить все серьезно относящиеся к своему бренду компании. У индуистов, которые первыми заявили свои права на свастику (на хинди ее называют свасти), это был положительный символ процветания. Но в настоящее время на Западе этот религиозный символ дискредитирован. Его многие узнают, но мало кто относится к нему с уважением.

Диаметрально противоположным примером является Чарли Вуд[1]. Потрясающий певец, автор песен и музыкант, играющий на органе Хаммонда (Мемфис, штат Тенесси). Пять ночей в неделю он выступает в клубе на Бил-стрит. Все, кто его знает или слышал о нем, в курсе, насколько это невероятный музыкант. Все относятся к нему

[1] http://www.charliewood.us

с уважением. Когда дело доходит до ритм-энд-блюза, ему нет равных. Но практически никто не знает, кто это такой.

Никакого признания, зато огромное уважение.

*Тебе* же нужно, чтобы тебя одновременно и узнавали, и уважали. Твое имя — это твой бренд.

Эта часть книги целиком посвящена способам, позволяющим добиться уважения и признания. Сейчас ты должен понять, что комбинация этих двух вещей является ценным активом, который нужно создавать и охранять. В отличие от большого и страшного отдела корпоративного маркетинга, подающего иски против малолетних шалопаев, которые незаконно используют на сайтах корпоративные изображения или фразы, тебе не придется тратить много времени, охраняя свой бренд от *чужих* посягательств. Потому что потенциальной наиболее разрушительной силой для этого бренда являешься ты сам.

> Твое имя — это твой бренд.

Не порти впечатление своим поведением. Тщательно выбирай места, в которых появляется твое имя. Не участвуй в низкопробных проектах и не рассылай отвратительные мейлы большим группам пользователей (и не публикуй в своем блоге сомнительные посты, которые может прочитать кто угодно). Не будь идиотом. Идиотов не любят даже сами идиоты.

Самое главное, помни, что вещи, которые ты добровольно делаешь и с которыми связываешь собственное имя, влияют на восприятие твоей личности другими людьми. А в современном мире, где изрядная часть нашего общения происходит в интернете на открытых форумах, сайтах и в рассылках, большинство наших поступков кэшируется, индексируется и становится доступным для поиска — навсегда.

> Google никогда и ничего не забывает.

Ты можешь о чем-то забыть, но Google помнит все.

Всеми силами оберегай свой бренд. Защищай его от себя самого. Ведь это все, что у тебя есть.

**Действуй!**

1. *Поищи информацию о себе в Google*. Введи в строку поиска собственное имя в кавычках. Посмотри первые четыре страницы результатов (если их наберется четыре страницы). Что может подумать о тебе человек, просмотревший только ссылки с этих четырех страниц? Целиком ли ты представлен на этих первых четырех страницах результатов поиска? Нравятся ли тебе фотографии, которые фигурируют на этих страницах?

   Снова ищи, но на этот раз в форумах и рассылках, обращая особое внимание на стиль общения. Ты скандалист?

## Совет 41
## Публикуй свой код

Представь, насколько упростился бы твой поиск работы, если бы фирмы уже пользовались написанными тобой программами. Ты мог бы говорить: «О, у вас используется система Nifty++? Я могу помочь с ее поддержкой, ведь именно я ее написал». И это бы все меняло. Лица, проводящие собеседования, и специалисты по комплектации штата помнили бы тебя. Что тебе и нужно.

Всего десять лет назад этот прекрасно звучащий сценарий имел не так уж много шансов на реализацию. Сначала нужно было поработать на производителя коммерческого программного обеспечения, чтобы твое имя можно было связать с продуктом, в разработке которого ты участвовал. Но все изменилось. Для разработки популярной программы больше не нужно работать на Больших парней.

Появился еще один вариант: открытый исходный код. Программы с открытым исходным кодом получили повсеместное распространение. Когда в связанной с информационными технологиями фирме дело доходит до нового проекта, старый спор *создать или купить* теперь звучит как *создать*, *купить или*, *может быть*, *загрузить*. Под лицензией с открытым исходным кодом выходят если не целые приложения, то платформы от небольших библиотек до полнофункциональных контейнеров приложений, которые фактически становятся стандартами.

И те, кто разрабатывает эти программы, по большей части такие же люди, как и ты. Они сидят дома по вечерам и выходным, для собственного удовольствия создавая программы. Разумеется, порой за созданием и поддержкой продукции с открытым исходным кодом стоят корпорации, производящие программное обеспечение. Но бóльшую часть работы выполняют индивидуальные разработчики в качестве хобби.

Хотя многие из тех, кто вносит свой вклад в проекты, просто развлекаются и упражняются, для участия есть и вполне реальные стимулы. Это возможность подняться по ступенькам социальной лестницы сообщества. Здесь люди делают себе имя. Зарабатывают репутацию в отрасли. Даже не имея такой цели, в процессе работы люди *рекламируют* себя.

Работа над программами с открытым исходным кодом не только дает возможность создать себе имя, но и демонстрирует твою увлеченность данной областью деятельности. Даже если фирма, куда ты пытаешься устроиться на работу, не слышала о твоей программе, сам факт ее создания и выпуска отличает тебя от других. Подумай сам, если бы тебе нужно было нанять разработчика программного обеспечения, предпочел бы ты человека, который отсиживает на работе с девяти до пяти, а потом идет домой смотреть телевизор? Или же ты обратил бы внимание на того, кто влюблен в программирование настолько, что посвящает ему часы после работы и даже выходные дни?

> Любой может пользоваться средой Rails. Но только немногие могут похвастаться, что являются ее частью.

Вклад в проекты с открытым исходным кодом позволяет *продемонстрировать* свои способности. Упоминание о коде для реального проекта будет смотреться в твоем резюме куда внушительнее, чем просто *слова* о знакомстве с технологией. Любой может упомянуть в резюме Rails или Nant. Но мало кто в состоянии похвастаться *своим вкладом* в разработку Rails или Nant.

Руководство проектом с открытым исходным кодом позволяет показать не только свои технические способности. Нужны еще и навыки руководителя, ответственность за выход версий, умение вести документацию и поддерживать продукт и сообщество для сплочения

вокруг дела, к которому ты прилагаешь усилия. И если ты в состоянии *со всем этим* успешно справиться — в свободное время, в качестве хобби, — значит ты удивительным образом отличаешься от множества людей, претендующих на подобную работу. Большинство компаний не в состоянии заставить своих разработчиков заниматься всем этим и делать это хорошо даже при условии *оплаты* их труда. Более того, некоторые разработчики не справляются даже с *частью* подобных задач. Демонстрация того, что ты не только можешь делать эту работу, но и настолько увлечен, что готов заниматься ею даже бесплатно, говорит о твоей потрясающей инициативности.

Создав нечто действительно полезное, ты можешь даже стать знаменитым. Это может быть известность в узких технических кругах, например среди людей, связанных с Rails. А если повезет, то ты *прославишься* и за пределами сообщества компьютерных фанатиков, как Линус Торвальдс. В любом случае, публикация собственного кода сделает тебя *более* известным. Если слава означает, что множество людей знают, кто ты такой, значит, с каждым новым человеком, который о тебе узнал, ты становишься более знаменитым. А сообщество разработчиков программ с открытым исходным кодом представляет собой *всемирную* сеть людей, которые в процессе поиска программ в Сети могут обнаружить твое творение, установить его и начать использовать. В этом случае они узнают о тебе, а дальнейшее распространение твоей программы поспособствует распространению твоего имени и твоей репутации. Ведь это и есть суть рекламы. А это именно то, что тебе нужно.

### Действуй!

1. На конференциях Стюарт Халловей[1] проводит семинар, который называется «Refactotum». Если будет возможность принять в нем участие, крайне рекомендую это сделать. Его суть состоит в следующем. Нужно взять фрагмент программы с открытым исходным кодом, снабженный модульными тестами. Запустить эти тесты в анализаторе. Определить наименее протестированную часть системы и написать тесты для расширения покрытия. Непроверенный код часто бывает попросту непригоден для тестирования. Переработай

[1] http://thinkrelevance.com

его, сделав более пригодным к тестированию. Отправь свои исправления в качестве заплатки к программе.

Все это измеримо и может быть проделано достаточно быстро. Ты обязательно должен попытаться.

## Совет 42
## Незаурядность

Традиционный комплекс маркетинга стоит на четырех *p*: product (продукт), price (цена), promotion (продвижение) и placement (распространение). Существует мнение, что, охватив все четыре категории, ты получишь полный план сбыта продукции. При этом всем категориям нужно придать одинаковый вес.

Но какова цель маркетинга? Он должен формировать связь между производителями и потребителями товаров или услуг. Эта связь начинается с наличия информации о продукте. Традиционным механизмом передачи информации является продвижение, к которому относятся такие вещи, как реклама, рассылки и образовательные семинары.

В последнее время много говорят о так называемом *вирулентном* маркетинге, возникающем в процессе непосредственного общения. Подобное явление имеет место, когда вещь настолько примечательна сама по себе, что потребители добровольно распространяют о ней сведения. Информация распространяется как вирус, ведь каждый новый потребитель потенциально может «заразить» еще несколько человек.

Вирулентный маркетинг является более предпочтительным не только потому, что печатать рекламу и покупать время на телевидении очень дорого. Все дело в том, что своим друзьям и знакомым потребители верят куда больше, чем телевизионной и почтовой рекламе. Человек скорее купит вещь, о которой услышал от коллеги, чем нечто, восхваляемое в брошюре, выпавшей из воскресной газеты.

Магистр маркетинга Сет Годин в книге «Фиолетовая корова. Сделайте свой бизнес выдающимся!» (Purple Cow: Transform Your Business

by Being Remarkable) делает до некоторой степени очевидное заявление, что проще всего заставить потребителя заметить твой продукт, сделав продукт из ряда вон выходящим. Годин осмеливается даже утверждать, что традиционные четыре *р* уже устарели и потребителя не затрагивают прежние нецелевые стратегии массового маркетинга. Единственный способ выделиться из толпы, говорит он, — и в самом деле стать незаурядным.

В этом месте циничный читатель может начинать аплодировать. Все маркетинговые ухищрения, к которым мы можем прибегнуть, — ничто перед силой незаурядных способностей. Но перед тем, как ты скажешь: «Я же говорил», давай вспомним определение слова *незаурядный*.

