# Наш код

Ремесло, профессия, искусство

Егор Бугаенко

Это полуавтобиографическая история о software-архитекторе из Кремниевой долины, который пишет код, тестирует, отлаживает, релизит, спорит с коллегами, организует и управляет...

ПИТЕР®

# Code Ahead (Volume 1)

by Yegor Bugayenko

# Наш код
# Ремесло, профессия, искусство

Егор Бугаенко

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

2019

ББК 32.973.2-018
УДК 004.4
Б90

**Бугаенко Егор**

Б90 Наш код. Ремесло, профессия, искусство. — СПб.: Питер, 2019. — 224 с. — (Серия «Библиотека программиста»).

ISBN 978-5-4461-1144-2

Быть программистом может быть интересно и весело, но быть разработчиком программного обеспечения — это сущий ад. Компьютеры логичны, люди — нет. Увы, в современной индустрии программного обеспечения не платят за программирование. Платят за разработку программного обеспечения, а это подразумевает выполнение задач в команде — вместе с другими людьми. Команды состоят из своенравных людей, а не из классов и методов Java.

Успех программного проекта зависит от умных инженеров, которые зачастую ленивы, невежественны, эгоистичны, раздражительны и попросту несчастны. Успех зависит от людей, которые нередко не умеют общаться, делиться знаниями, руководить и подчиняться, а также следовать указаниям. Он зависит от нашей способности формировать команды и участвовать в их деятельности. А еще от наших социальных навыков — порой в гораздо большей степени, нежели от навыков технических.

Драма? Согласен. Эта драма касается каждого из наших собратьев по профессии, поэтому, если стремитесь выжить в такой профессии, читайте эту книгу.

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

ББК 32.973.2-018
УДК 004.4



ISBN 978-1982063740 англ.
ISBN 978-5-4461-1144-2

# Оглавление

# Предисловие

Разработка программного обеспечения сегодня подразумевает не только написание кода. Речь идет не об алгоритмах, формулах, байтах, файлах или протоколах. И не об инструкциях, операторах, модульных тестах, пользовательских интерфейсах или сценариях развертывания. И даже не о производительности, масштабируемости, отказоустойчивости или надежности. Все эти компоненты важны, но не они делают одного программиста более эффективным, чем другого, или некую команду разработчиков успешнее конкурентов. Решающую роль в современном мире программного и аппаратного обеспечения играет кое-что другое: это не компьютеры, а люди.

Успех программного проекта зависит от людей, которые зачастую ленивы, невежественны, эгоистичны, раздражены и попросту несчастны. Он зависит от людей, которые нередко не умеют общаться, делиться знаниями, руководить и подчиняться, а также следовать указаниям. Он зависит от нашей способности формировать команды и участвовать в их деятельности. А также от наших социальных навыков — порой в гораздо большей степени, нежели от навыков технических.

В этой книге вы повстречаетесь с небольшой группой программистов, архитекторов и руководителей, которым платят за создание программного обеспечения. На протяжении следующих более чем двух сотен страниц они будут решать проблемы, обходить конфликтные ситуации, обсуждать особенности командной работы. Я надеюсь, что их диалоги и монологи помогут вам понять важность социального аспекта разработки программного обеспечения и в результате стать еще лучшим разработчиком или руководителем.

# Как читать эту книгу

Во-первых, рассматривайте эту книгу как художественное произведение. В нем есть сюжет, главный герой, несколько персонажей и диалоги между ними. Я решил выбрать этот формат в первую очередь потому, что описанные в книге мысли и идеи родились во время работы над реальными программными проектами и наиболее эффективный способ запечатлеть их — написать историю.

Во-вторых, в книге много сносок и ссылок на другие источники — как научные, так и профессиональные. Я старался использовать подходящие цитаты из разных источников, чтобы проиллюстрировать историю, описанную в книге. Советую вам прочитать все сноски — это часть истории, а не дополнительный материал.

В-третьих, одни сноски обозначены значком ★ — это те материалы, которые я рекомендую прочитать полностью. Другие же отмечены значком 👎 — это то, с чем я совершенно не согласен и о чем упоминаю только для того, чтобы показать, насколько некоторые суждения могут быть ошибочными.

В-четвертых, каждый раздел содержит несколько семантических блоков, состоящих из нескольких абзацев. В каждом блоке есть заметка на полях, отражающая то, что я хотел донести до вас, рассказывая эту историю.

**Вы можете просто читать заметки на полях, подобные этой, и вам будет понятна общая суть книги.**

Наконец, обратите внимание: эта книга *не* для менеджеров, руководителей, тестировщиков или обычных людей, которые просто любят читать. Это техническая книга, содержащая специфические термины и понятия, которые вы можете понять только в том случае, если вы разработчик программного обеспечения. Эта книга для программистов, которые, как и я, зарабатывают на жизнь написанием кода.

# Глава 1

# Адриан

Быть программистом может быть интересно и весело, но быть разработчиком программного обеспечения — это сущий ад. Компьютеры логичны, люди — нет. Увы, в современной индустрии программного обеспечения не платят за программирование. Платят за разработку программного обеспечения, а это подразумевает выполнение задач в команде — вместе с другими людьми. Команды состоят из иррациональных людей, а не из классов и методов Java. Я собираюсь устроиться на работу в одну из таких команд и выжить в ней. Я могу писать на Java, но для успеха этого недостаточно. Там понадобятся другие навыки. И некоторые из них мне еще только предстоит развить.

# Добытчики и недобытчики

Я сказал секретарю:

— У меня встреча с Крисом, он меня ждет.

Когда-нибудь у меня будут и свой секретарь, и свой офис, и свои программисты, и свой стартап с инвесторами, и большая миссия, которой мы будем следовать. И я буду известным и успешным. А пока этого нет, мне нужно платить за жилье. Похоже, эти ребята достаточно богаты для того, чтобы иметь со мной дело в течение последующих нескольких месяцев, и, сдается, им понравилось мое резюме. Сейчас я должен убедить их в том, что мечтаю остаться с ними навсегда.

**Поскольку любая архитектура программного обеспечения является продуктом деятельности людей, для улучшения продукта нужно в первую очередь улучшить людей.**

— На самом деле это *она*, — сухо ответила секретарь.

Крис появилась через минуту. Мы идем в переговорку, она предлагает кофе, но я прошу стакан воды. Она выходит, и я на автомате достаю свой телефон и проверяю Facebook. Крис возвращается со стаканом воды и в сопровождении чувака в темной футболке с логотипом GitHub. Она говорит, что очень рада знакомству, и уходит. Чувака зовут Адриан, он разработчик. Мы начинаем разговор. Своими вопросами он производит впечатление человека достаточно опытного. Я чувствую себя довольно комфортно — очевидно, что в технологическом плане я превосхожу его, так что беспокоиться мне явно не о чем.

— Нам нужен кто-то, кто исправит нашу архитектуру, — резюмирует Адриан через полчаса.

«Скорее вам нужен кто-то, кто исправит вас до того, как вы сможете что-то сделать с архитектурой», — подумал я, но вслух сказал:

— Буду рад помочь. Мне ваш проект кажется интересным как в технологическом отношении, так с точки зрения бизнес-процессов. Я не очень люблю работать над скучными проектами, независимо от

того, сколько за это платят. Я предпочитаю быть увлеченным своим делом, поэтому хочу заняться чем-то интересным и современным.

Адриан меланхолично улыбается. Возможно, он слышал это от каждого второго человека, сидящего в этой комнате...

Он задает мне непростой вопрос:

— Когда вы будете готовы приступить? Сколько времени вам необходимо для завершения ваших текущих проектов? Мы ищем человека на полный рабочий день.

Похоже, он очень гордится сказанным. Видимо, они думают, что полный рабочий день будет для меня абсолютным благом. Я уже понял, что мне придется сидеть в этом офисе с девяти до пяти, отрабатывая свою зарплату. Пока у меня нет своего стартапа, видимо, так оно и будет. На этой неделе это уже третья компания, где я прохожу собеседование. Предыдущие две мне еще не ответили, хотя казалось, что я им подхожу. Этот чувак выглядит куда более отчаявшимся. Подозреваю, его серверы падают чуть ли не каждую вторую ночь, и это станет *моей* проблемой, как только я подпишу контракт. Мне нужно быть осторожным.

— Пожалуй, недели хватит для завершения всех дел.

Я говорю то, что должен сказать, хотя сейчас у меня нет ни проекта, ни работы, ни дохода. Я не могу сказать ему правду — необходимо соблюдать правила игры. Я должен выглядеть востребованным и очень занятым. А с другой стороны, разве обещание завершить все дела за неделю выглядит логичным? Что это за проект такой, над которым я сейчас работаю, что я его мог бы завершить всего-навсего за неделю? Само собой, мы оба понимаем, что это вранье... но понимаем также то, что не можем с этим ничего поделать.

> Человек, обещающий быть лояльным к одной компании, сразу перестает быть лояльным к другой.

— Как долго вы работаете в этой компании? — спрашиваю я его, чтобы сменить тему.

— Два года, — отвечает Адриан.

— Вы создали большинство проектов компании?

— Да, я здесь первый разработчик. Два года назад я познакомился с Тони, нашим сооснователем и техническим директором. Вы увидите его на следующем собеседовании.

Я вижу, с каким уважением Адриан о нем говорит. Еще бы, ведь Тони платит ему деньги.

— Буду рад познакомиться с ним! — ответил я, и мы завершили разговор.

Крис написала мне три часа спустя. Тони хочет видеть меня завтра утром. Она говорит, что Адриан положительно отзывался обо мне. Видимо, они заинтересованы. Предыдущие две компании пока молчат, так что я вполне могу работать на Тони. Хотя я пока даже не знаю, сколько они готовы платить, в объявлении указано, что оплата труда «достойная».

Я даже не хочу представлять себя работающим на кого-то. Я могу вполне комфортно чувствовать себя в их офисе, притворяясь, что мне это интересно, смеясь над шутками Тони и спрашивая у Адриана, как дела у его детей. Но я не хочу писать для них код или, хуже того, быть ответственным за их технические сбои. А это то, что они попытаются сделать: переложить на мои плечи как можно больше дел.

**«Сколько может платить работодатель?» — правильный вопрос, который должен задать себе специалист.**

Я вовсе не лентяй. Я люблю писать код, но делать это для того, чтобы другой человек стал миллионером, — нет уж, спасибо. Моя жизнь стоит больше, нежели зарплата, которую они могут предложить. Честно говоря, я думаю, что именно такое мое отношение и было реальной причиной проблем, с которыми я столкнулся на предыдущей работе (да и на нескольких работах до того). Все хотели, чтобы я был хорошим сотрудником, а я хотел заниматься тем, что мне нравится: воплощать собственные идеи с помощью написания кода на Java. Меня уже четыре раза увольняли, а ведь мне всего 29. Пока не очень впечатляющая карьера. Что я делаю не так?

Я много раз слышал от разных людей: когда тебя собеседует работодатель, ты тоже должен его собеседовать[1]. Подобный подход мне кажется правильным, но не потому, что нужно быть разборчивым при выборе компании, частью которой мы хотим стать, — все они на начальном этапе одинаковы, просто разные названия, рынки и бюджеты. Мы должны «пособеседовать их со своей стороны», чтобы продемонстрировать, что мы особенно заинтересованы в них. Это как знакомство с девушкой[2]: нужно задавать вопросы и делать вид, что вас интересует ее душа, а не только тело.

Поначалу каждая новая компания, команда или проект — загадка. Неважно, сколько вопросов вы задаете об их DevOps-процессах, принципах управления и статическом анализе, ведь любой ответ может быть ложью, их выдумкой или все может работать совсем не так, как они это себе представляют. Я никогда не слушаю, что они говорят на первых собеседованиях. Вот на что я обращаю внимание: 1) деньги, которые они мне готовы заплатить; 2) численность команды; 3) должность, которую мне предложат.

Вопрос денег очевиден: чем больше зарплата, тем лучше. Не только потому, что я жадный и предпочел бы «Мерседес-Бенц», а не «Шевроле», но и потому, что хорошее финансирование означает, что проект ценный и интересный. Рассмотрите это с точки зрения рынка: если они могут заплатить больше,

**Заработная плата — это основной явный показатель важности вашего вклада в рынок.**

[1] *Powers P.* Winning Job Interviews. Career Press, 2005: «Очевидно, что они собеседуют вас, но на самом деле вы их тоже собеседуете. ...Даже если интервьюер не интересуется, есть ли у вас вопросы, подготовьте пару хороших вопросов и попытайтесь задать их».

[2] *Harvey S.* Act Like a Lady, Think Like a Man. Amistad, 2014: «Когда к вам подходит мужчина — у него уже есть план. И главный план — переспать с вами (или выяснить, что нужно для того, чтобы переспать с вами)».

чем другие, у них откуда-то есть деньги. Возможно, им удалось привлечь крупных инвесторов (значит, те поверили в их идеи); быть может, они уже получают приличный доход (из этого следует, что потребитель ценит их больше, чем конкурентов). Или, может, основатель унаследовал парочку миллионов и вложил их в сумасшедший стартап (это значит, что вместо того, чтобы быть промотанными в Вегасе, эти доллары разогревают рынок через бизнес, участвовать в котором я приглашен).

В любом случае деньги — показатель важности данного конкретного бизнеса на рынке. Чем больше денег, тем больше потребность рынка в этом проекте. Когда у проекта, над которым вы работаете, заканчиваются деньги — самое время перейти к другому, более важному для рынка. Подобный подход может показаться нелояльным, но все же он справедлив, если вы заботитесь о своих клиентах, а не о боссах, которые вам урезают зарплату каждые пару недель.

