БИБЛИОТЕКА ПРОГРАММИСТА

Чед Фаулер

Программист-ФАНАТИК

- Как сделать карьеру, занимаясь тем, что тебе нравится
- Самосовершенствование и личная ответственность
- Автоматизация работы и формирование команды
- Рынок ПО: инвестиции, продвижение, продажи

Pragmatic Bookshelf

ПИТЕР®

Библиотека программиста (Питер)

Чед Фаулер

Программист-фанатик

«Питер»
2009

Фаулер Ч.

Программист-фанатик / Ч. Фаулер — «Питер», 2009 — (Библиотека программиста (Питер))

ISBN 978-5-496-01062-7

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 многих других. Подобные знания обычно приходят лишь в результате многолетнего опыта реальной работы. По большому счёту 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. В формате ios.epub представлен издательский файл.

ISBN 978-5-496-01062-7

© Фаулер Ч., 2009
© Питер, 2009

Содержание

Чед Фаулер
Программист-фанатик

© Pragmatic Bookshelf; 2 edition (June 4, 2009)
© Перевод на русский язык ООО Издательство «Питер», 2015
© Издание на русском языке, оформление ООО Издательство «Питер», 2015

* * *

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 видение 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, 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.

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

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 крайней 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.

> Доходным может оказаться любой из концов кривой восприятия технологий

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 рискованно, 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 выяснилось, 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 областей, 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.

Используй дисбаланс рынка труда.

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 альтернативы 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 таковые. 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[2], 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!*

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

Конец ознакомительного фрагмента.

Текст предоставлен ООО «ЛитРес».

Прочитайте эту книгу целиком, купив полную легальную версию на ЛитРес.

Безопасно оплатить книгу можно банковской картой Visa, MasterCard, Maestro, со счета мобильного телефона, с платежного терминала, в салоне МТС или Связной, через PayPal, WebMoney, Яндекс.Деньги, QIWI Кошелек, бонусными картами или другим удобным Вам способом.