Безусловно, незаурядный — не то же самое, что хороший. Незаурядные продукты, как правило, *являются* хорошими. Но далеко не все хорошие продукты незаурядны. Быть незаурядным означает каким-то образом привлекать внимание. Ты не становишься незаурядным разработчиком программного обеспечения просто потому, что умеешь делать это лучше своих знакомых разработчиков. Того факта, что ты до какой-то степени превосходишь других, недостаточно для вирулентного распространения твоей репутации. Возможно, все, к кому обратятся с вопросом, дадут тебе блестящую характеристику, но *незаурядность* означает, что люди рассказывают про тебя еще *до того*, как им зададут вопрос.

Чтобы быть незаурядным, ты должен значительно отличаться от тех, кто тебя окружает. Многие из обсуждавшихся в этой главе стратегий продвижения себя на рынке труда ориентированы именно на это качество. Выпуск успешных программ с открытым исходным кодом, написание книг и статей, выступления на конференциях могут увеличить твою незаурядность.

Если посмотреть на представленный в последнем предложении далеко не исчерпывающий список, легко заметить, что каждый пункт, который я указал как потенциально примечательный, связан с *выполнением каких-либо действий*. Ты можешь быть самым умным или самым быстрым, но быть в данном случае недостаточно. Ты должен *действовать*.

Чтобы напомнить нам, что значит быть незаурядным, Годин использует слова *фиолетовая корова*. Заметьте, *не лучшая*, *не самая плодовитая*, *не самая красивая*. Фиолетовая корова выделится на фоне множества самых лучших, самых молочных и самых красивых. Увидев стадо коров, ты первым делом обратишь внимание на фиолетовую.

Демонстрируй способности или умри!

Что может превратить тебя и твои достижения в фиолетовую корову? Не просто стань мастером своего дела — напиши об этом книгу. Напиши генератор кода, который даст возможность за пять минут сделать то, что раньше занимало неделю. Не довольствуйся уважением коллег, стань наиболее признанным авторитетом своего города в выбранной тобой области, проводя семинары и практические занятия. В следующий проект добавь нечто ранее немыслимое.

Не позволяй себе быть *просто* лучшим среди других. Стань личностью и делай вещи, о которых *не смогут* не говорить.

**Действуй!**

1. Начни с малого, но постарайся сделать в рамках текущего проекта или работы *нечто* незаурядное. К примеру, можно поэкспериментировать с незаурядной продуктивностью. В расписании проекта зачастую много зазоров. Найди задачу, на которую все отводят неделю, и реши ее за день. Если потребуется, работай сверхурочно. Разумеется, постоянно работать сверхурочно не стоит, но в данном случае мы экспериментируем. Выполни свою работу в незаурядно короткий срок. Посмотри, начнут ли об этом говорить. Если нет, то почему? Если да, то что именно? Оптимизируй показатели и попробуй снова.

## Совет 43
## Заводи знакомства

Когда я в подростковом возрасте играл на саксофоне, люди часто задавали мне вопрос: «Ты ведь знаешь Криса?» А я не знал. Кажется, это был *еще один* старшеклассник из Арканзаса, нацеленный на карьеру джазового музыканта. И все, с кем я встречался, банально ожидали,

что мы обязаны быть знакомы, поскольку оба имели хобби, не совсем обычное для старшеклассников.

Однажды летом мне довелось увидеть, как джазовый оркестр Каунта Бейси выступает на берегу реки Арканзас. Благодаря отчасти хорошему настроению и нехарактерному для меня мужеству я оказался за сценой и в перерыве между выступлением смог поболтать с музыкантами. Я никогда не был особо болтливым, так что в данном случае можно говорить о руке судьбы. Я стоял и разговаривал с одним из саксофонистов, и тут подошел еще один молодой человек и тоже вступил в беседу. Через несколько минут оркестр вернулся к выступлению, а мы остались стоять. «Ты Крис/Чад?» — сказали мы одновременно.

Я начал общаться с Крисом. У него была странная способность знакомиться с лучшими музыкантами нашего города. Он был всего лишь старшеклассником, но уже выступал на концертах, заменяя самых уважаемых в Литтл Рок джазовых пианистов. Как музыкант Крис был хорош — особенно для своего возраста, — но не *настолько* хорош.

Я довольно быстро понял, что происходит. Часто мы проводили в джазовых клубах по нескольку вечеров в неделю. Для такого интроверта, как я, это был не самый приятный опыт. Как только группа, которую мы слушали, объявляла перерыв, Крис замолкал на полуслове и уходил общаться с музыкантами. Он напоминал робота. Должен признать, мне было до некоторой степени неприятно такое его поведение. Он был слишком предсказуем. Разве он не докучал этим бедным музыкантам? Они устраивали передышку и вряд ли жаждали общаться с этим проклятым пацаном! Я же был вынужден либо следовать за ним, либо неловко сидеть в одиночестве, ожидая его возвращения. В дни, когда у меня попросту не было сил, я выбирал второй вариант. Но в большинстве случаев я тащился за ним и пытался подстроиться под ситуацию.

К моему большому удивлению, создавалось впечатление, что отдыхающим музыкантам беседа с Крисом и даже со мной доставляет удовольствие. Он был дьявольски настойчив и всегда спрашивал, нельзя ли сесть вместе с группой, каким бы неуместным мне это ни казалось. Еще он просил дать ему урок, что подразумевало встречу

в домашней обстановке, где он слушал музыку и болтал о джазовых импровизациях. Иногда я тащился с ним, обуреваемый ощущением, что я навязываюсь.

Оказалось, что я был единственным, кого смущали такие знакомства Криса с музыкантами. Он получал приглашения сыграть за деньги на настоящих концертах с действительно хорошими группами. А я был просто парнем, составлявшим ему компанию. Он стал моим проводником к лучшим музыкантам нашего города. Единственным различием между нами был его более легкий характер.

С годами стратегия Криса «будь худшим» вкупе с его способностью откровенно навязываться людям позволила ему стать невероятным пианистом. В сущности, он пробил дорогу к выступлениям с известными джазовыми музыкантами. Я же так и остался его знакомым. Он приглашал меня участвовать в концертах, получающих широкую огласку, а у меня не было возможности ответить ему тем же.

С тех пор мне довелось столкнуться с подобным поведением классических музыкантов, американских буддистов, разработчиков программного обеспечения и даже офисных работников. Крис называл это «прилепиться», что делало его поведение еще более отвратительным в моих глазах. Но мораль истории такова: люди, достигшие реальных высот, не имеют ничего против твоего желания с ними познакомиться. Людям нравится, когда их ценят, они любят говорить на темы, которыми увлекаются. Тот факт, что перед нами профессионал, гуру, лидер или известный автор, не отменяет того, что это всего лишь человек, которому нравится общаться.

> Между нами и профессионалами стоит страх.

Судя по моему опыту (и экстраполируя его на окружающих), самым серьезным барьером между нами, смертными, и теми, кем мы восхищаемся, является наш собственный страх. Общение с умными и обладающими хорошими связями людьми, которые могут нас чему-то научить или помочь с поиском работы, является, возможно самым лучшим способом самосовершенствования, но большин-

ство из нас боится даже попробовать. Именно принадлежность к дружному профессиональному сообществу позволяет музыкантам, художникам и другим людям искусства сохранять свои преимущества и эволюционировать с течением времени. Эти гуру являются коммутирующими узлами социальной и профессиональной сети. И чтобы познакомиться с ними, достаточно отбросить излишнюю скромность.

Разумеется, не нужно случайным образом приставать к таким людям с болтовней. Очевидно, что тебе нужно найти тех, с кем у тебя есть что-то общее. Возможно, ты читал написанную кем-то статью, которая оказала на тебя большое влияние. Ты можешь показать этому человеку проделанную в результате работу и получить его совет. А может быть, ты написал программный интерфейс к созданной кем-то системе. Это великолепный и вполне законный повод завести с ним личное знакомство.

Знакомиться, разумеется, можно как виртуально, так и лично. Прочная связь не станет от этого менее прочной. Герои фронта программного обеспечения рассеяны по всему миру. Про музыкальную индустрию можно сказать то же самое, хотя утверждение о возможности связаться с любым музыкантом по электронной почте сложно принять на веру без доказательств. По большому счету, музыкальный мир больше склонен к образованию локальных профессиональных сообществ, в то время как разработчики программного обеспечения могут пользоваться преимуществом общения друг с другом, невзирая на место жительства. Это позволяет легко познакомиться не только с ведущими специалистами из твоего города, но и в принципе с ведущими специалистами. Точка. Некоторые из наиболее влиятельных в сфере программного обеспечения людей доступны как по электронной почте, так и в обычном чате.

История, которая привела к появлению *данной книги*, началась с сообщения по поводу библиотеки Ruby, отправленного по электронной почте одному из ее издателей. За письмом последовали многочисленные беседы в чате. Я чувствовал себя крайне неуверенно, отправляя то первое сообщение, но оказалось, что я не слишком досадил Дэйву. И вот, теперь вы можете читать мои откровения. Спасибо, Крис.

## А ДАВАЙТЕ МЫ ПРОСТО…

*Стивен Акерс, вице-президент по информационным технологиям компании Genscape, Inc.*

Любой, кто проводит много времени на работе, знает о постоянной борьбе между информационными технологиями (IT) и бизнесом (не связанным с IT). Причина этих раздоров практически всегда кроется в неправильном толковании, непонимании и несбывшихся ожиданиях. Проблема почти ежедневно усугубляется то и дело повторяемыми фразами, которыми пользуются представители обеих групп.

В области информационных технологий наибольшее раздражение вызывает фраза «А давайте мы просто...» Обычно она заканчивается так: «А давайте мы просто передадим эту работу на аутсорсинг? А давайте мы просто добавим еще разработчиков? А давайте мы просто повторим то, что мы делали в последний раз? А давайте мы просто заставим приложение работать быстрее? А давайте мы просто создадим новую базу данных?»

Проблема в том, что когда связанные с информационными технологиями люди слышат подобные фразы, их настораживает слово *просто*. Оно заставляет их думать, что, с точки зрения бизнесменов, подобные пожелания очевидны, тривиальны и легковыполнимы. А значит, любая неудача будет сигналом о том, что специалист не в состоянии справиться с простейшим заданием и его следует заменить.