Выбирать проект, в котором платят меньше и который «выглядит более интересным», и несправедливо, и нелогично. Не программистам решать, насколько это интересно и ценно. Такие решения должен принимать рынок, представленный платежеспособными клиентами или инвесторами. И они доносят до нас принимаемые решения с помощью наших зарплат. Когда зарплата повышается — рынок доволен тем, что мы для него делаем. Когда зарплата понижается — пора задаться вопросом: почему вы все еще делаете эту работу для рынка, если рынок ее больше не ценит?

Второй важный для меня показатель — размер компании. Слишком маленькие и слишком большие не годятся. Когда компания слишком мала, технические специалисты вынуждены совмещать очень много обязанностей, выполняя разноплановую работу: написание кода, конфигурирование серверов, подготовку презентаций для инвесторов и даже расстановку мебели и починку кофе-машины. С точки зрения карьеры это деградация и риск потери времени. В итоге придется выполнять кучу работы, не имеющей отношения к непосредственной должности, чтобы просто помочь команде выжить.

> Большие компании слишком политичны, маленькие — слишком хаотичны.

А шансы на выживание маленьких компаний довольно малы[1] (так утверждает статистика[2]). Я предпочитаю держаться подальше от проектов, в которых занято меньше 20 технических специалистов.

С другой стороны, у всех больших компаний есть проблема другого рода: политика. Люди там не работают — они сражаются друг с другом[3]. Я либо выживу на нижнем уровне корпоративной иерархии, нося титул «старшего разработчика» и получая кружку раз в год на день рождения, либо стану мастером интриг в какой-то определенной группе кретинов. Ни один из вариантов меня не привлекает. Так что я не хотел бы идти в компанию, в которой работает свыше тысячи человек.

Третий фактор, на который я обращаю внимание, — это предлагаемая мне должность. Я имею в виду фактическую позицию в иерархии. Для всех компаний характерна иерархия сотрудников, что бы там ни говорили адепты холакратии[4]. Всегда есть начальник высшего звена, затем идут люди, которые ему подчиняются (вплоть до самого нижнего уровня). Пребывания на нижнем уровне я всегда стараюсь избегать. Не только потому, что у меня есть чувство собственного достоинства, но и потому, что я ленив. Чем ниже вы

---

1 *Giardino C. et al.* Why Early-Stage Software Startups Fail: a Behavioral Framework. International Conference of Software Business, 2014: «Более 90 % стартапов не достигают успеха в основном из-за саморазрушения, а не из-за конкуренции».

2 *Griffith E.* On Message, Off Target. Fortune, July 2017: «Cambridge Associates, глобальная инвестиционная компания, расположенная в Бостоне, отслеживала эффективность венчурных инвестиций в 27 259 стартапах с 1990 по 2010 год. Исследования показали, что реальный процент неудачных стартапов, поддерживаемых венчурными средствами, не поднимался выше 60 % с 2001 года. Даже на фоне краха доткомов в 2000-м процент отказов достиг 79 %».

3 ★ *Owen J.* Management Stripped Bare: What They Don't Teach You at Business School. 3rd edition, Kogan Page, 2012.

4 *Badal J.* Can a Company Be Run as a Democracy? Wall Street Journal, April 2007: «Триада работает как демократия, и каждое решение должно приниматься единогласно. ...Лидеры триады, принявшие систему [управления организацией] в ответ на корпоративные распри в другой небольшой компании, где они встретились, уверяют, что она работает».

в иерархии, тем больше работы вам приходится выполнять и тем меньше денег вы получаете. Разделение труда в действии (и не только в сфере программного обеспечения)[1].

Следовательно, чем ниже предлагаемая позиция и чем она более техническая, тем больше будет трудностей: мне придется работать на кого-то, а не на себя. Это именно то, что я ненавижу делать. Я готов вкалывать, когда знаю, что результат будет моим. Но в командах разработчиков с жирной управленческой прослойкой более низкие уровни создают доход, который потребляется более высокими уровнями. Так какой мне смысл оставаться на нижнем уровне?

Вы когда-либо слышали об эксперименте, который Дидье Дезор с коллегами провел в университете Нанси?[2] Они взяли десять одинаковых клеток, в каждую посадили пять или шесть самцов крыс. Чтобы добыть пищу, грызунам нужно было пробраться сквозь маленькое отверстие, через аквариум доплыть до кормушки с пищей, взять немного еды и уплыть обратно, чтобы съесть ее (есть прямо из кормушки было невозможно). К концу эксперимента во всех клетках крысы разделились на две группы — добытчиков и недобытчиков. Добытчики плавали за едой, а недобытчики эту еду у них воровали.

Чему нас учит эта история? Тому, что вы либо работаете, либо крадете. Крысы не знали о десяти заповедях. Они понятия не имели, что воровство — грех[3]. Похоже, природа не имеет ничего против воровства (для нее это естественное распределение ролей). Точно так же для одних людей работа выступает предпочтительным видом деятельности. Для других более привычно воровство.

---

[1] *Piketty T.* Capital in the Twenty-First Century. Harvard University Press, 2014: «Чем выше кто-то поднялся в иерархии доходов, тем более впечатляющим будет рост. Даже если количество человек, получающих выгоду от такого повышения заработной платы, довольно ограниченно, они все же достаточно заметны, и эта заметность, естественно, поднимает вопрос о том, чем оправдан такой высокий уровень компенсации».

[2] *Desor D. et al.* Potential Stock Differences in the Social Behavior of Rats in a Situation of Restricted Access to Food. Behavior Genetics, Volume 25, Number 5, 1995.

[3] «Исход», 20:15: «Не укради». Десять заповедей, согласно книге Исход в Торе и Библии, — это законы жизни, данные Богом Моисею на горе Синай, чтобы передать их народу Израиля.

Мы, конечно, не крысы, и наше поведение намного сложнее, но общий принцип тот же: мы либо работаем, либо присваиваем результат работы других. Иерархии управления были придуманы для того, чтобы контролировать этот процесс и избежать постоянных боев (как между крысами в клетках). Мы больше не боремся с нашими начальниками, мы просто даем им те блага, которые производим[1], веря в справедливость[2]. Они, конечно, сами тоже что-то производят, но польза, которую они приносят, явно меньше, а зарплата выше[3].

**Чем выше вы в иерархии, тем меньше нужно делать именно вам и тем больше результатов других вам приписывают.**

Исходя из этого наиболее удобная для меня позиция в современной системе «добытчиков и недобытчиков» — выглядеть как добытчик, но получать оплату как недобытчик. Иными словами, выглядеть как усердно работающий инженер ПО, в реальности не делая практически ничего[4].

---

1 *Дюркгейм Э.* О разделении общественного труда. — 1893: «Разделение труда есть результат борьбы за существование; но оно представляет собой ее смягченную развязку. Благодаря ему соперники не вынуждены истреблять друг друга, но могут существовать бок о бок».

2 *Adams J. S.* Toward an Understanding of Inequity. The Journal of Abnormal and Social Psychology, Volume 67, 1963.

3 *Gilbert D. L.* The American Class Structure in an Age of Growing Inequality. 8th Edition, Pine Forge Press, 2011: «С 1980 по 2000 год реальная прибыль CEO выросла более чем на 600 % — это значительное увеличение в эпоху застоя заработной платы. Пропасть, отделяющая типичного CEO (директора) от типичного американского рабочего, выросла до колоссальных размеров. Среднестатистический CEO крупной корпорации в 1999 году заработал в 475 раз больше среднестатистического рабочего; в 1980 году их доходы различались в 42 раза».

4 *Maier C.* Bonjour Laziness: Why Hard Work Doesn't Pay. Vintage, 2006: «В крупных компаниях ищите наиболее бесполезные позиции: консультирование, оценка, исследования и обучение. Чем более бесполезна ваша позиция, тем меньше возможностей оценить ваш "вклад в деятельность фирмы". Избегайте, как чумы, рабочих ("локальных") позиций. Цель в том, чтобы стремиться к чему-то "на обочине": к типу "межведомственных" позиций, которые не имеют никакого значения и не подвергаются корпоративному давлению. Короче говоря, держитесь в стороне».

Подавляющее большинство команд в сфере разработки ПО не способны заставить программистов выкладываться по полной. Руководство, особенно на высших уровнях, не понимает разницу между Java и JavaScript. С другой стороны, на более низком уровне сложнее одурачить начальника, притворяясь, будто пишешь код, а на самом деле публикуя в это время новый пост в Facebook. Вот почему чем выше моя позиция, тем лучше. Никто не поймет, что я на самом деле делаю, поэтому я могу делать то, что хочу. Я участвую в проектах, за которые мне платят, но предпочитаю делать это тогда, когда мне хочется и когда это действительно необходимо. А не в девять утра.

## Второе собеседование

Офис выглядит симпатично, оформлен в минималистическом ключе. На стене в стиле поп-арт написаны слова «впечатляй» и «думай». Все остальное — абстрактные прямоугольники. Может быть, мне лучше было бы стать художником? Уверен, что я тоже могу нарисовать нечто подобное. Я могу даже добавить туда больше слов, например «воруй» и «доминируй», хотя не думаю, что многие захотели бы заказать такое для своих офисов.

Входит Тони. Ему за 40, он высокий, носит бороду, одет в черную рубашку и джинсы. Естественно, он улыбается. Я сомневаюсь, что он вообще знает, кто я такой. Интересно, где же Адриан?

— Адриан позвонил, сказал, что заболел, — говорит Тони, и мы идем по коридору в одну из переговорных. Стандартный вопрос про «чай или кофе», я прошу стакан воды. Он уходит и возвращается спустя минуту со стаканом.

Появляется Крис, с которой я уже познакомился. Скорее всего, она здесь директор по персоналу, а это значит, что компания не такая уж маленькая. Я до сих пор не определил размеры их офиса. И даже не знаю, с чем они работают. Я, наверное, должен был получше подготовиться к собеседованию. Но я никогда не делаю этого и, скорее всего, немного обижаю моих потенциальных работодателей — все они хотят видеть, насколько соискатели в них заинтересованы. Искренне говоря, я не настолько уж и заинтересован, но невежливо это демонстрировать. Однако уже слишком поздно: Тони и Крис сидят передо мной, я кладу телефон на стол экраном вниз и улыбаюсь.

**Очень часто руководителями являются вовсе не те, у кого есть лучшие навыки или пропуск во все помещения.**

Интересно, зачем позвали Крис? Ее не было с нами, когда я беседовал с Адрианом. Думаю, причины две. Во-первых, Тони — ее босс и она хочет показать ему, как усердно она работает, чтобы найти

новые таланты. Во-вторых, она будет принимать хоть какое-то участие при окончательном решении о найме. Стало быть, я должен произвести на нее впечатление?

На самом деле второе собеседование намного важнее, чем первое. Несмотря на то что именно Адриан способен оценить мои профессиональные качества, решение принимать будут Тони и Крис, почти не разбирающиеся в разработке программного обеспечения. И я уверен, что их решение будет основано исключительно на эмоциях. Поэтому я должен им понравиться.

— Как прошло собеседование с Адрианом? — начинает Крис.

Звучит как невинный вопрос, но здесь не все так просто. Если я скажу, что «все было здорово», это будет выглядеть, будто я не хочу делиться своими впечатлениями (либо потому, что у меня их нет, либо потому, что я чувствую какой-то негатив). Ответ «отлично» тоже не самый лучший вариант в этой ситуации. Похоже просто на формальную вежливость. С другой стороны, рассказывать слишком много о том, как я на самом деле отношусь к этому собеседованию, рискованно, потому что я думаю только о том, когда меня собираются нанять. И я не считаю Адриана технически более подкованным, чем я. Но им это знать не обязательно.

**Лесть — самый мощный инструмент, когда имеешь дело со слабыми и глупыми.**

Я должен выдать им что-то среднее. Что-то вроде: «Это было интересно, особенно это и то, но то и это озадачило меня». Такой ответ продемонстрирует, что я думал о том, о чем говорилось на первом собеседовании, и об их компании в течение нескольких дней, что у меня возник ряд вопросов, на которые я хочу получить ответы, что я тот, кто будет предан их идее, и т. д. и т. п. Ненавижу эти слова, но в такой ситуации они работают.

Я продолжаю: «Я действительно наслаждался нашей беседой. Большинство собеседований проходят скучно, говорят в основном о кандидате. А с Адрианом мы общались как два программиста — и обо мне, и о вашем проекте. Но все же я не понял, каков ваш долгосрочный план в отношении программного обеспечения?» Я просто

вернул мяч обратно. Все любят говорить о себе, а такие боссы, как Тони, больше думают о планах и стратегиях, рассказывают о долгосрочной перспективе и миссии компании. Я уверен, что он будет рад поведать мне больше о себе и команде.

— О, отличная мысль, позвольте мне рассказать, как мы его видим.

Тони заглатывает наживку и начинает рассказ. Крис, я уверен, слышала эту историю много раз, но изображает крайнюю заинтересованность. Еще бы, это ее начальник, и она должна усердно демонстрировать свою лояльность, верно?

Пока Тони излагает свое глобальное видение, я думаю о том, как бы подготовиться к следующему вопросу. Лучшая защита — нападение; соответственно, лучший способ избежать вопросов — самому задавать их.

— Значит, вы считаете, что сейчас для вас важно качество? — я перебиваю его сразу после того, как он упомянул, сколько нововведений они внедрили за последнее время в области Continuous Integration.

— Оно было важно всегда, но сейчас это абсолютно критично. База пользователей растет, и мы не можем позволить себе поставить под угрозу нашу репутацию, поэтому должны стараться поддерживать бесперебойную работу серверов.

— Они слишком часто отказывают? — я притворяюсь, что беспокоюсь за их серверы и их бедных клиентов.

**Качество и ответственность ничего не значат, если они не относятся к конкретным людям.**

Похоже, Тони не понимает, что неотвечающие серверы — это его личная ответственность как технического директора. Он уже должен быть уволен. Забавно, насколько техническим директорам нравится рассказывать о важности качества услуг, при этом не понимая, что качество в первую очередь означает личную ответственность. Для них «качество на первом месте» — просто маркетинговый слоган.

— Ну, не то чтобы... Но все же нам нужно улучшить качество.

Очевидно, что он лукавит. Так ведет себя большинство технически невежественных техдиректоров. Они боятся потерять свои насиженные места и вынуждены одновременно делать две вещи: требовать лучшего качества и быть аккуратными, чтобы не брать вину на себя. Они не могут сказать: «Наш парк серверов крайне нестабилен, и это моя вина», потому что на следующий вопрос, который задаст генеральный директор или совет директоров: «Почему тогда мы вам платим полмиллиона долларов в год?» — они не смогут дать вразумительный ответ. С другой стороны, они должны что-то сказать о нашей ответственности за качество, предпочитая не указывать, что подразумевается под словом «наша».

— Здорово, что вы понимаете важность! Очень необычно видеть руководителя высокого уровня, настолько разбирающегося в технологиях! — даже не могу поверить, что говорю это. Тем не менее это работает. Тони польщен и готов беседовать о качестве еще полчаса. Но вот Крис, кажется, раскусила мои уловки.

— Не могли бы вы рассказать о своем опыте? — перебивает она.

Это простой вопрос, потому что они и не понимают толком, с чем я работаю. Я начинаю рассказывать о продуктах, которые я использовал, о фреймворках, инструментах и языках. Они выглядят заинтересованными, продолжают кивать, но я знаю, что для них мой рассказ не имеет никакого смысла. Пришло время произвести на них впечатление. Такие ребята, как Тони, занимающие высокие посты, любят цифры и какие-то громкие фразы, которые они могут запомнить. Они хотят гордиться тем, что нанимают «хороших» сотрудников[1]. Они должны чувствовать, что поймали большую рыбу. В этом и заключается их работа: искать людей на рынке, а затем заставлять их работать.

---

[1] См. сноску 2 на с. 130: «Таким образом, отличная производительность труда сама по себе недостаточна и может даже не понадобиться для получения и удержания руководящего поста. Вас должно заметить руководство, вы должны влиять на показатели, используемые для измерения ваших достижений, и главное — убедить в своей эффективности тех, кто у власти».

Я показываю им несколько своих продуктов с открытым исходным кодом. Они довольно популярны на GitHub. Я говорю им, что создал их в свободное время и что количество моих последователей растет. Их это впечатляет. Уверен, что они не часто слышат что-либо подобное в этой комнате. Большинство программистов, сидящих в этом кресле, рассказывают истории о каких-то мертвых стартапах, на которые потратили долгие годы, называют имена давно уволенных техдиректоров, у которых они были в подчинении. Это скучно и совсем не впечатляет. У меня есть проекты с открытым исходным кодом, которые сейчас на рынке. Я известен в сообществе программистов, и Тони это понимает.

— Планируете ли вы продолжать работу над этой библиотекой с открытым исходным кодом? — интересуется Крис.

Коварный вопрос. Понятное дело, они не хотят, чтобы я тратил время на что-то еще, кроме их бизнеса. Они не понимают, что я ценен для них именно потому, что работаю над проектами с открытым исходным кодом. Они хотят найти профессионала и «заковать его в цепи». Но это попросту невозможно. Это можно сравнить с тем, как девушки предпочитают парней, пользующихся популярностью у других девушек, а выйдя замуж за такого плейбоя, хотят, чтобы он игнорировал других женщин. Они не понимают, что это невозможно: либо они превратят парня в своего рода домашнего питомца, которого больше не смогут любить. Парадокс, не правда ли?

Я профессионал, потому что я свободен, потому что сам что-то развиваю, потому что присутствую на рынке и пробую что-то новое каждый день. Они вполне осознают, что такой программист в команде как глоток свежего воздуха. Но продолжение свободной жизни им не понравится. Они ожидают, что я перестану делать все то, что делал раньше, и выйду за них замуж навсегда. Я не могу этого сделать. Однако не могу и сказать об этом вслух. Тоже парадокс, не так ли?

**Независимость делает нас профессионалами, но не ценится, когда мы работаем в команде.**

Приходится выкручиваться:

— Ну, сейчас это только поддержка, несколько часов в выходные дни. Я просто не могу бросить свои продукты на произвол судьбы, но, само собой, я хочу посвятить все свое время более серьезному проекту.

Я назвал их проект более серьезным, и они чувствуют гордость. Уверен, именно такой ответ они хотели услышать. Конечно, на самом деле это не просто поддержка. Естественно, мои собственные проекты для меня гораздо важнее, чем их бизнес. Конечно, я буду тратить отнюдь не несколько часов в выходные дни. Но я еще не встречал ни одного владельца бизнеса или даже наемного руководителя, который чувствовал бы себя комфортно, услышав честный ответ: «Мои проекты куда важнее, привыкайте к этому». Интересно, есть такие?

Пожалуй, корень проблемы здесь в том, что они не могут должным образом управлять нами, инженерами[1]. Они не могут контролировать, что мы делаем и какие результаты получаем, потому что они и в технологиях отстают, и как управленцы полные непрофессионалы[2].

Единственное, что они могут, — это проверить, работаем ли мы в офисе с чувством вины, также называемым ответственностью[3]. Они не знают[4], как правильно планировать проект, как

---

[1] *Furuyama T.* Fault Generation Model and Mental Stress Effect Analysis. Journal of Systems and Software, Volume 26, Issue 1, 1994: «71 % всех ошибок находятся в зоне ответственности руководителя проекта, специалиста по обеспечению качества и высшего руководства».

[2] *Barker R.* The Big Idea: No, Management Is Not a Profession. Harvard Business Review, Volume 88, Number 7, 2010: «Управление — это вовсе не профессия и никогда не может быть ею».

[3] *Rasmusson J.* The Agile Samurai: How Agile Masters Deliver Great Software (Pragmatic Programmers). Pragmatic Bookshelf, 2010: «Хорошая agile-команда всегда будет нести ответственность за результат. В команде все знают, что клиенты рассчитывают на них, и с самого первого дня стремятся приносить пользу».

[4] *Charette R. N.* Why Software Fails. IEEE Spectrum, Volume 42, Number 9, 2005: «Недальновидные решения руководителей проектов, вероятно, являются основной причиной сбоев программного обеспечения, тогда как неэффективное техническое управление может привести к техническим ошибкам, но их, как правило, можно локализовать и исправить».

ставить задачи, как проверять, действительно ли мы создаем что-то важное, как убедиться, что качество результатов приемлемо, и многое другое из того, что должен делать профессиональный управленец[1].

Вместо этого они популяризируют миф[2] о том, что разработка программного обеспечения — творческий процесс[3], и трусливо полагаются на наше желание быть «хорошими»[4] и «все делать правильно»[5].

---

[1] *Jones C.* Software Project Management Practices: Failure Versus Success. CrossTalk: «Сравним крупные проекты, которые успешно уложились в бюджет и график, с теми проектами, которые были выпущены позже намеченного срока, у которых заканчивался бюджет или которые не были завершены. У последних мы видим шесть общих проблем: плохое планирование проекта, плохую оценку затрат, плохие измерения, плохое отслеживание этапов, плохой контроль изменений и плохой контроль качества. Тогда как успешные программные проекты, как правило, в среднем были лучше по всем этим шести показателям. Возможно, наиболее интересной особенностью этих шести проблем является то, что все они связаны с управлением проектами, а не с техническим персоналом».

[2] *Anthes G. H. et al.* Lessons From India Inc. ComputerWorld, April 2001: «Существует миф о том, что разработка программного обеспечения — это творческий процесс, который в значительной степени зависит от индивидуальных усилий каждого. Это не так. По окончании первоначального этапа определения проекта и спецификации разработка становится трудоемкой механической работой».

[3] *Larman C.* Agile and Iterative Development: A Manager's Guide. Addison Wesley, 2003: «Большая часть программного обеспечения — это не задача массового или прогнозируемого производства. Разработка программного обеспечения — это разработка нового продукта. Поскольку модель прогнозируемого производства не подходит для программного обеспечения, заложенные в этой модели методы и ценности не будут полезны в этой отрасли».

[4] ❦ *Cockburn A. et al.* Agile Software Development: The People Factor. IEEE Computer, Volume 34, Number 11, 2001: «Если люди, занятые в проекте, достаточно хороши, они могут выполнить свое задание с помощью практически любого метода. Если они недостаточно хороши, ни один метод не поможет. Иными словами, "люди выше процессов"».

[5] ❦ *Highsmith J.* Agile Software Development Ecosystems. Addison Wesley, 2002: «Мы понимаем, что талантливые люди, обладающие внутренней мотивацией, способные работать в нестабильной среде и понимающие концепцию продукта, будут в лучшем виде делать все от них зависящее».

Они пропагандируют философию, согласно которой процессы управления не столь важны, как «гениальность»[1] людей. Они ожидают, что эти мифические «гениальные» программисты будут лояльными, приверженными корпоративным ценностям[2], ответственными, станут вести себя героически[3], гибко мыслить[4] и будут готовы работать под некомпетентным руководством[5] или вообще без него[6].

---

1 *Cohn M.* Agile Estimating and Planning. Prentice Hall, 2005: «Agile-команды ценят людей и взаимодействия по процессам и инструментам, так как знают, что хорошо функционирующая команда толковых людей с посредственными инструментами всегда будет превосходить команду посредственных сотрудников с замечательными инструментами».

2 См. сноску 5 на с. 126: «В сущности, программы корпоративной культуры призваны препятствовать развитию условий, в которых поощряется критический самоанализ. Они приветствуют унификацию норм и ценностей в организациях. Сотрудников отбирают и повышают на основе их (предполагаемого) принятия основных ценностей или восприимчивости к ним. В целом сотрудников периодически поощряют и награждают за отсутствие приверженности к идеям, которые не подтверждают и не укрепляют авторитет основных ценностей».

3 *James B.* Enough About Process: What We Need Are Heroes. IEEE Software, Volume 12, Number 2, 1995: «...подход можно назвать моделью "ковбой" или "большая магия". Согласно подобному видению одаренные люди создают программное обеспечение с помощью явно магических средств, без каких-либо конкретных указаний или поддержки. Этот подход также предусматривает наличие героев, но скорее в негативном смысле. Здесь не делается ничего для того, чтобы их взрастить. Скорее это их изматывает».

4 *Morris L.* Agile Innovation: The Revolutionary Approach to Accelerate Success, Inspire Engagement and Ignite Creativity. Wiley, 2014: «Хотя процессы и инструменты могут служить подспорьем для эффективной работы компетентных людей, нет ничего важнее основательного подхода и осмысленного сотрудничества, когда дело доходит до создания чего-то нового».

5 *Cockburn A.* Agile Software Development: The Cooperative Game. 2nd Edition, AddisonWesley Professional, 2006: «Слаженно работающая команда адекватных людей завершит проект практически независимо от процесса или технологии, которые им предлагается использовать».

6 *Adkins L.* Coaching Agile Teams: A Companion for ScrumMasters, Agile Coaches and Project Managers in Transition. Addison-Wesley, 2010: «Вот как это работает: вы не доверяете команде, поэтому говорите им, что делать. Они делают то, что вы сказали, а не то, что, по их мнению, должны делать. Результаты не те, которых вы ждали, поэтому вы снова говорите им, что делать, на этот раз более четко. И цикл продолжается. В итоге все теряют доверие».

Но нами, программистами, движет не лояльность или внутренняя мотивация. Мы должны писать наш код тогда, когда отчетливо видим путь к личному успеху, и написание высококачественного кода помогает нам двигаться вперед по этому пути. А чтобы это произошло, руководство должно определить правила игры, известные под названием «процесс»[1], и обеспечить их строгое соблюдение, что намного сложнее, чем «проявлять гибкость».

> Слабые руководители, не способные грамотно организовать работу над проектом, могут полагаться только на героизм людей.

Я бы сравнил проект со страной: либо ею должным образом управляют с помощью законов, либо она порабощается диктатором, которого все должны любить. В большинстве современных компаний руководители используют последний сценарий. Они ожидают, что мы будем любить клиентов и работать только из-за этого. Здесь нет законов, дисциплины, правил и принципов, потому что, как и любой диктатор, они недостаточно компетентны в их создании. Диктаторы захватывают власть и управляют силой[2]: это гораздо проще, чем создавать систему законов, которая будет

---

[1] См. сноску 2 на с. 119: «Некоторыми людьми в сообществе разработчиков программного обеспечения слово "процесс" рассматривается почти как матерное. Эти люди воспринимают "программные процессы" как жесткие, ограничивающие и неэффективные. Они думают, что отличный способ запустить проект — нанять лучших людей, которых только можно найти, предоставить им все необходимые ресурсы, а также свободу, чтобы они делали то, что у них получается лучше всего. Согласно этой точке зрения проекты, которые выполняются без какого-либо внимания к процессу, могут быть чрезвычайно эффективными. ...Проекты, в которых не уделяется внимание налаживанию эффективных процессов на ранних этапах, вынуждены внедрять их впоследствии, когда это занимает больше времени и приносит меньше пользы. ...Если в проекте слишком мало внимания уделяется используемым процессам, к концу проекта разработчики будут проводить все свое время на совещаниях, исправляя ошибки, и почти не смогут развивать программное обеспечение. Проект будет пребывать в состоянии стагнации».

[2] *McClelland D. C.* Power Is the Great Motivator. Harvard Business Review, January 2013: «Власть без дисциплины часто направлена на возвеличивание руководителя, а не на пользу компании».