В результате на такие просьбы чаще всего отвечают отказом. Специалисты в области информационных технологий хотят, чтобы бизнесмены осознали не только сложность данного задания и трудоемкость его выполнения, но и изначальную неправильность высказанной идеи. В этом и состоит проблема. Ведь в конечном итоге бизнесмен уходит с чувством, что компьютерщики всегда говорят «нет», в то время как у последних создается впечатление, что бизнесмен совершенно не понимает, что хочет.

Обычно я тоже так считал. Мне казалось, что бизнесменам нужен человек, который будет разбираться в том, чем они занимаются. Но наступил момент, когда я решил завершить свою карьеру в области информационных технологий и заняться бизнесом. Я был уверен, что все мои проекты обречены на успех, потому что я понимаю, как нужно действовать.

Забавно, насколько реальность порой отличается от наших планов. В итоге я все же добился успеха на ниве бизнеса, но это был далеко не тот легкий успех, на который я рассчитывал.

**А ДАВАЙТЕ МЫ ПРОСТО… (продолжение)**

Оказалось, что мне многое предстояло узнать. Например:

Во-первых, существуют коммерческие факторы, налагающие ограничения практически на каждый проект. И именно эти ограничения порой заставляют внедрять далеко не лучшее техническое решение.

Во-вторых, устанавливаемые бизнесменами сроки зачастую далеко не так случайны, как это кажется со стороны. Много раз дата появления законченного решения оказывала непосредственное влияние на успех проекта и даже на показатели фирмы.

Усвоив эти уроки, я понял, что специалисты по информационным технологиям концентрируются не на той части вопроса «А давайте мы просто...?» На самом деле ключевое слово тут *мы*. Оно означает, что бизнесмены считают компьютерщиков важной частью своей команды. Они обращаются за помощью в решении задачи, которая в результате приведет фирму к успеху.

Поэтому, услышав в следующий раз эту ужасную фразу, сопротивляйся порыву сказать «нет». Сконцентрируйся на слове *мы* и уверенно ответь: «Да, мы можем ввести в проект еще несколько разработчиков, но это плохая идея, и вот почему...» Но не останавливайся на этом. Объяснить свою позицию мало. Нужно копнуть глубже, чтобы понять, какие коммерческие ограничения существуют в данном виде бизнеса. Со временем это сформирует твое представление о бизнес-отрасли, позволив точнее оценивать ставящиеся перед тобой задачи. Сочетание такого понимания с техническими навыками превратит тебя из посредника, который всегда отвечает «нет», в партнера, без которого бизнес просто не может существовать.

### Действуй!

1. Напиши письмо автору своей любимой программы. Поблагодари его или ее, а затем внеси предложение, задай вопрос или предпримии любую другую попытку установить личные отношения с этим человеком. Постарайся добиться ответа. Если программа распространяется бесплатно или имеет открытый исходный код, предложи свою помощь в работе.
2. Вспомни, кем из твоего города ты восхищаешься и у кого хотел бы учиться. Подумай, где ты можешь встретить этого человека (хорошую возможность предоставляют встречи профессиональных сообществ или доклады). Отправляйся туда и изо всех сил постарайся завязать с ним разговор, даже если это заставляет тебя чувствовать себя неловко — *особенно* если это заставляет чувствовать себя неловко.

# ЧАСТЬ V

# Сохраняя конкурентные преимущества

Помнишь Тиффани, блиставшую в 1980-х? Тогда она занимала верхние строчки хит-парадов, а ее песни постоянно крутили по радио. Она пользовалась колоссальным успехом, за короткое время став известной личностью.

Но когда ты в последний раз про нее слышал (если вообще слышал)? Почему-то мне кажется, что ты не сможешь дать ответа на этот вопрос. Лично я не смог.

У Тиффани было то, что требовалось для успеха в 80-е, — по крайней мере на короткое время. Миновали 90-е, и Тиффани вышла из моды. Даже если она и пыталась удержаться наверху, то не смогла поменяться достаточно быстро, чтобы удержать любовь — или хотя бы внимание — своих фанатов. Как только вектор пристрастий публики перекинулся с подросткового рока на грандж, Тиффани моментально устарела.

С твоей карьерой может случиться то же самое. Описываемый в этой книге процесс представляет собой цикл, который следует повторять до выхода на пенсию. Исследовать, инвестировать, выполнить, прорекламировать, повторить. Застряв надолго на любой итерации этого цикла, ты рискуешь моментально устареть.

Этот момент может подкрасться к тебе неожиданно, если его явно не отслеживать. И если он застает тебя врасплох, значит, уже слишком поздно. Скорее всего, Тиффани не имела представления, каким популярным со временем станет грандж. Она изо всех сил старалась остаться подростком, поп-звездой тинейджеров, и к моменту, когда первые строчки хит-парадов начали занимать композиции в стиле грандж, она необратимо вышла из моды.

В этой части мы поговорим о том, как не стать человеком, добившимся успеха всего один раз и на недолгое время.

## Совет 44
## Ты уже устарел

Многих из нас сфера информационных технологий привлекает тем, что там постоянно появляется что-то новое. Захватывающая и свежая рабочая атмосфера. Всегда есть возможность научиться чему-то

новому. И в то же время приводящая в уныние ситуация, когда наши вложения в связанные с определенной технологией знания обесцениваются быстрее, чем твой «шевроле» последней модели. Сегодня это новый популярный предмет, а завтра — устаревшее барахло с ограниченным сроком годности.

Гари Хэмел в книге «Во главе революции: как добиться успеха в турбулентные времена» (*Leading the Revolution: How to Thrive in Turbulent Times by Making Innovation a Way of Life*) рассказывает, что как только лидеры начинают почивать на лаврах, самоуспокоенность приводит к появлению «белых пятен». Чем более успешным является твой бизнес, тем выше вероятность привыкания к определенной модели его ведения, что делает его более уязвимым перед новичками, приходящими с радикальной идеей — пусть даже глупой. Ведь эта идея вполне может стать причиной того, что твоя успешная модель бизнеса начнет восприниматься как старый вылинявший свитер в клубе. Аналогично обстоят дела и с технологиями. Тот, кто в то или иное время достиг мастерства в одной из господствующих технологий, например в J2EE или .NET, может чувствовать себя уверенно. Он находится в выигрышном положении, не так ли? Все посвященные трудоустройству сайты и газеты подтверждают правильность выбранного пути.

> Твои блестящие новые навыки уже устарели.

Но будь осторожен. Успех порождает гордыню, что, в свою очередь, ведет к самоуспокоенности. Может показаться, что волна J2EE никогда не кончится. Но любая волна рано или поздно спадает или достигает берега. Слишком длительный комфорт может оставить тебя беззащитным, и ты так и не узнаешь, что делать в мире без J2EE.

Несмотря на сказанное, можно вспомнить, что люди десятилетиями пророчили смерть языку COBOL. Каждый новый участник рынка получает звание «COBOL XXI века» или что-то в этом роде. В настоящее время так называют язык Java. При всей моей нелюбви к языку COBOL я считаю, что назвать Java «COBOL XXI века», это, безусловно, комплимент. Как бы многие из нас ни мечтали о его смерти, COBOL никуда не делся и продолжает использоваться *очень долгое время*. Программисты работают с этим языком на протяжении всей

своей карьеры. А в нашей отрасли, напоминающей американские горки, это уже о чем-то говорит. Но в условиях современной экономической ситуации сложно утверждать, возможны ли другие аналогичные инвестиции.

История языка COBOL — исключение из правил. Немного найдется платформ, предоставляющих столь прочную базу для трудоустройства. Я вовсе не призываю тебя отказаться от изучения популярных технологий. Это безответственный подход. Просто хотелось бы *обратить внимание* на тот факт, что чем более популярной технологией ты владеешь, тем выше для тебя риск остаться в каменном веке этой технологии.

Мы все слышали про закон Мура, согласно которому мощность компьютеров должна удваиваться каждые восемнадцать месяцев. Количество месяцев может слегка варьироваться, но легко убедиться в том, что технология движется вперед примерно с той же скоростью, что и в 1965 году, когда основатель Intel Гордон Мур постулировал свой закон. А вместе с достижениями в области аппаратных мощностей появляются новые возможности в области программного обеспечения.

Вычислительные мощности *удваиваются*. За столь стремительным технологическим прогрессом обычному человеку угнаться сложно. Даже при наличии актуальных знаний, если ты не успеваешь освоить Новую Большую Область Деятельности, ты опоздал. Ты можешь быть впереди текущей волны, но позади следующей. В подобной среде крайне важным становится выбор момента.

Ты должен осознать, что, даже находясь на передовом крае современных тенденций, ты плетешься в хвосте более новых веяний. Начинай *заранее* думать о том, что предстоит изучить. Какие недоступные в данный момент возможности появятся через два года? Как быть, если дисковое пространство станет настолько дешевым, что его можно будет получать практически бесплатно? Что, если процессоры начнут работать в два раза быстрее? Что, если нам уже не придется заботиться об оптимизации? Как весь этот прогресс повлияет на популярность конкретных технологий?

Да, это в определенном смысле риск. Но не играть в эту игру значит сразу обречь себя на поражение. Вступив же в нее, в худшем

случае ты обогатишься знаниями, которые нельзя будет напрямую применить в своей работе в ближайшую пару лет. Поэтому лучше заглядывать вперед и идти на риск. Ведь в случае удачи ты останешься на переднем крае и продолжишь быть экспертом в передовых технологиях.

Разница между слепцом и провидцем в том, что второй заглядывает вперед и четко представляет себе направление роста.

**Действуй!**

1. Каждую неделю выделяй время на знакомство с передовыми технологиями. Тебе потребуется по меньшей мере два часа на то, чтобы узнать, какие новые веяния появились в последнее время, и начать выработку нужных навыков. Создавай простые приложения. Реализуй сложные фрагменты текущих проектов на основе новейшей технологии, чтобы понять, чем она отличается и какие новые возможности предлагает. Составь расписание для подобных упражнений и старайся его придерживаться.

## Совет 45
## Ты уже безработный

Работы, на которую тебя наняли, больше не существует. Возможно, ты все еще получаешь зарплату. Может быть, ты способствуешь росту прибыльности предприятия. Допускаю даже, что ты приводишь в полный восторг своего работодателя. Но свою работу ты уже потерял.