работать сама по себе[1]. Менеджмент в программных проектах также не может создать эффективную систему управления, поскольку у него нет необходимых для этого знаний. Вместо этого руководители ожидают нашей любви. Разве не очевидно, что довольно скоро эта любовь превратится в ненависть и либо мы уйдем, либо проект прикажет долго жить?

Теперь посмотрим, сколько любви эти ребята ожидают от меня. По моему опыту, как правило, чем больше они ожидают, тем слабее их менеджмент. Что не так уж и плохо: они не будут управлять мною, я буду изображать любовь, а они будут платить мне.

— Отлично, это то, что мы хотели услышать, — Крис выглядит довольной.

Если я подойду им, это будет в первую очередь ее успехом. Она опубликовала это объявление на платформе Craigslist, и я отправил ей письмо первым. Она не хочет, чтобы я провалил собеседование, особенно после первой беседы с Адрианом. Так что она определенно на моей стороне.

— Мы понимаем важность открытого исходного кода, — ответ Тони звучит убедительно, хотя я абсолютно уверен, что лично он не написал ни единой строчки открытого исходного кода. Скорее всего, он создал пару программ на «Паскале» еще в университете, а затем просто переключил свою карьеру на менеджмент и зарабатывание денег. И сейчас он «понимает важность открытого исходного кода». Смешно!

---

[1] *Moore B. Jr.* Social Origins of Dictatorship and Democracy: Lord and Peasant in the Making of the Modern World. Penguin, 1966: «Автор рассматривает развитие демократии как долгую и пока, конечно, не завершенную борьбу за достижение таких трех сравнительно близких целей, как: 1) ограничение самовластия правителей; 2) замена произвольных правил правления справедливыми и рациональными; 3) обеспечение участия населения в выборе правителей. Казнь королей была наиболее драматичным, но ни в коем случае не наименее важным аспектом первой цели. Усилия по установлению верховенства права, авторитета законодательной власти, а впоследствии использование государства в качестве механизма для обеспечения человеческого благосостояния — знакомые и известные аспекты двух других».

— Круто, это очень необычно — встречаться с техническим директором, который верит в открытый исходный код!

Я буду продолжать ему льстить. Похоже, ему нравится то, что я говорю. Да и Крис на моей стороне. Так что полный порядок.

— Можете ли вы рассказать нам о самой серьезной ошибке, которую вы сделали в своей карьере? — спрашивает она и улыбается. Скорее всего, она прочитала этот вопрос в книге «100 вопросов, которые нужно задавать на собеседованиях». Ну, я должен играть по их правилам и выдать что-нибудь. Само собой, я не буду рассказывать им о неверной конфигурации сценариев развертывания, из-за которой в прошлом году наши тестовые учетные записи к базе данных были выложены на производственный сервер и несколько тысяч пользователей лишились доступа к своим учетным записям на несколько часов, пока мы решали проблему и занимались повторным развертыванием. Нам повезло, мы обнаружили ошибку всего лишь через 40 минут после развертывания. И да, это была моя вина, поскольку именно я писал скрипты.

**Честность в бизнесе — не что иное, как недостаток квалификации.**

Также я не буду рассказывать им историю, когда меня уволили, потому что мой непосредственный начальник увидел мое резюме на сайте по поиску работы. Да, было весело, я хорошо помню тот день. Я появился в офисе, и он сказал: «Нам нужно поговорить». Вы знаете, каково это, так ведь? Ты всегда ощущаешь привкус этого последнего «разговора». Я тоже. Он сказал: «Чувак, мы нашли твое резюме на сайте по поиску работы, похоже, ты к нам не лоялен. Так что мы решили тебя отпустить». Ну что тут сказать? Я был смущен. Не потому, что я его там разместил (каждый профессиональный разработчик должен это делать), а потому, что я не позаботился о достаточной анонимности своего CV.

Серьезно, воистину профессиональные программисты обязательно должны быть на рынке постоянно, даже когда они работают. Особенно когда работают. Мы всегда должны быть открыты для новых возможностей, чтобы оставаться в форме, иметь возможность

сравнивать варианты, понимать рынок и быть лучше других. Мы должны видеть тех, других, должны ходить на собеседования, оценивать, что другие могут предложить, чтобы понять, насколько мы хороши для них. Если они перестают звонить нам или предлагать работу — это сигнал, что с нами что-то не так и пора это исправить.

Без данной информации мы быстро становимся ограничены одним офисным пространством, одним проектом, одним технологическим стеком, одной командой и одним руководителем. Несомненно, мы будем эффективны и на этой закрытой территории, но в конечном итоге наша эффективность станет снижаться и мы начнем деградировать.

**Демонстрируемая способность учиться на ошибках — наиболее привлекательная черта кандидата в команду.**

Большинство работодателей не понимают этого, так же как они не понимают важности для нас открытого кода[1]. Они действуют как вышеупомянутые девочки, которых привлекают плейбои. Во что превращаются эти плейбои? В скучных мужей, лежащих на диване с кружкой пива и пультом от телевизора в руках. Без открытого исходного кода и без рынка мы не можем быть профессионалами. Мы должны быть на виду и постоянно держать форму.

Я не буду им это рассказывать. Лучше поведаю им что-либо, что они наверняка хотели бы услышать.

— Это случилось два года назад, когда я только начал работать над своим предыдущим проектом. Был один парень, который мне не очень нравился. Он противился абсолютно всему, что я предла-

---

[1] *Hars A. et al.* Working for Free? Motivations of Participating in Open Source Projects. Proceedings of the 34th Annual Hawaii International Conference on System Sciences, IEEE, 2001: «Мотивация участия в проектах с открытым исходным кодом оказалась более сложной, чем ожидалось. Важную роль играли как внутренние факторы (внутренняя мотивация, альтруизм и идентификация с сообществом), так и внешние (прямая компенсация и ожидаемая отдача). Также более значительными, нежели ожидалось, были факторы, которые обещали будущие денежные вознаграждения, такие как позиционирование и продвижение себя на рынке труда».

гал, и мы явно не ладили, хотя были в одной команде. Я не был его боссом, но я был архитектором и должен был давать ему задания. Но это было невозможно, и я даже попросил, чтобы технический директор уволил парня. Я потратил кучу сил, энергии и эмоций на регулярные перепалки с ним. Но то, что произошло позже, меня немало удивило. Однажды некоторые сотрудники задержались в офисе — мы готовились к очередному релизу. И этот парень стал рассказывать нам о своем собственном фреймворке с открытым исходным кодом, над которым он работал. Я заинтересовался и начал расспрашивать его более подробно. Спустя несколько недель мы стали друзьями. Он начал уважать меня после того, как я его зауважал. Вот урок, который я усвоил: не говорите людям, что им делать, пока не узнаете их достаточно и не заслужите их уважения. Относиться к этому парню без уважения было одной из моих самых больших ошибок в профессиональной карьере. Все еще чувствую себя неловко из-за этого. Но мы по-прежнему остаемся друзьями.

Я вижу, что Крис готова пустить слезу. Тони улыбается. Они оба идиоты. Вообще, история довольно близка к реальности, что-то подобное действительно произошло. И я действительно верю, что авторитет, основанный на уважении, намного сильнее авторитета, полученного благодаря должности. Но нет, они не идиоты. Это была хорошая история.

У меня всегда есть подобные заготовки для собеседований. Всем работодателям интересны наши «ошибки», и каждая ошибка всегда должна иметь счастливый конец, когда главный герой, то есть я, чему-то учится. Работодатель хочет видеть, какие мы выносим уроки — это означает, что мы обучаемы и управляемы. Никто не любит работать с упрямцами. Точнее, практически никто не знает, как с ними работать. Но они самые лучшие.

Эта история продемонстрировала мою готовность учиться на своих ошибках и делиться своими знаниями. Теперь Тони уверен, что у меня не будет проблем с его программистами: я буду уважать их. Действительно? Ну, давайте сначала посмотрим, кто они такие.

— Да, вы правы. Мы тоже предпочитаем уважать нашу команду, а не руководить ею, — резюмирует Крис с улыбкой. Она выглядит очень серьезной, будто я, ребенок, только что узнал что-то новое, а она, учительница, рада этому.

**Уважение к руководству — привлекательная иллюзия, которая по душе большинству команд и боссов.**

— У вас есть вопросы? — спрашивает Тони, ожидая ответа: «Да, конечно!»

Сейчас моя очередь. Я хочу узнать то, что хочу узнать: зарплату и должность.

— Можно узнать, сколько технического персонала в команде?

Пожалуй, начну с самого невинного вопроса.

— Больше 50, и мы расширяемся. К концу года нас будет 80, — Тони гордится такими цифрами. Действительно, 50 программистов в Калифорнии означает, что годовой бюджет заработной платы близок к 10 миллионам. Это число он наверняка внесет в свое резюме, когда придет его черед искать новую работу. А может, он владелец этого бизнеса? Это я узнаю позже. Он определенно выглядит как владелец.

— Насколько важной будет моя позиция в команде с точки зрения возможного вклада?

Я спрашиваю об этом и горжусь собой. Это правильный способ поставить вопрос, не раскрывая моего сильного желания быть главным и ничего не делать. Вопрос звучит так, будто мне очень интересно внести как можно больший вклад и поэтому мне нужна более высокая должность.

— Нам нужна ваша помощь с серверной частью нашего продукта. Мы думаем о позиции старшего разработчика. Как вы на это смотрите?

Он хочет нанять меня. Сейчас подходящий момент. Мне вообще не нравится название Senior Developer. Это означает написание кода восемь часов в день. Полный отстой. Я должен получить особый статус с самого начала. Потом будет гораздо сложнее это сделать. И Тони — «мой человек». У него очень сильная позиция, и в этот

момент он может пообещать мне кое-что, что я буду использовать позже.

— Адриан — глава серверной группы? — аккуратно делаю ставку.

— Да, мы планировали прикрепить вас к его группе, — объявляет Тони мою ставку.

— Я буду у него в подчинении? — снова делаю ставку.

— Да...

Тони готов поднять ставку, но объявляет через секунду:

— И в моем, конечно!

**Стремитесь к надежным должностям, которые обычно означают расплывчатые обязанности и большую самостоятельность.**

Вот! Это то, что я хотел. Он боится потерять меня. Услышав мой вопрос об Адриане, он сразу понял, что мне не нравится эта идея. Я не хочу быть в подчинении у Адриана, потому что это делает мою позицию намного ниже и не позволяет мне работать напрямую с Тони. Вот почему он добавил про себя! Чтобы поставить меня как минимум на один уровень с Адрианом.

— Было бы здорово работать и с другими группами. Я чувствую, что могу внести свой вклад в других сферах, особенно в DevOps. Я могу быть в вашем подчинении и работать вместе с Адрианом, например. Что вы думаете? — я иду ва-банк.

Тони смотрит на Крис. А Крис — на Тони.

— Давайте подумаем об этом. Мы свяжемся с вами, — резюмирует Тони.

Я уверен, что он согласится. И это будет означать, что я буду в очень привилегированном положении, рядом с Адрианом, без какой-либо личной ответственности за кого-либо. Что-то вроде архитектора. Может быть, даже моя должность будет называться «архитектор программного обеспечения». Они встают и пожимают мне руку. Я пошел домой. Думаю, все прошло хорошо.

# Открытый исходный код

На следующий день я, как и ожидал, получил письмо от Крис. Она говорит, что я для них большая находка, а они — отличная возможность для меня. Они готовы платить 140 тысяч долларов в год, а также оплачивать мою медицинскую страховку. Это довольно приличная зарплата в Кремниевой долине[1]. Давайте посчитаем: 47 524 доллара нужно отдать государству в виде подоходного налога, 42 тысячи я плачу за свою студию в Маунтин-Вью, 10 800 долларов в год — мой автокредит. Следовательно, на карманные расходы остается 3250 долларов в месяц. Сто долларов в день. Неплохо, но очень далеко от того, чтобы быть миллионером.

В письме также говорится, что моя должность — архитектор программного обеспечения и я буду подчиняться Адриану и Тони. Кажется, они не понимают, что невозможно подчиняться двум людям одновременно. Это для меня плюс: они не ожидают, что я не собираюсь подчиняться вообще. И не буду.

**Вреден не контроль, а неспособность правильно его организовать.**

Я сталкивался с такой ситуацией много раз, когда люди действительно не понимают слово «подотчетность» или просто не хотят его понимать. Они думают, что просить кого-то о подотчетности перед ними — это оскорбление, демонстрация силы, контроль[2].

---

[1] *Perry T. S.* Where Are the U. S. Firms That Pay the Best Salaries? Silicon Valley (Mostly). IEEE Spectrum, April 2017: «Google, базирующийся в Маунтин-Вью, оказался под шестым номером со средней общей компенсацией 155 250 долларов. Facebook, расположенный в Менло-Парке, — под седьмым номером со средней общей компенсацией 155 тысяч долларов. Twitter, находящийся в Сан-Франциско, — под номером 22 со средней общей компенсацией 142 тысячи долларов».

[2] *Myatt M.* The Most Misunderstood Aspect of Great Leadership. Forbes, December 2012: «Контроль ограничивает потенциал, инициативу и таланты. Отказ от него способствует сотрудничеству, поощряет инновации и создает новые возможности. Лидеры контроля создают узкие места, а не увеличивают пропускную способность. Они сигнализируют об отсутствии доверия и уверенности и часто оказываются если не высокомерными, то как минимум бесчувственными».

И во многих случаях они правы. Большинство сотрудников весьма чувствительны к слову «контроль», полагая, что свободны и их не нужно контролировать. Мы не понимаем, что злоупотребление управлением, которого мы так боимся, исходит не от тех, кто хочет контролировать, а от тех, кто не знает, как это сделать.