Полностью мы уверены только в том, что все меняется. В экономике происходят сдвиги. Рабочие места исчезают в офшоры и возвращаются назад. Предприятия ищут способы адаптации. В нашей отрасли до равновесия пока далеко. Она напоминает неуклюжего подростка в период полового созревания. Неловкого, некрасивого и меняющегося год за годом, день за днем.

Поэтому, если ты работаешь программистом, перестань считать себя таковым. Привыкай к мысли, что, возможно, это больше не

твоя профессия. Продолжай работать, но не расслабляйся. Не *отождествляй* себя с программистом. Или с дизайнером. Или с тестировщиком.

Собственно, все дело в том, что в наше время (если только другие времена вообще когда-либо бывали) небезопасно слишком сильно отождествлять себя с работой, на которую тебя наняли. В условиях постоянно меняющейся обстановки и обстоятельств цепляющийся за профессию человек создает опасный диссонанс, влияющий на качество его работы. Часто возникают ситуации, когда, к примеру, человек, желающий быть программистом, при необходимости выполняет обязанности руководителя проекта. И делает это плохо.

Ты и твоя профессия — не одно и то же.

До потери работы у тебя могли быть планы. Ты мечтал о своем продвижение по должности. Ты мог работать проектировщиком, и готов выполнять обязанности архитектора, как только появится такая возможность. Ты представлял себе весь путь в управляющей цепочке от архитектора до аналитика и руководителя группы.

Но ты уже потерял работу, и планы изменились. Они будут меняться и дальше. Каждый день. Наличие амбиций — это здорово, но не стоит слишком сильно полагаться на придуманную картинку. Ты просто не можешь себя позволить столь узко смотреть на отдаленное будущее. Целясь прямо в движущуюся мишень, ты никогда в нее не попадешь. Целиться нужно туда, где она через некоторое время окажется. И путь из одной точки в другую больше не описывается прямой линией. В лучшем случае это дуга, а скорее всего, вообще какая-нибудь загогулина.

### Действуй!

1. Если ты программист, попробуй на пару дней примерить на себя роль тестировщика или руководителя проекта. Способен ли ты в любой момент начать работать по специальностям, о которых раньше никогда не задумывался? Составь список и попробуй примерить на себя каждую из ролей. Возможно, это даже не повлияет на результат твоей работы, но ты посмотришь на нее другими глазами.

# Совет 46
# Дорога в никуда

Одной из самых больших проблем Америки является общество, ориентированное на достижение цели. Это люди, всегда ставящие во главу угла *результат*, причем не важно, чем является процесс, познанием, построением карьеры или управлением автомобилем. Мы так сконцентрированы на результате, что забываем оглянуться вокруг.

Но если вдуматься, эта концентрация на результате является логической противоположностью того, на что нам приходится тратить время. Мы практически все время что-то *делаем*, и только на короткие мгновения действительно достигаем целей. К примеру, когда ты создаешь программу, все время уходит на процесс разработки, а не на порождаемый этим процессом момент появления готовой программы.

Аналогичным образом обстоят дела с карьерой. Ее реальным смыслом являются не продвижение по службе и не увеличение заработной платы. На самом деле все упирается во время, которое ты тратишь в попытках получить эти преимущества. Или, что еще более важно, время, которое ты тратишь на работу *безотносительно* будущих достижений.

А если так, то есть если именно сама работа является сутью твоей деятельности, то ты уже прибыл к месту своего назначения. Целенаправленное, ориентированное на конечный результат мышление, которое мы обычно практикуем, ведет только от одной цели к другой. Оно не имеет логического конца. Но большинство из нас не в состоянии понять, что концом в данном случае является *сам путь к цели*.

В примере с разработкой программного обеспечения это будет момент передачи заказчику готового кода. Ему требуется приложение, и ты концентрируешься на том, чтобы его создать. Но при этом приложение никогда не будет полностью «готовым». Одна версия тянет за собой другую. Слишком сильная концентрация на конечном продукте отвлекает нас от реальной деятельности: непрекращающейся работы над новой программой.

Нацелившись на результат, ты забываешь о качестве процесса. А некачественный рабочий процесс порождает некачественный продукт, который соответствует минимальным требованиям, но непригляден внутренне. Ты думаешь о краткосрочной конечной цели, а не о неизбежной доработке продукта в будущем.

> Концентрируйся на том, что делаешь, а не на том, что должно быть сделано.

Работая спустя рукава, ты создаешь некачественный продукт, но при этом существует и обратная зависимость. Настроившись на то, чтобы побыстрее достигнуть результата, ты начинаешь корректировать рабочий процесс. В итоге в действие вступает *теория разбитых окон*. Получается замкнутый круг.

Поэтому перестань постоянно спрашивать: «Мы уже закончили?», надеясь услышать в ответ «Да». Важен не конечный пункт, а дорога к нему.

**Действуй!**

1. Тхить Нят Хань в книге «Чудо осознанности» (*The Miracle of Mindfulness*) дает совет: в следующий раз, когда тебе нужно будет помыть посуду, делай это не для того, чтобы она стала чистой. Попытайся получить удовольствие от самого процесса. Не думай о том, как ты закончишь работу. Сосредоточься на самом процессе.

   Мытье посуды является рутинным делом, от которого практически никто не получает удовольствия. Разработчики программного обеспечения в течение рабочего дня сталкиваются с множеством не менее нудных задач, таких как регистрация рабочего времени или отчет об издержках. В следующий раз, когда тебе придется заняться подобными вещами, попробуй сконцентрироваться на процессе выполнения, преодолев горячее желание быстрее с ними покончить.

## Совет 47
## Составь план

Если твоя работа в основном связана с поддержкой, легко погрязнуть в рутине и надолго зависнуть в этом состоянии. У разработчиков про-

граммного обеспечения есть подобный опыт. Занимаясь поддержкой библиотеки или приложения, которым пользуются другие разработчики, ты обречен на вечное отлавливание ошибок (или того хуже), если, конечно, у тебя нет других планов на собственное будущее. Отвечая на запросы пользователей, ты можешь внести какое-либо улучшение, но, по большому счету, изменения в код вносятся крайне неохотно, ведь считается, что работа над программой уже завершена.

Но пока приложением пользуются, работу над ним считать законченной нельзя. Аналогичным образом можно выразиться о тебе и о твоей карьере. Если ты не собираешься уходить из отрасли, без плана развития не обойтись. Если бы в *Microsoft* в свое время решили, что Windows 3.1 полностью готова, мы бы до сих пор работали на Маках. Если бы разработчики сервера Apache сочли свой сервер законченным после выхода версии 1.0, вряд ли они господствовали бы на современном рынке.

Персональный план развития позволит тебе определить, двигаешься ли ты куда-нибудь или топчешься на месте. Когда ты день за днем приходишь в один и тот же офис, работая над практически одинаковыми вещами, обстановка вокруг тебя не меняется. Нужны видимые издалека маркеры, которые дадут тебе понять, что ты не стоишь на месте. Роль таких маркеров могут играть «подсистемы» твоего продукта.

Не продумав все как следует и не составив план, нельзя заглянуть за горизонт. В частях II и III мы говорили о том, насколько важно осознанно подходить к выбору карьеры и инвестициям в собственные профессиональные качества. Казалось бы, я веду речь об одноразовом выборе области инвестиций, но на самом деле любой вариант должен стать частью общей картины. Для лучшего понимания контекста представим каждый набор навыков или умений как функцию приложения. Согласитесь, что приложение с единственной функцией — это смешно.

Более того, приложение с набором не связанных друг с другом функций способно только запутать пользователей. *Это записная книжка или приложение для обмена мгновенными сообщениями? Игра или браузер?* Составленный заранее план не только поможет придерживаться выбранного курса, но и даст более общее представление о том, что ты можешь предложить. Станет понятно, что ни одна характери-

стика не должна стоять особняком. Каждая следующая инвестиция — это часть большого целого. Некоторые вещи потрясающе смотрятся вместе. Комбинации других вещей заставляют потенциального работодателя задуматься. *Он системный администратор или специалист по компьютерной графике? Она занимается архитектурой приложений или эксперт по автоматизации контроля качества?*

Хотя освоение разных навыков — это хорошо, поскольку они расширяют твои горизонты, имеет смысл поразмыслить, какое впечатление производит твой набор навыков. Без четкого плана твое резюме будет напоминать произведение Джека Керуака[1], а не единый набор логически связанных умений. Без хорошего плана очень легко и *в самом деле* заблудиться.

**Действуй!**

1. Перед тем как строить планы на будущее, стоит описать свое *текущее положение*. Составь хронологию своей карьеры. Покажи, с чего ты начал, какие навыки и рабочие обязанности требовались на каждом этапе. Обрати внимание, когда происходили постепенные улучшения, а когда совершался большой скачок вперед. Определи промежуток времени, за который произошло твое главное повышение. Используй эту информацию в качестве базиса и строй дальнейшую карьеру. Четкая картина пройденного пути позволяет ставить более реалистичные цели. Постоянно обновляй план. Это замечательный способ продемонстрировать собственные достижения по мере движения к новым намеченным целям.

## Совет 48
## Отслеживай состояние рынка

Глупо потратить деньги на акции, цена которых постоянно меняется, и забыть об этом. Даже если ты тщательно исследовал все возможности и осознанно выбрал *область* для инвестиций, не за-

[1] Джек Керуак (Jack Kerouac) — американский писатель, поэт, представитель литературы «бит-поколения». Свой стиль сам Керуак называл «спонтанной импровизационной прозой». — *Примеч. ред.*

бывай о нестабильности рынка. Когда дело доходит до финансовых вложений, нельзя действовать по принципу «выстрелил и забыл». Даже если сегодня акции растут, завтра их котировка может резко измениться.

Кроме того, ты можешь упустить хорошую возможность. Представь, что ты нашел беспроигрышный вариант, дающий 10 % годового дохода. Это выглядит здорово, но только при условии, что доходность остального рынка не совершит скачок. Даже если рабочая лошадка, на которую ты поставил сегодня, не сойдет с дистанции, ее возможности могут оказаться совсем не впечатляющими по сравнению с открывающимися перспективами.

Не обращать внимание на меняющиеся условия рынка, значит потерять деньги, которые *можно* заработать.