Любая группа людей нуждается в иерархии власти. Неважно, пишет группа код или взбирается на гору[1]. Работа группы должна быть скоординированной, чтобы быть продуктивной. Независимо от того, насколько умны члены группы, насколько они любят друг друга и сколько раз они делали эту работу раньше, им нужны[2] инструкции и субординация, распределенные иерархически. Кто-то наверху должен говорить, что сделать, а все остальные должны подчиняться[3].

Конечно, этот «кто-то» должен быть умным и давать толковые указания. Более того, чтобы быть еще продуктивнее, лидер должен собрать нужную информацию от членов группы, прежде чем давать

---

1 *Berkel Van L.* Hierarchy, Dominance and Deliberation: Egalitarian Values Require Mental Effort. Personality and Social Psychology Bulletin, Volume 41 (9), 2015: «Мы предполагаем, что есть первоначальная тенденция поддерживать иерархию — это проще и быстрее, а также более глубоко укоренено. Эгалитаризм социально ценится в более старшем возрасте, но эгалитарные ценности существуют в контексте этих более старых иерархических ценностей. ...Люди настроены на доминирование и уважение — они играют центральную роль в общественной жизни».

2 *Zitek E.* The Fluency of Social Hierarchy: The Ease With Which Hierarchical Relationships Are Seen, Remembered, Learned and Liked. Journal of Personality and Social Psychology, Volume 102 (1), 2012: «Социальные иерархии — это свободные социальные стимулы, то есть они легче обрабатываются и поэтому больше нравятся, чем менее иерархические стимулы. ...Когда в социальных отношениях трудно разобраться, люди еще больше предпочитают иерархию».

3 *Boehm C.* Hierarchy in the Forest: The Evolution of Egalitarian Behavior. Harvard University Press, 2001: «Для того чтобы добиться стабильной эгалитарной иерархии, основной поток власти в обществе должен повернуть вспять... что требует значительных усилий для поддержания этого состояния. Наша политическая природа способствует формированию ортодоксальных иерархий — таких как иерархия шимпанзе, или горилл, или людей, живущих в племенах либо государствах».

какие-либо инструкции. Но это вторично[1]. Главный фактор успеха групповой координации заключается в том, что все подчиняются начальнику и делают то, что он говорит (это еще называется дисциплиной). Так были организованы все армии[2] с древних времен Тимура и Александра. Разве не так они выиграли свои сражения?[3]

Парадокс современного менеджмента состоит в том, что мы все хотим выигрывать свои сражения, но не желаем слышать такие слова, как «контроль», «подчинение» или «подотчетность». Мы считаем, что управление происходит как-то само по себе, просто потому, что каждый из нас — хороший человек, делающий «правильные вещи». Это может быть отчасти правдой, но, как и в армии, несмотря на то что верность и честность солдат высоко ценятся, намного важнее точно следовать приказам, отданным командирами. Это ключевой аспект.

**Четкие правила подчинения приводят к дисциплине, это приводит к порядку, который, в свою очередь, приводит к успеху.**

Когда я слышу, как мои начальники говорят, что они против контроля и хотят, чтобы я был в подчинении перед всеми и ни перед кем, я сразу понимаю, что буду управлять своим боссом, а не наоборот. Опять же здесь подойдет сравнение с армией. Если командир недостаточно умен и силен и не способен организовать часть таким образом, чтобы каждый солдат знал, кому подчиняться, —

---

1 *Bolton P. et al.* Leadership, Coordination and Mission-Driven Management. The Review of Economic Studies, Volume 80, Issue 2, 2013: «Мы утверждаем, что ключевым качеством хорошего лидера является та форма самоуверенности, которую мы будем называть решительностью. Решительного лидера отличает твердость убеждений: на его мнение мало повлияет новая информация о той среде, в которой работает его организация».

2 ★ *Tzu S.* The Art of War. 5th Century BC: «Если генерал уверен в своих людях, но всегда настаивает на том, чтобы его приказы выполнялись, выигрыш будет обоюдным».

3 *Renatus F. V.* De Re Militari. 390 A. D.: «Люди античного мира, обученные на опыте, предпочитали количеству дисциплину. Именно благодаря совершенству дисциплины их маленькие армии могли успешно противостоять всем врагам».

солдаты будут управлять офицером. Они создадут хаос, который поможет им делать меньше, получать больше и не выполнять приказы. В конце концов подразделение проиграет битву, солдаты будут захвачены, а офицер казнен. И скажите, что он этого не заслуживает!

Как сказал Вэй Ляо-цзы в 200 году до н. э., «если они будут бояться нас, они будут презирать врага; если они боятся врага, они будут презирать нас»[1]. Это буквально означает, что в правильно организованной воинской части солдаты настолько боятся нарушить правила, что опасность врага — ничто по сравнению с этим. И это не потому, что их офицеры такие жестокие и свирепые, что хотят наказывать и издеваться. Отнюдь. Это потому, что их командиры понимают: выживание части и всей армии зависит от дисциплины, к которой они могут принудить. Чем больше порядка, тем выше шансы на выживание.

Дисциплина означает прежде всего четкую субординацию. Я, как солдат, всегда должен точно знать, кто мой командир, кому я подчиняюсь и кто может наказать меня, если я что-то делаю неправильно. Если я этого не знаю, я презираю командира, команду, компанию и инвесторов. Разве это не было бы справедливо, если бы я относился к ним как к очередной дойной корове?

Крис говорит, что я буду подчиняться Адриану и Тони одновременно. Чего они ожидают от меня после такого? Я могу презирать их, как те древние солдаты армии китайского императора презирали своего командира.

**Свобода и дисциплина не должны противоречить друг другу, но все же они, как правило, противоречат.**

Они боятся сказать Адриану, что я нахожусь в подчинении у Тони, потому что Адриан обидится, и я это понимаю. Он собеседовал меня и предполагал стать моим боссом. Теперь они должны сказать ему, что ситуация изменилась. И они даже не знают, как ему это объяснить. Почему

[1] ★ *Conner S. et al.* Military Strategy Classics of Ancient China. Special Edition Books, 2013, p. 222.

вдруг этот новый парень будет на равных с ним? Они не могут сказать, что этого хочет кандидат. Следующий ход, который может предпринять Адриан, — шантажировать их какой-то похожей идеей. Например, он может сказать, что хочет получить более высокую должность только потому, что ему это нравится, и они окажутся в довольно тяжелом положении — будет очень трудно ему отказать, если они только недавно согласились на мое требование.

С другой стороны, они боятся сказать мне, что я буду в подчинении лишь у Адриана, потому что не хотят потерять меня. У них недостаточно аргументов, чтобы объяснить мне, почему это невозможно. Например, у них нет четкой письменной иерархии ролей и обязанностей, где были бы очевидны моя позиция и сфера влияния. Они не знают, как управлять мною, как определить мои обязанности и ответственность, как очертить границы моих полномочий.

Вот почему они говорят: «Вы будете в подчинении у Адриана и Тони». Тем самым они просто сказали мне: «Вы можете делать все, что хотите, над вами не будет босса». Доволен ли я этим? Ну еще бы!

Выждав день, я принял предложение, мы подписали контракт и договорились, что я начну с понедельника.

Сегодня понедельник. Адриана еще нет. Ну, сейчас всего девять утра, и он начальник группы. Конечно, он появится позже. Одна из самых больших привилегий, которую вы получаете при продвижении на руководящую должность, — возможность появляться в офисе ближе к обеду.

Я обхожу офис, чтобы увидеть, кто есть кто. В нем все еще почти пусто, всего несколько человек. Они выглядят расслабленными. В офисе недавно был ремонт, на стенах висит парочка абстрактных картин. Недавно я узнал, что одна из самых дорогих картин в мире выполнена Джексоном Поллоком в его уникальном стиле капельной живописи. Название картины *«№ 5, 1948»*, она была продана в 2006 году за 140 миллио-

**Вполне естественно, что большинство работает усерднее и зарабатывает меньше, чем руководящее меньшинство.**

нов долларов[1]. А вот что действительно интересно — сам Поллок продал ее в 1949-м всего за 1500 долларов и погиб в автомобильной аварии семь лет спустя, когда ему было 44. Посмотрите эту картину — поймете, о чем я говорю.

Разве эта история не подтверждает мою прежнюю точку зрения о том, что в этом мире есть добытчики, а есть недобытчики? Есть те, кто делает работу (рисует), и те, кто крадет (перепродает)[2]. И количество, которое они крадут, намного больше, чем деньги, которые получают работающие люди. Несправедливо? Не знаю. Похоже, это задумано самой природой. Мы как животные призваны доминировать и воровать, если можем. Если мы достаточно сильны, то делаем это. С другой стороны, в стремлении помешать людям быть животными и постоянно бороться за добычу[3] была придумана мораль[4].

---

[1] *Vogel C.* A Pollock Is Sold, Possibly for a Record Price. The New York Times, November 2006: «Голливудский магнат индустрии развлечений Дэвид Геффен продал классическую капельную картину Джексона Поллока за 140 миллионов долларов. ...Эксперты арт-мира нашли покупателя — им оказался мексиканский финансист Дэвид Мартинес, который недавно приобрел двухэтажные апартаменты в южном здании Time Warner Centre за 54,7 миллиона долларов».

[2] *Маркс К., Энгельс Ф.* Манифест Коммунистической партии. — 1848: «Общество все более и более раскалывается на два больших враждебных лагеря, на два больших, стоящих друг против друга класса — буржуазию и пролетариат. ...В Древнем Риме мы встречаем патрициев, всадников, плебеев, рабов; в Средние века — феодальных господ, вассалов, цеховых мастеров, подмастерьев, крепостных, и к тому же почти в каждом из этих классов — еще особые градации».

[3] *Hirschi T.* Causes of Delinquency. University of California Press, 1969: «Мы все животные и, следовательно, естественным образом способны совершать преступные действия... курица, крадущая кукурузу у своей соседки, ничего не знает о моральном законе; она не хочет нарушать правила; она хочет просто съесть кукурузу... Никакой мотивации делать что-то неправильное не требуется, чтобы объяснить поступки. Таким образом, никакой особой мотивации к преступлению для человека... не требуется для объяснения его преступного деяния».

[4] Приписывается Фридриху Ницше, но я не нашел источника: «Мораль — это просто выдумка, используемая стадом посредственностей, для того чтобы сдерживать немногочисленных выдающихся личностей».

Нас учат, что воровство — это грех; похоже, вся Вселенная против воровства[1]. Однако (и к сожалению) это не так[2, 3]. Вселенная и Бог, если Он существует, всецело за — именно так они создали эту планету и нас.

Я не занимаюсь покупкой картин за 1500 долларов и последующей их перепродажей за 140 миллионов по одной простой причине: я на это не способен. Вот почему моя работа состоит в том, чтобы писать код на Java, сидя в этом уютном офисе и зарабатывая 100 баксов в день. Ну, если честно, это намного больше доходов миллионов людей, которые едва зарабатывают сотню баксов в месяц[4]. В отличие от крыс мы не разделены пополам на добытчиков и воров — только несколько процентов из нас можно отнести к ворам, в то время как большинство — добытчи-

---

[1] *Gert B.* Morality: Its Nature and Justification. Oxford University Press, 2005: «Из того, что моральные суждения могут быть сделаны в отношении всех рациональных людей, следует, что мораль универсальна и то, что представляется различными моральными системами, на деле является просто характеристиками или вариациями универсальной морали или моральной системы».

[2] *Moore G. E.* Principia Ethica. Cambridge University Press, 1922: «Если этика хочет утверждать, что некоторые способы поведения являются "долгом", то она этим хочет сказать, что вести себя таким-то и таким-то образом — значит создавать наибольшую возможную сумму добра. Предписание, обязывающее "не убивать", говорит нам, что ни один поступок, называемый убийством, ни при каких обстоятельствах не осуществит столько добра в универсуме, сколько его несовершение».

[3] *Joyce R.* The Evolution of Morality. The MIT Press, 2006: «Нас интересует гипотеза о том, что человеческая мораль является продуктом естественного отбора, а не экстравагантно неправдоподобная гипотеза о том, что любая мораль должна быть продуктом естественного отбора».

[4] *The World Bank.* Poverty Overview. October 2016: «Согласно последним оценкам, в 2013 году 10,7 % населения земли жили менее чем на 1,9 доллара в день... Половина крайне бедных людей живет в Африке, к югу от Сахары... Подавляющее большинство бедняков живут в сельской местности, они плохо образованны, в основном заняты в сельскохозяйственном секторе, более половины из них моложе 18 лет».

ки[1]. Кажется, что крысы, которые разделены поровну, гуманнее и этичнее, чем мы, не так ли?

В общем, я нахожу самый удобный стол, который пуст и правильно расположен (чтобы я сидел спиной к стене и никто не мог видеть экран моего монитора). Это очень важно — никто никогда не должен видеть его. Давайте назовем это конфиденциальностью. На самом деле это не имеет ничего общего с конфиденциальностью, это исключительно контроль. Я не хочу, чтобы кто-нибудь шпионил за мной, а также имел рычаги влияния на меня в организации; и я собираюсь сделать в этом офисе многое из того, что не имеет ничего общего с тем, за что они мне будут платить. Я буду работать над моими проектами с открытым исходным кодом и продолжу заниматься собственным стартапом. Можно ли мне так делать? Этично ли это? Что сказали бы крысы?

Теперь я проверяю свою электронную почту, а затем смотрю новый запрос на включение изменений в моем фреймворке с открытым исходным кодом. Иметь популярный проект с открытым исходным кодом очень интересно, но в то же время вызывает стресс. Вы можете найти немало статей от разработчиков открытого исходного кода, где они жалуются и оправдываются[2]. Неважно, что они говорят[3] и что я чувствую сейчас, сопровождая проект с открытым

---

1 *Treanor J.* Half of World's Wealth Now in Hands of 1% of Population. The Guardian, October 2015: «Это последнее доказательство того, что крайнее неравенство вышло из-под контроля. Действительно ли мы можем счастливо жить в мире, где 1 % наиболее богатых людей владеет половиной богатства, а самая бедная половина владеет всего лишь 1 %?»

2 *Lawson N.* What It Feels Like to Be an Open-Source Maintainer. https://goo.gl/6D7BAs: «Вы не хотите создавать новые проекты, так как знаете, что это только увеличит ваши затраты на обслуживание. На самом деле это порочный круг: чем успешнее вы будете, тем больше вас будут заваливать нотификациями GitHub».

3 *Fogel K.* Producing Open Source Software: How to Run a Successful Free Software Project. O'Reilly Media, 2006: «Большинство проектов бесплатного программного обеспечения оказываются провальными».

исходным кодом; я искренне верю, что у каждого серьезного разработчика программного обеспечения должен быть собственный open-source-проект (или несколько).

Двадцать лет назад, когда я был ребенком, в мире мало кто знал, что такое открытый исходный код, поскольку большая часть программного обеспечения создавалась для частного использования и для конкретного оборудования[1]. Кроме того, практически не было Интернета и инструментов, позволяющих поделиться кодом[2]. В настоящее время ситуация совершенно иная[3]. Возьмите любой коммерческий программный продукт, присутствующий на рынке, — будь то мобильное приложение или сайт — и проанализируйте его код. Большая часть будет исходить из его зависимостей — библиотек и фреймворков, — и только небольшая часть будет написана его авторами[4].

Мы живем в мире повторного использования программного обеспечения. Количество кода, создаваемого компаниями для достижения своих бизнес-целей, растет, а процент закрытого кода уменьшается[5]. Что это значит для нас, программистов? Одно: нам

---

1 *West J.* How Open is Open Enough?: Melding Proprietary and Open Source Platform Strategies. Research Policy, Volume 32, Issue 7, 2003: «Компьютерные платформы предоставляют интегрированную архитектуру стандартов аппаратного и программного обеспечения в качестве основы для разработки дополнительных ресурсов. Наиболее успешные платформы принадлежали частным спонсорам, которые контролировали их эволюцию и назначали соответствующие вознаграждения».

2 *Tozzi C.* For Fun and Profit: A History of the Free and Open Source Software Revolution (History of Computing). The MIT Press, 2017.

3 *Sandred J.*, Managing Open Source Projects: A Wiley Tech Brief. Wiley Computer Publishing, 2001: «Поверхностного взгляда достаточно, чтобы увидеть, что программное обеспечение с открытым исходным кодом есть везде. Для меня ясно, что все будущие сетевые информационные приложения будут основаны на технологии с открытым исходным кодом».

4 *Frakes W. B. et al.* Software Reuse Research: Status and Future. IEEE Transactions on Software Engineering, Volume 31, Number 7, 2005: «Большинство программных систем не новы. Скорее это варианты систем, созданных прежде».

5 *Golden B.* Why Enterprises Embrace Open Source. CIO, June 2015: «В корпоративных ИТ заметна возрастающая приверженность продуктам с открытым исходным кодом; они более предпочтительны, нежели их проприетарные альтернативы».

нужно знать, что такое открытый исходный код и как с ним работать[1]. Мы больше не можем сказать, что «моя компания платит мне за написание кода для нее, поэтому я не занимаюсь открытым исходным кодом». Это утверждение все менее логично, поскольку сейчас невозможно создать какую-либо часть программного обеспечения, не взяв что-либо из Интернета совершенно бесплатно.

Открытый исходный код вы не сможете изучить и понять за несколько дней. Это дикая территория со своими собственными правилами, принципами, привычками и установившимися практиками. Это целый мир[2], полный рисков, врагов[3], баталий[4], неудач и разочарований[5]. Вы не можете просто войти в него

**Невозможно быть успешным разработчиком программного обеспечения, не будучи активным пользователем продуктов с открытым исходным кодом.**

---

1 *Androutsellis-Theotokis S. et al.* Open Source Software: A Survey from 10,000 Feet. Foundations and Trends in Technology, Information and Operations Management, Volume 4, Numbers 3–4, 2010: «Программное обеспечение с открытым исходным кодом повлияло на весь глобальный рынок программного обеспечения. Это затронуло вопросы монополии, конкуренции и представленности на рынке».

2 *Dagenais B.* Moving into a New Software Project Landscape. Proceedings of the 32nd International Conference on Software Engineering (ICSE), IEEE/ACM, 2010: «Новички — это исследователи, которые должны ориентироваться в незнакомой среде. По мере накопления опыта они в конечном итоге занимают свою нишу. Как и исследователи природы, они могут испытать культурный шок, заблудиться без посторонней помощи или столкнуться с иными препятствиями».

3 *Coelho J. et al.* Why Modern Open Source Projects Fail. Proceedings of ESEC/FSE'17, 2017: «Наиболее распространенная причина неудач проекта — появление более сильного конкурента с открытым исходным кодом».

4 *Zhou M.* Who Will Stay in the FLOSS Community? Modeling Participant's Initial Behavior. IEEE Transactions on Software Engineering, Volume 41, Number 1, 2015: «Начало участия в проекте FLOSS сопряжено с трудностями, поскольку новые участники могут не иметь опыта и не знать норм проекта».

5 *Lee A.* Understanding the Impressions, Motivations, and Barriers of One Time Code Contributors to FLOSS Projects: A Survey. Proceedings of the 39th International Conference on Software Engineering (ICSE), IEEE/ACM, 2017: «Поначалу участие в проекте FLOSS может испугать вас. Разработчики должны понимать процесс подачи, уметь интерпретировать полуавтоматические сообщения об отклонении и справляться с другими задачами».

завтра и объявить себя профи. Никто не будет вас слушать, вы не получите никакой помощи, вы не будете знать, куда обращаться за информацией, вы просто потеряетесь[1].

Открытый исходный код — это инструмент, с которым должен уметь работать каждый профессиональный программист. Поэтому вам нужно сделать только одно: использовать его каждый день. И лучшая причина для этого — быть автором собственного продукта с открытым исходным кодом, который ценится сообществом[2]. Одно дело — быть пользователем, а вот быть автором — совершенно другая история. Если вы никогда не пробовали этого, вы не знаете мира открытого исходного кода.

Я до сих пор помню, как несколько лет назад обнародовал одну из своих Java-библиотек и вскоре обнаружил, что некоторые программисты до сих пор используют Windows в качестве своей платформы для разработки. Они начали сообщать об ошибках, жалуясь на то, что библиотека не компилировалась, не проходила модульные тесты и не работала на их ноутбуках. Сначала я был удивлен, так как на моей машине все было в порядке, но потом понял, что они используют Windows. Мне пришлось внести кучу изменений в код и в модульные тесты, чтобы сделать их совместимыми с этой умирающей (или уже мертвой) операционной системой. Вот так я познакомился с открытым исходным кодом. Кто бы еще сказал мне правду честно и откровенно?

---

[1] *Steinmacher I.* The Hard Life of Open Source Software Project Newcomers. Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE, ACM, 2014: «При внедрении проекта с открытым исходным кодом участники сталкиваются с множеством различных препятствий, которые усложняют их вклад и во многих случаях приводят к отсеву».

[2] *Lakhani K. R.* Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects. MIT Sloan Working Paper No. 4425-03, 2003: «Подавляющее большинство (> 61 %) заявили, что их основной проект FLOSS был по меньшей мере таким же творческим, как и все, что они делали в своей жизни».

# Утренние совещания

Адриан уже в офисе и звонит мне с предложением присоединиться к их очередному совещанию. Это ритуал, о котором многие команды разработчиков узнали из современного движения Agile/Scrum[1], не понимая, как и для чего он проводится. Честно признаться, этого и я не знаю. Я имею в виду, что не знаю, как это делать правильно. Если, конечно, есть способ делать это правильно.

Нас 12. Мы становимся в круг, стоим, смотрим в центр. Излишне говорить, что я чувствую себя немного неловко, потому что все глазеют на меня.

— Доброе утро всем. Познакомьтесь с новым членом нашей команды, он присоединился к нам сегодня, — начинает Адриан.

Все говорят: «Привет», я отвечаю более формально: «Здравствуйте».

— Обычный раунд сделаем, я начну, — берет инициативу Адриан. — Я все еще работаю над ошибкой пула соединения с базой данных, надеюсь исправить ее сегодня.

**Совещания по проверке статуса — отличный инструмент для синхронизации команды при слабом менеджменте.**

Адриан быстро заканчивает предложение, засовывая руки в карманы своих синих джинсов размера XXL и уставившись на свои ботинки.

— Я все еще реализую интерфейс получения XML. Мне понадобится еще два дня, потому что есть некоторые проблемы с модульным тестированием, но с ними я справлюсь, — говорит высокий парень в белой футболке.

— Хорошо, пожалуйста, держи меня в курсе, нам это нужно как можно скорее, — Адриан играет в большого босса. Совершенно

---

[1] *Sutherland J. et al.* Scrum: The Art of Doing Twice the Work in Half the Time. Currency, 2014.

очевидно, что эти слова никак не помогут, высокий парень закончит тогда, когда сможет, и не раньше.

— Само собой, — отвечает он серьезно.

Я уверен, что он улыбается самому себе, выслушивая подобные «приказы» Адриана на протяжении нескольких месяцев. Адриан смотрит на следующего парня в очереди.

— Я помогаю Тому решить проблему с производительностью базы данных, мы уже заканчиваем, но, думаю, понадобится поработать над этим сегодня и завтра, — сообщает парень.

— Том, ты все еще не определил, дело в запросе или в Java? — интересуется Адриан.

Том, стоящий рядом со мной, говорит:

— Скорее всего, это Java, поскольку все индексы на месте. Мы проверили вчера.

— Странно... Ладно, позже обсудим, — говорит Адриан.

Следующая жертва — толстый чувак в очках.

— Я все еще решаю вчерашнюю проблему со скриптом.

Он смотрит прямо на Адриана и явно нервничает:

— Он работает... извиняюсь, не работает... Я исправлю это сегодня, уже близок к завершению.

— Какой скрипт? — Адриан выглядит раздраженным.

Чувствую, чувака он явно недолюбливает.

— Преобразователь XML... Я объяснил тебе в пятницу... там ошибка, связанная с пустыми значениями.

— Но я же говорил тебе, ты не должен над этим работать, для нас это сейчас неважно! — Адриан злится и начинает объяснять, почему это неважно.

Терпеливо наблюдая за происходящим, я думаю: зачем мне на это тратить свое время? Зачем мне слушать этот информационный шум? Эта информация для меня абсолютно не имеет значения. Единственный человек, которому это нужно, — Адриан, поскольку он руководитель этих программистов. Если ему нужно знать, с чем они работают, он вполне может спросить их и в частном порядке. Или, что еще лучше, просто попросить их обновить статус тикетов,

с которыми они работают[1]. Почему этот высокий парень должен знать, какую ошибку Том исправляет? Зачем мне знать, что толстый чувак исправляет что-то неважное? Какая мне разница?

Представим, что я очень лоялен и действительно забочусь об этой компании. Даже в таком случае зачем мне знать, чем занят Том? Ну, разве что я буду работать с ним. Но в этом случае я просто спрошу Тома. И не утром перед всеми, а тогда, когда это мне понадобится. Почему 12 человек должны тратить свое время, слушая мои истории?

> **Совещания влияют на наши эмоции и призваны сплотить группу.**

Единственное разумное объяснение, которое у меня есть, состоит в том, что группе нужен этот ритуал, чтобы оставаться единой командой. Совещание не предназначено для обмена информацией — мы ее не понимаем, и большая ее часть нам неинтересна в любом случае — оно предназначено для обмена эмоциями. Мы все смотрим друг на друга, что-то говорим о наших вчерашних результатах и планах на сегодня, объясняем наши ошибки, улыбаемся и смеемся, но ключевая эмоция — вина. Это то, что нас объединяет[2]. Это своего рода клей[3].

---

[1] *McConnell S.* Rapid Development: Taming Wild Software Schedules. Microsoft Press, 1996: «Отслеживание — это фундаментальная деятельность по управлению при разработке программного обеспечения. Если вы не отслеживаете проект, вы не можете управлять им. Вы не сможете увидеть, осуществляются ли ваши планы и что вам следует делать дальше. Вы не сможете отслеживать риски для вашего проекта. Эффективное отслеживание позволяет обнаруживать проблемы на ранней стадии, пока есть время все исправить. Если вы не отслеживаете проект, вы не можете заниматься быстрой разработкой».

[2] *Baumeister R. F. et al.* Guilt: An Interpersonal Approach. Psychological Bulletin, Volume 115 (2), 1994: «Чувство вины может быть интерпретировано в контексте отношений как фактор, который укрепляет социальные связи, вызывая символическое подтверждение заботы и приверженности».

[3] *Scheff T. J.* The Emotional/Relational World: Shame and the Social Bond. Handbooks of Sociology and Social Research, Springer, 2001: «Если стыд и социальная связь являются ключевыми компонентами социальной интеграции, то признание стыда может быть тем клеем, который скрепляет отношения в обществе, а непризнание стыда — силой, которая их разделяет».