То же самое касается и инвестиций в знания. Сейчас Java — оптимальный вариант. А что должно случиться, чтобы ситуация изменилась? И как понять, произойдут ли подобные изменения?

Что делать, если *Sun Microsystems* неожиданно пойдет под откос? В последние годы лидирующее положение этого монстра пошатнулось, а Java не является открытым стандартом. Хотя сейчас *Sun* ратует за концепцию открытого кода. В любой момент *Sun* может превратить собственные язык и виртуальную машину в инструмент получения прибыли. Она может добавить в Java несовместимые изменения и вызвать панику во всей отрасли.

С головой погрузившись в код, ты пропустишь нужный сигнал, и момент будет упущен. Так, казалось бы, важный навык может утратить свою первоначальную ценность на рынке труда. Конечно, это неправдоподобная ситуация, но полностью исключить ее нельзя.

Более вероятно, что, будучи полностью довольным своей работой и своими текущими навыками, ты можешь проморгать приход Следующей Великой Вещи. Десять лет назад ты бы удивился, если бы тебе рассказали, какое место займут объектно-ориентированные языки с механизмом сборки мусора. Но ведь, скорее всего, ты видел определенные знаки. Кто знает, с какой Следующей Великой Вещью мы столкнемся через десять лет?

Держи глаза и уши открытыми. Следи за новостями отрасли как с позиций бизнеса, так и с чисто технической стороны, пытаясь найти разработки, могущие привести к далеко идущим последствиям. Как говорит Тим О'Рейли[1] из *O'Reilly and Associates*, присматривайся *к самым технически продвинутым работникам*. Это те суперэнтузиасты, которые всегда находятся на самых передовых рубежах, по крайней мере в сферах, касающихся их увлечений. Утверждение Тима, справедливость которого мне довелось проверить на практике, состоит в том, что, обнаружив таких людей и понаблюдав за их увлечениями, ты сможешь предугадать, с чем нам придется иметь дело через год или два. Непонятно почему, но это действительно так.

Наблюдай за самыми крутыми гиками.

Всегда следует помнить, что в технологическом секторе любая хорошая инвестиция в какой-то момент перестает таковой быть. И если не следить за состоянием рынка, это изменение может стать для тебя большим сюрпризом. Вряд ли это то, к чему ты стремишься.

**Действуй!**

1. Попробуй в следующем году стать самым крутым гиком. Или по крайней мере *подружись* с таким человеком.

## Совет 49
## Этот толстяк в зеркале

К сожалению, у меня есть лишний вес. Это моя давняя проблема. Но когда я жил в Индии, мне удалось *сильно* похудеть. Отчасти благодаря диете. Отчасти благодаря физическим нагрузкам. Но в основном из-за болезни. После возвращения в Соединенные Штаты я начал набирать вес. Я так расстроился, что записался в тренажерный зал, и вес начал возвращаться в норму.

[1] http://tim.oreilly.com/

Я прошел несколько таких циклов. Самое интересное — я не мог отследить моменты набора и сброса веса. Я узнавал об этом, только когда мне кто-то говорил об этом или одежда начинала сидеть по-другому. Моя жена видит меня каждый день, поэтому она тоже не замечала происходящего, а посторонние люди предпочитают не затрагивать темы *лишнего* веса. А вот в Индии они не стесняются об этом говорить.

Я не могу следить за колебаниями веса, так как вижу себя слишком часто. Когда тебе постоянно что-то показывают, заметить можно только очень быстрые изменения. Если уставиться на цветочный бутон, сколько времени пройдет, прежде чем ты заметишь хоть какое-то изменение? А вот если уйти и вернуться через пару дней, разница сразу бросится в глаза.

С твоей карьерой дела обстоят точно так же. На самом деле ты ее *не замечаешь*. И это проблема. Можно каждый день смотреться в воображаемое зеркало и не замечать изменений. Кажется, что ты приспособлен к ситуации, как и раньше. Что ты столь же конкурентоспособен, как и раньше. Что твои навыки столько же актуальны, как и раньше.

А затем в один прекрасный день твоя работа (или выбранная тобой отрасль) перестает тебе подходить. В первый момент ты ощущаешь дискомфорт, но критическая точка уже достигнута, поэтому нужно либо действовать быстро, либо купить другую пару штанов (разумеется, в переносном смысле).

В случае колебания веса у нас есть некая шкала, которая позволяет легко оценить уменьшение веса (или увеличение, как в моем случае). Но шкалы для измерения конкурентоспособности или навыков разработчика программного обеспечения не существует. Как было бы легко на ее основе автоматически корректировать размер зарплат. Но ничего подобного не существует, нам придется разрабатывать собственные критерии.

Проще всего выбрать доверенное лицо и поинтересоваться. Наставник или близкий коллега видит тебя со стороны и может дать более объективную оценку твоего положения. Ты можешь обсудить свои способности как разработчика программного обеспечения,

руководителя проекта, собеседника, члена группы — словом, рассмотреть любую грань из того набора, который делает тебя тем, кто ты есть. В *General Electric* практикуется метод 360-градусной оценки, формализующий данную идею и заставляющий работников интересоваться своими показателями у коллег, руководителей и внутренних заказчиков. Несмотря на двусмысленность его названия, это замечательный метод, позволяющий оценить свои рабочие качества с разных сторон.

Проходя через подобную процедуру (самостоятельно или с чьей-либо помощью), ты узнаешь крайне важную информацию — в каких областях ты не разбираешься. Ты вовсе *не обязан* немедленно устранять все пробелы. Достаточно помнить об их существовании. Без четкого проговаривания этих аспектов ты будешь закрывать глаза на свои недостатки. А именно в этом случае можно легко попасть в неблагоприятную ситуацию, и это станет для тебя полнейшим сюрпризом. Если неприятность должна случиться, лучше заранее знать о ее приближении.

Разработчик, оцени самого себя!

Впрочем, даже при наличии волшебной *шкалы всесторонней самооценки* от нее не было бы никакой выгоды, если бы ты ей не пользовался. Запланируй контрольные мероприятия. Ты не отразишься в зеркале, если не назначишь для этого конкретное время. Фраза «не забудь спросить, что о тебе думают» не обладает достаточным посылом. Используй программу-календарь с напоминалкой и назначай самому себе встречу для самооценки. Первым делом определи измерительную систему, а затем вставь ее в расписание. Если подобная процедура не интегрирована в твою рабочую жизнь, ты просто не будешь ее выполнять.

Если в фирме, где ты работаешь, подобные вещи уже практикуются, не отбрасывай их как странную причуду отдела кадров. Воспринимай их серьезно и *извлекай* выгоду. Они могут быть некорректно реализованы, но в их основе лежит *правильная* мотивация.

Наконец, выработав систему оценок и определив расписание, приступай *к фиксации результатов*. Держи записи со своими оценка-

ми под рукой. Часто просматривай их и анализируй. Связав процесс самооценки с физическим артефактом, ты сделаешь его более основательным. Факт того, что ты устарел, не должен стать для тебя таким же неожиданным, как обнаружение ставших слишком тесными штанов.

### Действуй!

1. Воспользуйся методом 360-градусной оценки:

   - Составь список заслуживающих доверие лиц, мнением которых ты можешь поинтересоваться, не испытывая дискомфорта. В список должны попасть представители разных групп: коллеги, заказчики, руководители (и подчиненные, если они у тебя есть).
   - Перечисли десять характеристик, которые являются мерами твоего профессионализма (с твоей точки зрения).
   - Преврати список в опросный лист. Попроси участников дать тебе оценку в терминах каждой из характеристик. Включи туда вопрос «О чем еще мне следовало спросить?»
   - Раздай опросные листы людям, список которых ты составил на первом этапе. Попроси дать честные отзывы, тебе не нужна приукрашенная действительность.

   Внимательно изучи результаты и составь список действий, которые ты собираешься предпринять. Если ты задал корректные вопросы правильным людям, то получишь итог, подразумевающий конкретные меры. Поделись результатами опроса с теми, кто принимал в нем участие, — не ответами, а изменениями, которые ты запланировал. Не забудь поблагодарить всех, кто тебе помогал.

   Время от времени повторяй эту процедуру.

2. Начни делать записи. Это может быть как блог, о котором мы говорили в совете № 39, так и личный дневник. Записывай, над чем ты работаешь, что ты изучаешь, что думаешь об отрасли в целом.

   Через некоторое время перечитай старые записи. Ты по-прежнему придерживаешься такого мнения? Или, может быть, теперь это выглядит наивно? Насколько ты изменился?

## Совет 50
## Ловушка для обезьян

Роберт Пирсиг в книге «Дзен и искусство ухода за мотоциклом» (*Zen and the Art of Motorcycle Maintenance: An Inquiry into Values*) рассказывает поучительную историю о том, как в Южной Индии ловят обезьян. Не знаю, правда ли это, но из этой истории можно извлечь полезный урок.

В Южной Индии, где обезьяны много лет докучали людям, был изобретен оригинальный способ их ловли. В земле вырывалась глубокая и достаточно узкая ямка, с небольшим расширением на дне и узким входом. И в эту дыру кидался рис.

Обезьяны любят поесть. Собственно, именно это качество превращает их во вредителей. Они запрыгивают в машины или бросаются в толпу людей, чтобы вырвать еду прямо из твоих рук. В Южной Индии каждый про это знает. (Поверь, когда ты спокойно стоишь в парке, а на тебя внезапно нападает макака, чтобы что-то у тебя отобрать, это крайне нервирует.)

По словам Пирсига, обезьяны подходили к таким дырам, обнаруживали там рис и тут же совали туда лапы, жадно набирая как можно больше риса и сжимая его в кулаках. В более широкой нижней части это легко можно было сделать, но узкое отверстие мешало вытащить сжатый кулак наружу. Обезьяна застревала.

Разумеется, ничто не мешало ей просто бросить рис и отправиться восвояси.

Но обезьяны ценят еду настолько, что не могут заставить себя ее выбросить. Они будут цепляться за этот рис, пока не вытащат его наружу или не умрут. Смерть, как правило, наступает раньше.

Пирсиг называет это *ригидностью ценностей*. Данное состояние наступает, когда ты настолько уверен в ценности какой-либо вещи, что не в состоянии подвергнуть ее сомнению. Обезьяны настолько ценили рис, что даже перед необходимостью выбора между рисом, пленом и смертью не могли понять, что в этот момент правильнее всего будет его выбросить. Обезьяны в этой истории выглядят до

крайности глупо, но у большинства из нас есть собственные эквиваленты рису.

Если тебя спросят, хорошо ли помогать голодающим детям в развивающихся странах, ты, скорее всего, не задумываясь, ответишь утвердительно. Если кто-то попытается поспорить с тобой по этому вопросу, ты решишь, что он сумасшедший. *Это* и есть пример ригидности ценностей. Ты веришь в необходимость помощи детям так сильно, что не можешь представить, как в это можно *не* верить. Разумеется, не все ценности, в которые мы столь твердо верим, являются плохими. Для большинства людей вера в Бога (или атеизм) также представляет собой набор непоколебимых личных убеждений и ценностей.

Но далеко не все ценности, в которые мы верим, хорошие. Бывает и такое, что вещи, допустимые в одних обстоятельствах, оказываются совершенно недопустимыми в других.

К примеру, легко зациклиться на выбранной технологии. Особенно часто это происходит в случаях не самых популярных технологий. Но тебе настолько нравится какая-нибудь из них, ты придаешь такое большое значение ее защите, что начинаешь рассматривать любую такую возможность как битву, в которую стоит ввязаться, даже если очевидно, что защищаешь не то. Например, я сталкивался (в чем, наверное, был сам виноват) с чрезмерно рьяной группой поддержки операционной системы Linux. Ее пользователи были готовы установить эту систему на компьютер любого администратора, секретаря и вице-президента корпорации, не обращая внимания на тот факт, что с практической точки зрения инструментарий Linux не выдерживал сравнения с инструментарием коммерческих операционных систем. Ты выглядишь глупо и вызываешь у заказчиков недовольство, навязывая хорошее программное обеспечение совсем не тому, кому следовало бы.

Изменение своего веса сложно заметить, так как мы каждый день видим себя в зеркале. Аналогично обстоят дела с ригидностью ценностей. Так как наша карьера строится постепенно, изо дня в день двигаясь в том или другом направлении, легко стать жертвой собственной косности. Ты знаешь, что некий метод работает, и продолжаешь им пользоваться. Или, к примеру, ты всегда мечтал подняться

до руководителя и настойчиво стремишься к этой цели, не принимая во внимание того факта, что *просто программировать* нравится тебе куда больше.

Но может случиться и так, что выбранная тобой технология устареет, внезапно оставив тебя без твердой опоры. Подобно лягушке в медленно нагревающемся горшке с водой, ты вдруг можешь обнаружить себя в кипятке. В середине 1990-х, когда дело касалось служб печати и файлов в масштабе предприятия, многие из нас готовы были молиться на платформу NetWare от *Novell*. *Novell* предвосхитила время со своей службой каталогов, и тот из нас, кто «был в курсе», доходил в критике конкурирующих технологий до откровенных дерзостей. Продукция *Novell* занимала бóльшую часть рынка, и было сложно представить, что ситуация может измениться.

Ригидные ценности делают тебя уязвимым.

Ничто не заставляло думать, что *Novell* проигрывает *Microsoft*. Компания *Microsoft* не выпустила волшебную реализацию Active Directory, которая заставила бы нас завопить: «Вот это да! К черту NetWare!» Но передовые технологии от NetWare постепенно стали устаревать. Под большинством NetWare-администраторов вода закипела еще до того, как они поняли, что котел нагревается.

Помни о ловушке для обезьян как при выборе направления будущей карьеры, так и при выступлениях в защиту технологий. Ведь целенаправленный выбор может оказаться последней горстью риса, которую ты схватишь перед тем, как твоя карьера полетит к чертям.

### Действуй!

1. *Найди ловушку для обезьян*. Назови свои жесткие критерии. Какие ценности, о которых ты даже не задумываешься, управляют твоими ежедневными действиями?

   Нарисуй таблицу с двумя столбцами: «карьера» и «технология». В каждом столбце перечисли ценности, которые ты считаешь непоколебимо верными. К примеру, в столбце «карьера» укажи характеристики, которые *всегда* считал своими сильными сторонами. Перечисли и то, что считаешь своими

слабостями. А как насчет карьерных *устремлений*? («Я хочу стать генеральным директором!») Укажи правильные пути достижения цели.

В столбце «технология» перечисли, что ты считаешь самым ценным в технологиях, в которые решил вложиться. Какие наиболее важные атрибуты технологии следует учитывать в процессе выбора? Каким образом ты создаешь масштабируемые системы? Какой видишь наиболее продуктивную среду для разработки программного обеспечения? Какие платформы лучше, а какие хуже всего в общем случае?

Завершив работу над списком, начни его перечитывать и для каждого пункта мысленно формируй противоположное утверждение. Ты допускаешь, что противоположный взгляд на вещи тоже может быть верным? Попробуй честно оценить каждую из записанных тобой характеристик.

Это список твоих уязвимостей.

2. *Знай своего врага*. Выбери технологию, которую ты больше всего ненавидишь, и сделай на ее основе один проект. Разработчики имеют склонность разбиваться на конкурирующие лагери. Приверженцы .NET ненавидят фанатов J2EE, а те, в свою очередь, платят им той же монетой. Любители операционной системы UNIX ненавидят Windows, а любители Windows терпеть не могут UNIX.

   Придумай простой проект и попробуй создать *прекрасное* приложение в рамках ненавидимой тобой технологии. Если ты пишешь на языке Java, покажи этим фанатам .NET, как настоящий разработчик использует их платформу! В лучшем случае ты поймешь, что столь ненавидимая тобой технология на самом деле не так уж плоха и на ее основе можно создавать приличный код. Кроме того, у тебя появится (разумеется, пока не развитый) новый навык, из которого в будущем ты, возможно, сумеешь извлечь выгоду. В худшем случае ты выполнишь практическое упражнение, получив дополнительную базу для своих аргументов.

## Совет 51
## Избегай каскадного планирования карьеры

В начале этого тысячелетия один небольшой протест сильно изменил всю индустрию разработки программного обеспечения. Группа экс-

пертов-разработчиков в какой-то момент поняла, что существующие методы ведения проектов могут привести как к успеху, так и к неудаче. В промышленных условиях, где большинство проектов оканчивается неудачей, они, как им казалось, нашли способ, ведущий к успеху. Эта группа называла себя альянсом специалистов по быстрой разработке ПО (Agile Alliance).

В индустрии в те времена считалось, что единственным способом разработки проектов является спуск сверху вниз по цепочке серьезно распланированных, жестко регламентированных процессов. Аналитики формировали список требований, передавали документацию архитекторам, которые создавали архитектуру и передавали ее дизайнерам, которые, в свою очередь, работали над детализированным дизайном. Затем все это отдавалось разработчикам, перед которыми стояла задача написать код дизайна на каком-нибудь языке программирования. Наконец, спустя месяцы, а иногда и годы усилий код собирался и отдавался группе тестировщиков, которые утверждали его развертывание.

Иногда вариации на тему такого процесса давали неплохие результаты. Когда в начале проекта все в деталях знают, что им предстоит делать, столь строгий подход к планированию позволяет получить хорошо продуманное и качественное программное обеспечение. Но в большинстве случаев исполнители не знали всех деталей реализации будущего проекта. Чем больше и сложнее проект, тем менее вероятно, что кто-то сможет вообразить все его особенности в мельчайших подробностях и написать спецификацию. Такой процесс называют *каскадным*, и в наши дни это слово почти всегда воспринимается как синоним плохого процесса.

В итоге члены альянса специалистов по быстрой разработке поняли, что следование жестко регламентированному процессу хотя и порождает хорошо протестированное и тщательно документированное программное обеспечение, но совсем не отвечает пожеланиям пользователей. Их бунт выразился в создании гибких методик. Это были процессы разработки приложений, в которые легко вносились изменения. На планирование и дизайн отводилось меньше времени. Если программы гибкие, значит, их модификация *может* обходиться дешево. Гибкие методики рассматривают изменения как неотъемле-

мую часть процесса разработки и упрощаются, чтобы сделать этот процесс по возможности дешевым и простым.

Сейчас все это звучит тривиально. Но изначально гибкие процессы стали источником конфликтов и дискуссий. В теории идея детального планирования и жесткости выглядела без сомнений верной. Просто на практике она работала далеко не всегда.

На ранних стадиях моего перехода к гибким подходам (особенно это касается экстремального программирования) я начал рассматривать все через призму гибкой разработки. Появлялись силы и мотивации, выходящие за рамки написания программ. Как только я сталкивался со сложной проблемой, то понимал, что постепенный подход к решению, допускающий изменения, в моем случае всегда является менее сложным и более эффективным.

Но почему-то на осознание того факта, что самым сложным проектом, которым я когда-либо занимался, — самый напряженным и важным — была моя карьера, потребовалось немало времени. Я запланировал свою карьеру сверху вниз, как проект программы, созданный в рамках каскадного процесса. В итоге со мной и с моей карьерой начинали происходить те же вещи, которые возникают в программных проектах.

Я успешно продвигался к должности вице-президента корпорации и директора по информационным технологиям. От зеленого программиста я быстро шагнул к архитектору программного обеспечения, руководителю и директору и мог легко представить, чем завершится моя цепочка. Но при всех моих успехах я начал ощущать, что занимаюсь не тем, что мне нравится. Более того, чем успешнее я становился, тем меньше получал удовольствия от работы.

Я делал для себя ровно то же самое, что строго регламентированные процессы делают для заказчиков. Я *отлично* справлялся с созданием для себя карьеры, которая была *мне не нужна*.

Сначала я не мог осознать, что подобная проблема решается просто. Достаточно *поменять* свою карьеру. Это может означать что угодно. Для меня это было возвращение к серьезным технологиям, которые изначально так восхищали меня в этой отрасли. Для моих знакомых это был переход от системного администрирования к разработке про-

граммного обеспечения, от работы, не связанной с компьютерами, — к программированию или вообще уход из профессии и посвящение себя любимому делу.

Как и в ситуации с разработкой программного обеспечения, стоимость изменений не должна быть слишком высокой. Разумеется, превратиться из тестировщика в адвоката вряд ли будет легко. Но в трансформации из управленца в программиста и обратно нет ничего сложного. Несложно найти и новую фирму. Или переехать в другой город.