Эмпатия и страх — два основных источника вины[1]. Во-первых, нам не нравится, когда другие страдают[2]; во-вторых, мы сами боимся быть отвергнуты группой[3]. Сочетание этих двух психологических стимулов мотивирует нас оставаться лояльными и вносить свой вклад[4]. Если бы мы работали изолированно, имея перед собой только какие-то задачи, то что заставляло бы нас не лениться? Наши планы, повестки дня, расписание, тикеты, требования? Я сомневаюсь, что они работали бы так же сильно, как и ответственность (читай: чувство вины), которую мы чувствуем перед командой[5]. Во-первых, мы не хотим подводить коллег, например не уделяя достаточно внимания некоторым задачам, или игнорируя определенные проблемы, или просто срезая углы. Во-вторых, мы не хотим, чтобы они плохо о нас подумали и по-

---

[1] См. сноску 2 на с. 47: «Мы предлагаем два источника: эмпатическое возбуждение и беспокойство по поводу социальной изоляции. Оба они являются важными, мощными источниками аффекта и мотивации в близких, общинных отношениях».

[2] *Hoffman M. L.* Development of Prosocial Motivation: Empathy and Guilt. The Development of Prosocial Behavior, Academic Press, 1982: «Это интуитивно очевидно, и есть немало свидетельств тому, что большинство людей с раннего возраста чувствуют вину после причинения вреда кому-либо».

[3] *Baumeister R. F. et al.* Anxiety and Social Exclusion. Journal of Social and Clinical Psychology, Volume 9, 1990: «Тревога рассматривается как распространенная и, возможно, изначальная форма дистресса, возникающая в ответ на фактическое или потенциальное исключение из важных социальных групп».

[4] *Tangney J. P.* Moral Affect: The Good, the Bad and the Ugly. Journal of Personality and Social Psychology, Volume 61, Number 4, 1991: «Эмпатия, ориентированная на других людей, широко или узко определенная, как правило, рассматривается как "хороший" моральный аффективный навык или опыт, потому что предполагается, что она способствует теплым и близким межличностным отношениям, альтруистическому и просоциальному поведению, препятствует межличностной агрессии».

[5] *Bohns V. K.* Guilt by Design: Structuring Organizations to Elicit Guilt as an Affective Reaction to Failure. Organization Science, Volume 24 (4), 2012: «Мы полагаем, что некоторые функции на работе должны быть улучшены не во избежание негативного воздействия, а для продвижения определенной формы негативного аффекта, которая имеет тенденцию быть конструктивной (вина)».

тенциально отвергли нас. Похоже на то, что у этих двух причин один и тот же корень[1].

Он начинает развиваться с момента нашего рождения или даже раньше. Пока мы маленькие и уязвимые, наши матери, родители, учителя и другие авторитетные люди обеспечивают нас ресурсами, дают защиту. Поскольку наша главная задача как живых организмов — остаться в живых, мы не можем позволить себе отторжения — это немедленно означало бы приостановку поставки ресурсов, а затем смерть. Мы должны быть с ними, любить их и делать счастливыми, чтобы остаться в живых. Позже, когда мы вырастаем[2], возникает то, что психологи называют эмпатией. Они утверждают, что это врожденное свойство[3]. Однако, похоже, это просто искусственная производная[4, 5]

---

[1] *Novembre G.* Empathy for Social Exclusion Involves the Sensory-Discriminative Component of Pain: a Within-subject fMRI Study. Social Cognitive and Affective Neuroscience, Volume 10 (2), 2015: «Опыт социального отторжения может активизировать те участки мозга, которые активизируются при переживании физической боли... Этот паттерн активизации мозга распространяется и на случаи наблюдения за социальной болью этого же типа у других».

[2] *Roth-Hanania R.* Empathy Development From 8 to 16 Months: Early Signs of Concern for Others. Infant Behavior and Development, Volume 34, Issue 3, 2011: «Невысокий уровень аффективной и когнитивной эмпатии к другим людям, находящимся в дистрессе, уже был очевиден до двух лет и постепенно увеличивается (не всегда значительно) по мере достижения двух лет. Просоциальное поведение было редким в первый год и существенно возросло в течение второго года».

[3] *Howe D.* Empathy: What It Is and Why It Matters. Palgrave Macmillan, 2013: «Поэтому у эмпатических реакций и просоциального поведения действительно есть сильный генетический компонент. Эти врожденные различия можно обнаружить даже у маленьких детей, некоторые из них постоянно демонстрируют просоциальное поведение, сохраняемое на протяжении всей жизни».

[4] *Karyagina T.* Empathy Development: Natural or Cultural? Promises, Pedagogy and Pitfalls: Empathy's Potential for Healing and Harm. Inter-Disciplinary Press, 2016: «Итак, эмпатия — естественное или культурное явление? Мой ответ таков: культурное. Естественной является его основа на уровне мозговой деятельности».

[5] *Knafo A. et al.* The Developmental Origins of a Disposition Toward Empathy: Genetic and Environmental Contributions. Emotion, Volume 8, Issue 6, 2008: «Эмпатия была связана с просоциальным поведением, и эта связь в основном обусловлена воздействием окружающей среды».

инстинкта выживания — разочарование других не идет нам на пользу[1].

Родители должны научить нас вести себя так, как принято в обществе. Если они этого не делают, общество наказывает их, а не нас, детей. Чтобы упростить процесс обучения, были изобретены абстрактные категории, такие как вина и стыд[2]. Если маленький ребенок выплюнул еду на пол, мать повышает голос, и ребенок понимает, что уже недалеко и до смерти — в следующий момент она может бросить его. Тем не менее голос, как примитивный сигнал, работает только с маленькими детьми. Повзрослев, ребенок понимает, что еда на полу не приведет к отвержению или смерти. Чувство вины как эмоция более высокого уровня помогает ему связать свои детские страхи с последующими взрослыми ошибками: каждый раз, когда он плюет на пол, его мать повышает голос и говорит: «Не делай этого, это плохо» (или более агрессивно: «Ты плохой мальчик!»).

**Наше выученное стремление делать правильные вещи намного сильнее любой другой внешней мотивации, которую может предложить бизнес.**

Таким образом, ребенок обучен тому, что быть плохим или делать что-то плохое, разочаровывающее его мать, означает быть виноватым. Позже, когда он встает перед командой на утреннем совещании, он подсознательно вспоминает, что результатом вины и разочарования этих людей может быть отчуждение, повышенный голос его матери и потенциальная смерть. Его внутренний контроль, основанный на чувстве вины, развивается[3].

---

[1] *Allott R.* Evolutionary Aspects of Love and Empathy. Journal of Social and Evolutionary Systems, Volume 15 (4), 1992: «Эмпатия явно повышает приспособленность, имеет значение для выживания, поскольку служит способом общения между членами семьи, группы или даже между враждебно настроенными людьми или группами».

[2] *Delumeau J.* Sin and Fear: The Emergence of the Western Guilt Culture, 13th–18th Centuries. Palgrave Macmillan, 1990.

[3] *Nye F. I.* Family Relationships and Delinquent Behavior. Greenwood Press, 1973.

Для этого бедного толстяка Адриан — «фигура матери». Адриану даже не надо повышать голос, не надо наказывать или угрожать. Все, что ему нужно сделать, — это сказать, что он разочарован. Чувство вины толстяка сделает все остальное. Это свяжет небольшую ошибку в коде Java с врожденным страхом смерти.

Таким образом, наблюдение за тем, как вся группа становится каждое утро в круг, вызывает у нас чувство вины и помогает нам помнить, что нельзя вредить проекту, разочаровывать начальника, быть эгоистами, что мы не можем свободно делать то, что хотим. Мы чувствуем себя частью группы, и группа может отвергнуть нас, если мы пойдем против нее.

Вот почему мне не нравятся эти совещания. Но хорошая новость заключается в том, что я полностью осознаю, что происходит, и не даю развиваться этому чувству вины. Я просто говорю себе: «Ты не ребенок, вокруг нет ни матери, ни учителя. Ты работаешь здесь не потому, что боишься быть виновным; ты здесь потому, что тебе платят. Убедись, что ты делаешь то, что требуется, чтобы получать больше, вот и все». Иными словами, я всегда стараюсь работать из-за жадности, а не из-за вины. Я считаю, что эгоистичные материальные потребности мотивируют гораздо лучше.

Плохая новость в том, что завтра на этом шоу придется выступать и мне. И 11 человек будут смотреть на меня и очень внимательно слушать. Хотя они ничего не поймут, но постараются выглядеть абсолютно серьезными и сосредоточенными. Я буду смущаться, как они сейчас; они мне отплатят тем же. Вот для чего это шоу: чтобы все мы чувствовали смущение и вину из-за неспособности сказать: «Я сделал все, что вы сказали, готов выполнить ваши следующие задания, сэр!»

# Хаос в отслеживании задач

Кремниевая долина — небольшая территория на западе Соединенных Штатов. Она действительно маленькая, примерно 50 на 10 миль, с населением менее четырех миллионов человек. На севере — Сан-Франциско (SF), на востоке — Сан-Хосе (SJ); большая автострада № 101 соединяет эти два города. Поездка от SF до SJ займет около 50 минут, если нет пробок, то есть, как правило, ночью. В другое время на это может уйти два часа.

Это действительно коттеджный поселок. Даже SF, один из крупнейших городов США, в основном состоит из 2–3-этажных таунхаусов. Остальная часть долины выглядит так, будто на дворе 1950-е. Роскошные автомобили, припаркованные перед некоторыми домами, — единственное, что напоминает вам, в каком вы веке. Смотрели сериал «Кремниевая долина» (в некоторых источниках «Силиконовая долина». — *Примеч. ред.*)? Забудьте все, что вы там видели! Это абсолютная чушь, долина выглядит совсем не так.

Это место скучное, примитивное, некрасивое, депрессивное, поверхностное, дискриминационное, расистское и... богатое. Вы можете здесь успешно делать лишь деньги. Если вам нужны радость, веселье, эмоции, искренность, сочувствие, дружба или любовь, ищите их в другом месте.

**Политкорректность — это болезнь, быстро поражающая технологическую индустрию.**

Самая большая проблема, конечно, безумие политкорректности (и с каждым годом оно растет[1]). Сама идея политкорректности выглядит более чем благородно — защищать тех, кто не может защитить себя от оскорблений[2]. Однако не так давно это превратилось в охоту на ведьм: все, что вы говорите или создаете[3], может рас-

---

[1] *Henriques G.* Political Correctness is All about Slave Morality. Psychology Today, https://goo.gl/vUVafR, April 2016: «В последнее десятилетие я все больше беспокоюсь о том, что политкорректность превращается в притесняющую праведность, которая во многих отношениях глубоко неправильна; безусловно, нужно будет от нее отказаться, когда она полностью превратится в абсурд».

[2] *Hughes G.* Political Correctness: a History of Semantics and Culture. Wiley, 2010.

[3] *Kimball R.* The Rape of the Masters: How Political Correctness Sabotages Art. Encounter Books, 2005.

сматриваться как оскорбление и вас могут обвинить в политической некорректности[1]. Вы не можете думать, шутить и даже заниматься научными исследованиями о мужчинах и женщинах[2], черных и белых, умных и глупых, американцах и мексиканцах, красивых и некрасивых. Лучше не думать и не шутить вообще, тогда вы будете в безопасности.

Ситуация очень похожа на времена СССР и нацистской Германии[3], когда обе империи заявляли о свободе, но в реальности свобода была только у тех, кто соглашался с основным курсом партии. В Кремниевой долине вас не посадят в тюрьму и не расстреляют за инакомыслие, как это делали Сталин и Гитлер; вас подвергнут остракизму, уволив[4] и публично пристыдив. Политкорректность стала очень эффективным инструментом обратного запугивания: бывшие жертвы превращаются в обидчиков[5]. Я часто спрашиваю себя, так ли это нужно технологическому сектору, чтобы быть инновационным?

Адриан прерывает мои мысли. Черт, это всего лишь второй день, а он уже чего-то хочет от меня. Он определенно чувствует себя ответственным за то, чтобы дать мне работу. Я готов, но мне страшно. Я должен дать отпор, иначе стану обычным программистом и моя жизнь в последующий год (или около того) в этом офисе будет кошмаром.

---

1 *Altman S.* E Pur Si Muove. https://goo.gl/sJAXYs: «Кажется, в Сан-Франциско легче говорить какую-то ересь. Обсуждать какую-либо спорную идею, даже если у вас единодушие с 95 %, — опрометчивая идея».

2 *Rind B.* Archives of Sexual Behavior. Volume 37, 2008: «Исследования, выходящие за рамки навязанных культурных ценностей и идеологических норм, которые к тому же ставят под сомнение доминирующие в обществе конструкты сексуальности, могут быть подвергнуты острой критике за фанатизм, оскорбительный характер и псевдонаучность».

3 *Lessing D.* Sunday Times, May 10, 1992: «Политкорректность — естественное продолжение партийной линии. То, что мы снова видим, — это самозваная группа дружинников, навязывающая свои взгляды другим. Это наследие коммунизма, но, похоже, они этого не видят».

4 *Tiku N.* James Damore's Lawsuit is Designed to Embarrass Google. Wired, January 2018.

5 *Green D. G.* We're (Nearly) All Victims Now! How Political Correctness is Undermining Our Liberal Culture: «Группы, которые были на политическом уровне признаны жертвами, начинают использовать свою власть, чтобы заставить замолчать людей, осмелившихся критиковать их».

— Есть минутка? — начинает он вежливо.

— Конечно, что у тебя? — громко отвечаю я, делая вид, что готов выполнить все, что нужно ему, компании, стране и человечеству.

**Сильный менеджмент прежде всего означает готовность признать, что принуждение неизбежно.**

Я должен продемонстрировать свою готовность — это делает людей счастливыми. На самом деле я бы сказал, что чем выше у человека показушная лояльность и готовность помочь, тем выше риск предательства. Не доверяйте тому, кто говорит, что заинтересован в том, чтобы помочь вам, работать с вами, посвятить свое время вашему проекту, быть преданным делу. Они лгут, и вы не можете на них положиться. Доверяйте тем, кто говорит, что ненавидит работать над этим тупым проектом, что он здесь просто из-за денег, что его достали эта база исходного кода, эта команда, этот проект и этот язык программирования. Эти люди говорят вам правду, и вы действительно сможете положиться на них, когда все пойдет наперекосяк.