Карьерные изменения — это не строительство небоскреба, они не потребуют выкинуть на помойку весь наработанный ранее опыт. В настоящий момент я целыми днями пишу программы на Ruby, но мой опыт руководителя, занимавшегося переводом операций в офшоры, остается актуальным и помогает в моих трудах. Работодатели и клиенты это понимают и используют данное преимущество.

Важно понимать, что подобные изменения не только возможны, но и *необходимы*. Как разработчик ты никогда не позволишь вовлечь себя в ненужный заказчику проект. Гибкие методики просто не дадут тебе этого сделать. О твоей карьере можно сказать то же самое. Ставь большие цели, но по пути постоянно вноси коррективы. Учись на собственном опыте и меняй цели при необходимости. В конечном счете нам всем нужен довольный заказчик (а в деле планирования карьеры ты сам являешься собственным заказчиком), а не выполнение требований.

## Совет 52
## Лучше, чем вчера

Исправить ошибку легко (в общем случае). Что-то идет не так? Тебе об этом кто-то сообщил? Если ты можешь воспроизвести ошибку, то для исправления достаточно устранить вызывающие ее факторы и удостовериться, что проблем больше не возникнет. Если бы все трудности исчезали так просто!

Однако подобная ясность сопровождает далеко не каждую проблему или трудную ситуацию. Большей частью серьезные проблемы вы-

глядят как огромный запутанный клубок потенциальных неисправностей. Это утверждение применимо к разработке программного обеспечения, управлению карьерным ростом и даже образу жизни и здоровью.

Сложная система с кучей ошибок нуждается в переделке. Твоя карьера тормозится с каждой минутой. Твой сидячий образ жизни, намертво связанный с компьютером, неуклонно превращает твое тело в кашу. Все эти проблемы намного серьезнее и сложнее, чтобы их можно было *просто устранить* как обычную ошибку. Все они запутанны, сложно измеримы и состоят из множества мелких решений, часть из которых работать не будет!

Трудность решения сложных задач легко лишает нас мотивации и заставляет обратить внимание на те проблемы, которые проще анализировать и быстрее решать. А сложное дело мы откладываем на потом. Это откладывание вызывает у нас чувство вины, которое нас расстраивает, что, в свою очередь, приводит к еще большим проволочкам.

В теме 49 я признался, что всю свою жизнь боролся с лишним весом. По себе знаю, что когда твоя форма далека от идеальной, призыв «просто приди в форму» сложно даже принять, а не то что ему последовать. И чтобы еще больше все усложнить, попытки предпринять какие-то усилия, направленные на улучшение своей формы, не дадут результатов ни сразу, ни даже через неделю. Ты можешь *целый день* работать над тем, чтобы прийти в форму, а через неделю не обнаружить ни одного следствия потраченных усилий.

Подобные демотиваторы лишают тебя воли к борьбе еще до ее начала.

С недавнего времени я вплотную занялся своей проблемой. Почти каждый день посещаю спортивный зал, стал следить за своим питанием, словом, делаю все, что только можно. Но даже при такой серьезной программе увидеть результаты достаточно сложно. Однажды вечером, когда меня опять захлестнуло ощущение отсутствия мотивации, мой друг Эрик Кастнер опубликовал в социальной сети Twitter следующее сообщение:

> Помоги привести в форму мою $%!^... спрашивай меня каждый день: «Сегодня было лучше, чем вчера?» (питание/упражнения) — сегодня: ДА!

Прочитав это сообщение, я понял, что нашел ключ к проблеме лишнего веса. Впрочем, я уже знал эту методику, ведь в жизни мне пришлось *успешно* решить немало серьезных проблем. Секрет в попытках сделать так, чтобы задача, над которой ты работаешь, *сегодня стала лучше, чем вчера*. Вот и все. Это легко. И как утверждает Эрик, энтузиазм легко сохранить, имея дело с реальными, осязаемыми шагами по направлению к далекой цели.

Недавно мне довелось поработать с одним из наиболее сложных и безобразных Ruby-приложений, когда-либо попадавшихся мне на глаза. Моя компания унаследовала этот консалтинговый проект от другого разработчика. Нужно было внедрить несколько ключевых подсистем и избавиться от огромного количества ошибок и проблем с производительностью. При попытке внести эти изменения мы обнаружили, что код представляет собой жуткую кашу. Нанявшая нас фирма была ограничена во времени и финансах, поэтому такой роскоши, как возможность написать все заново «с нуля», мы были лишены, хотя этот код проще всего было выкинуть в мусорную корзину.

Мы медленно двигались вперед, внося одно небольшое изменение за другим, причем каждое из них занимало намного больше времени, чем ожидалось. Казалось, что это чудовищное нагромождение кода никогда не закончится. Работа над приложением была утомительной и безрадостной. Но со временем исправления начали вноситься быстрее, а некогда неприемлемая производительность повысилась. И все это благодаря простому стремлению каждый день делать код лучше, чем он был вчера.

Порой это было разбиение огромного метода на более мелкие методы, но со значимыми именами. Иногда это сводилось к удалению иерархий наследования, не принадлежавших к используемой объектной модели. Временами мы всего лишь исправляли неработающий модульный тест. Но так как все эти изменения вносились постепенно, они происходили достаточно безболезненно. Вполне реально переделать один метод за время, обычно уходящее на приготовление чашки кофе или обсуждение последних новостей с коллегами. Небольшое улучшение мотивирует. Ты видишь эффект *в том фрагменте*, над которым ты работаешь, сразу же после внесения изменений.

Впрочем, даже постепенные изменения не позволяют увидеть эффект в *целом* проекте. Попытки добиться большего уважения среди коллег или улучшить состояние собственного здоровья, несмотря на отдельные каждодневные улучшения, не приводят к непосредственно осязаемым результатам. Именно поэтому, как мы уже видели ранее, такие большие цели лишают нас мотивации. А значит, в случае больших, сложно достижимых целей, к которым ты стремишься, важно думать не о том, как ты каждый день немного продвигаешься вперед, а о том, как ты каждый день *улучшаешь* качество работы. К примеру, я не могу гарантировать, что сегодня буду весить меньше, чем вчера, но *в состоянии* контролировать количество усилий, направленное на достижение поставленной цели. И сделав сегодня больше, чем вчера, я с полным правом могу гордиться своим достижением. Такое последовательное измеримое улучшение *качества работы* выводит меня из цикла вины и постоянных откладываний, который, в конечном счете, мешает большинству из нас в достижении Больших Важных Целей.

Ты должен радоваться даже *небольшим* «лучше». Написать сегодня на один тест больше, чем вчера, достаточно, чтобы оказаться ближе к цели «лучше разобраться в модульном тестировании». Если ты начинаешь «с нуля», один дополнительный тест в день будет вполне устойчивым прогрессом. И к моменту, когда ты уже не сможешь показывать лучшие, чем вчера, результаты, окажется, что ты уже «разобрался в модульном тестировании» на достаточном уровне и увеличивать количество тестов дальше просто не нужно. При этом если начиная «с нуля», ты в рамках собственных планов усовершенствования хочешь в первый же день написать пятьдесят тестов, то в этот день ты, скорее всего, тяжело поработаешь, а вот второго дня уже не случится. Поэтому улучшения должны быть небольшими и постепенными, но *ежедневными*. Кроме всего прочего, небольшие улучшения снижают цену неудачи. Пропущенный день всего лишь сдвигает точку отсчета на завтра.

Эта простая аксиома хороша своей применимостью как к тактическим целям, таким как завершение проекта или доработка фрагмента кода, так и к возможным высокоуровневым целям. Насколько больше усилий к построению своей карьеры ты приложил сегодня? Завяжи новое знакомство, опубликуй исправление для проекта с открытым

исходным кодом, напиши вдумчивый пост в своем блоге. Увеличь на единицу по сравнению со вчерашним числом количество пользователей, которым ты сегодня помог на техническом форуме, связанном с твоей областью знаний. Каждый день работая над самоусовершенствованием чуть лучше, чем вчера, ты почувствуешь, как цель сделать невероятную карьеру, изначально напоминающая безбрежный океан, становится более осязаемой.

**Действуй!**

1. Составь список сложных или труднодостижимых целей; они могут быть как личными, так и профессиональными. Если список получится длинным, ничего страшного. Подумай над каждым пунктом, что ты можешь сделать, чтобы уже сегодня стать лучше, чем вчера. На следующий день снова перечитай список. Был ли вчерашний день лучше, чем позавчерашний? Как сегодня еще заметнее улучшить свои показатели? Проделывай это каждый день. Добавь напоминание в календарь. Каждое утро трать по паре минут на обдумывание сегодняшних свершений.

## Совет 53
## Стань независимым

В стрессовых ситуациях я часто с нежностью вспоминаю время работы в большой корпорации. Как мой собственный офис, так и огромная иерархия руководителей позволяли мне чувствовать себя комфортно. Тогда мы шутили, что в большой компании умный человек может устроиться так, чтобы *почти ничего* не делать. В большинстве случаев вина за провал проекта ложилась на такое количество людей, распределенных по такому количеству уровней корпоративной иерархии, что было сложно понять, где же конкретно произошел сбой. Если выполнение каких-то вещей затягивалось, сложность организации сводила причины этого к тому, что никто по большому счету был не в курсе точных сроков сдачи проекта.

В дни, когда ты не чувствуешь особого желания заниматься текущими делами, работа в крупной компании дает тебе возможность сесть

и, к примеру, немножко побродить по интернету. Или пораньше уйти домой. Или взять отгул. При всех моих претензиях к крупным компаниям в работе там есть свои преимущества.

Проблема в том, что предоставляемый корпорацией уровень защиты замедляет твой рост. Возможность спрятаться за щитом посредственностей, наполняющих большинство подразделений, лишает тебя стимула преуспеть. Даже самых благонамеренных из нас манит спокойный оазис сайта YouTube или любимая коллекция комиксов[1].

В этом смысле крупная компания является прекрасным местом, позволяющим ощутить себя наполовину неработающим, если ты в какой-то момент выдохся. Но если ты хочешь быть не как все (а ты этого хочешь!), корпорация подходит тебе в такой же степени, как кондитерская человеку, мечтающему избавиться от жировых отложений. Где же выход? Стань независимым!

У тебя есть набор навыков. Ты их отточил. Ты знаешь, сколько ты стоишь. Осталось пройти последний тест — стать независимым работником. В этом случае у тебя нет бюрократического аппарата, за которым можно спрятаться. Ты напрямую подконтролен людям, которые тебе платят. Представление о предоставляемой тобой услуге возникает по результатам твоей работы. Группы, с которой можно поделить ответственность, если что-то пойдет не так, уже нет. Есть только ты, твой опыт и твои навыки исполнителя.

Превращение во фрилансера заставляет учиться продавать себя, одновременно проверяя правильность выбранных тобой областей бизнеса и технологий. Став независимым, уже нельзя полагаться на то, что заказчики сами тебя найдут, как это происходит при работе в крупной компании. Заказчиков приходится *искать* самостоятельно. После чего их еще нужно убедить в том, что тебе имеет смысл платить.

Еще следует понять, сколько ты стоишь. Твоя работа стоит $50 в час? Или $250? Как ты собираешься оплачивать счета? Чем подтверждаются те деньги, которые тебе, с твоей точки зрения, должны платить? *Ты на самом деле стоишь столько, сколько ты думаешь?*

---

[1] Если ты ищешь нечто подобное, рекомендую сайт http://toothpastefordinner.com. Я провел там немало веселых часов.

## ЛЮБОПЫТСТВО — ЭТО СИЛА

*Майк Кларк, независимый консультант/программист*

Мои родители утверждают, что в детстве я был крайне любознательным. Задавал множество вопросов, читал все, что попадалось под руку, а принцип работы механизмов изучал, разбирая их на части. Как оказалось, это были не просто слова — я так никогда и не перерос своей ненасытной любознательности. На это часто не обращают внимания, но я считаю, что любопытство — это сила. И для его развития порой требуется всего лишь небольшая практика.

Оглядываясь назад, я могу выделить несколько повлиявших на мою карьеру событий, которые произошли в основном потому, что мной двигало любопытство. Я приведу несколько примеров в надежде, что они заставят вас прислушаться к голосу своего любопытства:

Никогда не думал, что стану программистом. Меня восхищали самолеты и космические корабли, и логичным выбором стало поступление на авиа- и ракетостроение в Университет аэронавтики Эмби-Риддла. Но через год напряженной работы я обнаружил, что студенты-компьютерщики живут куда веселее нас. Новая стипендиальная программа включала в себя применение компьютерного моделирования к проблемам авиации. Я интересовался компьютерами в последних классах школы, но никогда не рассматривал программирование как вариант будущей работы. Итак, я начал знакомиться с компьютерными фанатиками, чтобы понять, чем они занимаются. Через некоторое время я перешел на другую программу обучения. В итоге это небольшое изменение оказалось одним из лучших решений. Учеба по-прежнему была сложной, но я наслаждался каждой минутой. Моя изначальная любознательность, направленная на программирование, быстро превратилась в страсть, которая заставила меня подать заявку на стажировку в NASA и стала первым шагом к карьере в области информационных технологий. Я по сей день не могу переоценить потенциальную награду за поиск ответа на вопрос, над чем же коллеги-технари работают для собственного удовольствия.

Как только я начинал чувствовать себя удобно, это было сигналом, что пришло время попробовать что-то новое. За много лет написания встроенных программ для аэрокосмической промышленности я полностью освоился (что для меня связано со скукой) с языками C и C++. Примерно в это время мое любопытство заставило меня обратить внимание на веб-программирование, так как оно радикальным образом отличалось от программирования встроенных систем. К сожалению, на рабочем месте у меня не было доступа в интернет (это был один из тех самых, сверхсекретных проектов), поэтому писать программы для Сети

**ЛЮБОПЫТСТВО — ЭТО СИЛА (продолжение)**

я учился вечерами и по выходным. В итоге эти дополнительные усилия дали мне возможность поработать над новым проектом, в котором требовался язык Java. Мне довелось написать множество сетевых Java-приложений для множества проектов... и работодателей. Интерес к веб-разработке послужил катализатором к разностороннему развитию моих способностей, что в конечном счете позволило хорошо продвинуться по карьерной лестнице.

По собственной прихоти я начал изучать Ruby on Rails. Ruby — это фантастический язык, заставивший меня по-другому взглянуть на программирование. Среда Rails аналогичным образом изменила мой взгляд на сетевые приложения. В те времена у меня не было заказчиков, готовых платить за программирование на языке Ruby или работу в среде Rails, но это не имело никакого значения. Мне было любопытно, я просто ничего не мог с собой поделать. Я стал меньше работать за деньги, тратя это время на освоение тонкостей Ruby и Rails. Тогда мне и в голову не приходило, что в начале 2005 года у меня появится возможность создать одно из первых коммерческих Rails-приложений, а Дэйв Томас пригласит меня помочь в написании книги по Rails. Мой интерес к пока новой технологии начал еще один успешный виток моей карьеры.

Мое любопытство направлено не только на технологии; аналогичный интерес у меня вызывают различные области бизнеса. Это привело сначала к самостоятельной работе независимого консультанта, затем к открытию образовательной компании (*The Pragmatic Studio*). Интерес к аспектам ведения малого бизнеса дал мне возможность приобрести множество новых навыков: продажа, маркетинг, поддержка пользователей и т. п. То, что я увидел картину в целом, помогло мне улучшить свои навыки программирования.

А что вызывает ваше любопытство? Попробуйте некоторое время заниматься тем, что вам интересно, и посмотрите, что получится. Вы можете удивиться, обнаружив, к чему все это приведет!

Стать независимым сложно. Это проверка на прочность всех твоих профессиональных качеств. *Возможно*, ты к этому еще не готов. Впрочем, никто не заставляет делать радикальный шаг и сразу идти до конца. Рассматривай это как проект саморазвития и попробуй продать на рынке труда свое свободное время. Поставь себе цель заключить договор на конкретный срок и постарайся полностью удовлетворить желания клиента. Работай по вечерам или в выходные (но только не

в своем кабинете в рабочее время!). Ты многому научишься, не теряя своей подушки безопасности. В худшем случае ты несколько недель будешь перерабатывать, не справишься с проектом и вернешься в свой уютный офис с новым пониманием своей роли. В лучшем — ты достигнешь больших успехов, полюбишь то, что делаешь, и найдешь новый путь реализации карьерных устремлений и получения финансового вознаграждения.

Обозреватель Сэмми Лабри предлагает альтернативный способ достижения независимости. Если ты работаешь на крупную компанию, подумай о переходе в небольшую фирму. Если ты сотрудник компании, занимающей устойчивое положение на рынке, попробуй свои силы в недавно созданном предприятии. Небольшая недавно созданная фирма даст тебе лучшее из обоих вариантов: работу на полный рабочий день со стабильной зарплатой *и* возможность непосредственно столкнуться со всеми проблемами данного бизнеса.

# РАДУЙСЯ ЖИЗНИ!

> А я говорю вам: когда вы трудитесь, вы исполняете часть самой ранней мечты земли, уготованную вам в те времена, когда эта мечта родилась. И работая, вы истинно любите жизнь. А возлюбить жизнь через работу — значит приблизиться к глубочайшей тайне жизни.
>
> *Джебран Халиль Джебран, пророк*

Если как разработчик программного обеспечения ты дошел до этапа, когда можешь позволить выбирать, в каком направлении строить свою карьеру, прими мои поздравления! Можешь считать себя счастливчиком. Во многих культурах возможность решать, чем зарабатывать на жизнь, — привилегия, доступная единицам. Разработчики программного обеспечения, как правило, могут не волноваться по поводу отсутствия крыши над головой или денег на еду.

Ты мог выбрать любую карьеру, но выбрал этот путь, который еще и крайне увлекателен. Он связан с творчеством. Он требует глубоких размышлений и вознаграждает тебя ощущением, что ты умеешь нечто, чего большинство не может даже представить. Конечно, ты можешь беспокоиться о том, как перейти на следующий уровень, какое впечатление ты производишь или как заслужить уважение сотрудников своей фирмы или коллег по отрасли, но если ты минутку как следует поразмышляешь на эту тему, то поймешь, что живешь очень даже прилично.

Разработка программного обеспечения является делом *одновременно* и сложным, и благодарным. Это творческий акт, своего рода искусство, но его результатом (в отличие от искусства) становятся вполне конкретные измеримые ценности.

Разработка программного обеспечения — это счастье!

В конечном счете самое главное, что я усвоил за время своего путешествия по карьерной лестнице, — это ощущение, что важно не то, чем ты зарабатываешь на жизнь или чем ты *обладаешь*. Самое главное — как ты принимаешь все эти вещи. Твое внутреннее ощущение. К удовольствию от работы, так же как и к выбору карьеры, следует *целенаправленно стремиться* и *выбирать* те пути, которые могут нам его обеспечить.

# СПИСОК ЛИТЕРАТУРЫ


1. Kent Beck. Extreme Programming Explained: Embrace Change. Addison-Wesley Longman, Reading, MA, 2000.
2. Douglas Coupland. Microserfs. Regan Books, New York, NY, USA, 1996.
3. Tom Demarco and Timothy Lister. Peopleware: Productive Projects and Teams. Dorset House, New York, NY, USA, Second, 1999.
4. Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, MA, 1995[1].
5. Seth Godin. Purple Cow: Transform Your Business by Being Remarkable. Portfolio Hardcover, USA, 2003.
6. Andrew Hunt and David Thomas. The Pragmatic Programmer: From Journeyman to Master. Addison-Wesley, Reading, MA, 2000.
7. Gary Hamel. Leading the Revolution: How to Thrive in Turbulent Times by Making Innovation a Way of Life. Plume, New York, NY, USA, 2002.
8. Thich Nhat Hanh. The Miracle of Mindfulness. Beacon Press, Boston, MA, 1999.
9. Robert M. Pirsig. Zen and the Art of Motorcycle Maintenance: An Inquiry into Values. Perennial Classics, New York, NY, USA, Reprint Edition, 2000.
10. Steven A. Silbiger. The Ten-Day MBA: A Step-By-step Guide To Mastering The Skills Taught In America's Top Business Schools. Harper Paperbacks, New York, NY, USA, 1999.


[1] *Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж.* Приемы объектно-ориентированного проектирования, СПб.: Питер, 2013.