За популярное в наше время позитивное мышление[1] люди прячутся[2], когда не могут или не хотят говорить правду[3]. Если я соберу

---

[1] *Ehrenreich B.* Bright-Sided: How the Relentless Promotion of Positive Thinking Has Undermined America. Metropolitan Books, 2009: «"Хороший" командный игрок — "позитивный человек" по определению. Он часто улыбается, не жалуется, не слишком критичен и подчиняется всему, что требует начальство».

[2] *Burkeman O.* The Antidote: Happiness for People Who Can't Stand Positive Thinking. Text Publishing, 2012: «Попытки почувствовать себя счастливым часто делают нас несчастными. И именно наши постоянные усилия по устранению негатива — незащищенности, неуверенности, неуспешности или грусти — заставляют нас чувствовать себя такими неуверенными, тревожными, неуспешными или несчастными. ...Для того чтобы быть по-настоящему счастливыми, нам, возможно, нужно быть готовыми испытывать больше негативных эмоций или по крайней мере научиться перестать избегать их».

[3] *Oettingen G.* Rethinking Positive Thinking: Inside the New Science of Motivation: «Те люди, которые позитивно фантазируют о будущем (а это, вероятно, мы все), таким образом заводят себя в тупик. С одной стороны, они непреднамеренно расслабляются и обманывают себя, думая, что достигли своих желаний. Но при этом их мечты когнитивно связывают их с этими же желаниями, поддерживают их фантазии, игнорируя ту информацию, которая в ином случае могла бы побудить их посмотреть вокруг, получить представление о своих желаниях и, возможно, пойти другим путем. Слишком часто результатом становятся разочарование, неуспех и в крайних случаях глубокие переживания».

собственную команду, то буду искать только скептиков, даже если они могут выглядеть нарочито «пессимистично», зато лгать они не смогут или не захотят.

А я сейчас лгу Адриану, выражая свою готовность помочь. Если бы я говорил ему правду, то произнес бы: «Чувак, оставь меня пока в покое. Я в этом офисе всего лишь второй день. Дай мне хотя бы две недели свободного времени, чтобы я делал вид, что изучаю продукт. В любом случае никто не заметит». Однако Адриан недостаточно догадливый, чтобы услышать эту правду, и заставляет меня делать то, что ему нужно.

Он объясняет:

— Есть API для платежей, и с ним ряд проблем. Возможно, ты мог бы исправить их. Я действительно не знаю, что не так. Может объяснить Бао. Это все на Java, поэтому я решил, что ты можешь помочь нам прямо сейчас.

Он ждет моего вопроса. Что я понял? Не так много. Есть какой-то Бао, есть какой-то Java-код с ошибками, и они ждут, что эти ошибки будут исправлены в ближайшее время. Они хотят, чтобы я отвечал за неисправленные ошибки, я должен быть готовым к тому, что стану козлом отпущения.

— Кто такой Бао? — начинаю с самого нейтрального вопроса.

— Он из группы платежей, я могу познакомить вас, — Адриан не двигается, не встает, не спешит меня с кем-то знакомить.

— Он написал этот код?

— Нет, другой человек. Он ушел в марте работать в Facebook. Он был хорошим разработчиком.

Интересно, что именно сделало его хорошим разработчиком? То, что его взяли в Facebook, или написанный им просто потрясающий код?

— Действительно? Он здесь долго проработал?

— Около года. Он был весь в себе, всегда сидел в наушниках. Я, в общем-то, и не скучаю по нему, — смеется Адриан.

Смеюсь и я. Но мой вопрос все еще висит в воздухе: почему он был хорошим разработчиком? Мне всегда интересно, что люди говорят обо мне, когда я ухожу. Даже программисты, такие как Адриан, которые должны объективно судить коллег по их результатам, на эти результаты не обращают особого внимания. Вместо этого они

оценивают других по очень смутным социальным поведенческим факторам — таким как «ушел работать в Google», «ездил в офис на велосипеде», «встречался с парнем» или что-то подобное. Я никогда не слышал, чтобы люди говорили, что есть разработчик, который «всегда писал модульные тесты» или «очень круто разбирался в UML». Я их не виню. Но получается, что ваши политические взгляды или чувство юмора гораздо важнее качества вашего кода. Ужас, правда?

**Ваши социальные достижения намного важнее качества вашего кода.**

— Как-то аж страшно, — отвечаю я. — Ты когда-либо видел этот код?

Я улыбаюсь, но мне действительно не по себе. Он сказал, что они ожидают, будто я «помогу им прямо сейчас» и исправлю что-то, что парень из Facebook насоздавал в прошлом году, сидя в наушниках. Да ни за что!

— Да, конечно, видел!

Адриан явно лжет.

— Хорошо, тогда дай мне доступ. И хотелось бы поговорить с Бао. Где он?

Я встаю, Адриан тоже встает, но гораздо медленнее.

— Он приходит позже, — огорченно отвечает Адриан. И это хорошая новость для меня — в этом офисе можно «приходить позже».

— Не проблема, давай я посмотрю код, сделаю все, что смогу, — бодро рапортую я и возвращаюсь к своему столу.

Спустя несколько минут от Адриана приходит письмо с логином и паролем в их внутренний репозиторий Git. Я захожу в репозиторий и открываю его в IntelliJ IDEA. Несколько сотен Java-файлов, код выглядит просто кошмарно, это один из фреймворков, которые я ненавижу. У меня нет желания даже прикасаться к ним. Ну-с, посмотрим, кто такой Бао и что он скажет. Адриан знакомит нас. Бао выглядит лет на 40. Самоуверенный, высокий, азиатской внешности. Нравится ли он мне? Пока еще не знаю, но он показался мне человеком с завышенным самомнением, хотя довольно дружелюбным.

— Есть проблема с многопоточностью в пуле соединений с базой данных. Просто исправь ее. Это должно быть несложно, — говорит

Бао, глядя мне в глаза. Теперь я знаю, откуда придут неприятности. В его словах столько неправильного, что я даже не знаю, с чего начать.

— Откуда ты знаешь? — я стараюсь сохранять спокойствие.

— Слушай, я работаю инженером программного обеспечения более 15 лет. Поверь мне, я знаю, о чем говорю, — он дружелюбно улыбается и объясняет мне, почему считает, что проблема именно в пуле соединений. Но какого черта он говорит мне, где проблема, вместо того, чтобы просто объяснить, что это такое? «Мужик, сохраняй спокойствие, не теряй самообладания», — говорю я сам себе.

— Тогда я многому у тебя научусь! — отвечаю я с улыбкой, но он не улавливает в моих словах сарказма.

— Слушай, всегда спрашивай меня, если чего-то не понимаешь. Я самый опытный программист в этой компании, — он больше не улыбается.

**Обмен знаниями, если он не структурирован и не формализован, неизбежно будет полон разочарований.**

Может, шутит? Вроде не похоже. Он действительно говорит на полном серьезе. Кажется, этот чувак больше заботится о своем положении в иерархии, чем о технической проблеме, решения которой ожидает от меня. Ему намного важнее видеть себя выше меня, чем делегировать задачу так, чтобы ее можно было выполнить эффективнее. Это не значит, что он не хочет, чтобы задание было выполнено, но он видит власть как ключевой инструмент для достижения этой цели. Сначала он должен попытаться стать выше меня и достичь с моей помощью своих целей, затем он будет давать мне указания. Что я должен делать? Бороться, что же еще! Но не в открытом бою, где я наверняка проиграю.

— Да, я слышал об этом, — я перестаю улыбаться и начинаю льстить ему самым безобразным образом.

— Отлично, — заглатывает наживку он, и я вижу, что он зациклен на самом себе.

— Каковы симптомы проблемы? Ты о них сообщал?

Я всегда ожидаю определенных формальностей между тем, кто ставит задачу, и ее исполнителем. В данной конкретной ситуации

это единственный способ отстраниться от этого сумасшедшего чувака. Я не должен позволять ему говорить мне, что делать. Он должен указать задачу, а я напишу код, который всех устроит. В противном случае, если я позволю ему рассматривать мою «проблему с многопоточностью» так, как он посчитает нужным, он будет исправлять мои ошибки, а я стану рабом или козлом отпущения. Я должен запретить любые обсуждения свойств нашего API и разрешить ему говорить только о насущной проблеме. Я считаю, что должен быть какой-то трекер ошибок, в котором сообщается об ошибке, и должен быть конкретный человек, сообщивший об ошибке. Возможно, это Бао, хотя, скорее всего, это не он. В его отделе работает более 20 программистов. Я уверен, что он никогда не пишет никакого кода и не может лично испытывать никаких проблем с API. Итак, вопрос в том, каковы требования?

— Слушай, просто проверь синхронизацию в пуле и исправь ее. Если ты разбираешься в Java и базах данных, для тебя это не проблема, — говорит он и отворачивается от меня, глядя в свой ноутбук. — Прости, здесь что-то срочное, дай мне знать, когда проблема с многопоточностью будет решена, проверим на нашей стороне.

Он читает несколько писем, а я смотрю на него со стороны.

— Хорошо, дай мне немного времени, — я поднимаюсь и медленно ухожу.

Что я чувствую? Меня как будто изнасиловали. Я возвращаюсь за свой стол и пытаюсь проанализировать, что сейчас произошло. У Бао достаточно власти в этой компании, потому что он глава отдела, работает здесь куда дольше, нежели я, и у него есть информация, которая нужна мне для решения проблемы (этого от меня ожидает мой начальник Адриан). Если я не решу ее по какой-либо причине, вина будет на мне. Бао это понимает. Если мне не удастся выведать информацию у Бао, я даже не смогу объяснить, что он саботировал процесс, ведь он скажет, что объяснил мне, что нужно исправить. Если я не буду знать, как это исправить, то буду выглядеть непрофессионалом. Этого хватит, чтобы Тони принял

организационные решения. Он меня уволит? Ну, не прямо сейчас, но моя репутация будет серьезно подпорчена.

Могу ли я пожаловаться Адриану? А на что жаловаться-то? Ну, я могу сказать, что Бао не объясняет, в чем проблема, по существу, а просто говорит мне, что делать. И это удивит Адриана? Не думаю. Они давно работают вместе, поэтому знают, как здесь все устроено. Они оба прекрасно осведомлены о хаосе, который позволил Бао сделать то, что он только что сделал. Да, конечно, глубинной причиной возникшей ситуации является отсутствие структуры управления.

**Строгая формальная система отслеживания ошибок предотвращает спекуляции с информацией.**

В самом деле, если бы об ошибке уже сообщалось где-то в системе отслеживания ошибок, у нее были бы автор, симптомы, история изменений, серьезность и приоритет, а также все остальные атрибуты, которые должны быть у правильного отчета об ошибке, как учил меня доктор Майерс в своей книге[1], — и роль Бао была бы минимальной. Ему просто нечего было бы сказать, потому что он не владел бы нужной информацией. Все было бы в свободном доступе, в истории тикетов. Я бы просто открыл тикет, прочитал, решил проблему с многопоточностью (если это действительно так и Бао не приврал) и закрыл его. Конечно, такая прозрачность могла бы серьезно повлиять на позицию Бао, поскольку гораздо меньше людей приходило бы к нему за «советом», как он это называет, но на самом деле они просто приходят выразить свое уважение и продемонстрировать подчинение великому эксперту, который «знает все!»[2].

---

[1] ★ *Myers G. J. et al.* The Art of Software Testing. 3rd Edition, Wiley, 2011.

[2] ★ *Adams S.* The Dilbert Principle: a Cubicles Eye View of Bosses, Meetings, Management Fads & Other Workplace Afflictions. Harper Business, 1996: «Хороший способ для неэффективных людей цепляться за власть в организации — создавать монополию на информацию. Эта информация должна быть важной, но не критически важной. Другими словами, ваши коллеги должны хотеть получить информацию, которую вы скрываете, но не настолько сильно, чтобы они душили вас до смерти, когда вы ее не выдаете».

Погодите... Может быть, где-то есть система тикетов и Бао просто не сказал мне об этом?

— Мы где-нибудь отслеживаем проблемы? — спрашиваю я у Адриана и использую «мы» вместо «вы, идиоты», чтобы выглядеть более лояльным и вежливым.

— Да, конечно, служба поддержки регистрирует все ошибки, поступающие от пользователей, и публикует их в JIRA, но вот правильно описать... в общем, получается не лучшим образом.

— А как насчет технических ошибок внутри отделов и между ними?

— Нет, мы их не отслеживаем, потому что всегда легче рассказать об этом лично. Мы же в одном офисе!

Похоже, он этим гордится. Ну, я могу понять его. Он — часть этого хаоса и находится практически в таком же положении, как и Бао. Он не настолько без ума от власти, но ему определенно нравится слыть экспертом.

Я заметил, что в предыдущих компаниях, где я работал раньше, хаос всегда благоволит людям, которые остаются дольше. Обычно они не самые лучшие сотрудники, но самые стабильные и адаптирующиеся, как Адриан и Бао. Они накапливают знания о продукте и не хотят делиться ими с кем-либо просто так. Любые правила документирования или отслеживания ошибок и системы пугают их, потому что они представляют собой очевидную угрозу их работе. Чем больше они делятся, тем меньше чувствуют собственную важность.

**Чем выше цена информации в команде разработчиков программного обеспечения, тем менее эффективна команда.**

Давайте рассмотрим этот пример снова. Если бы проблема уже была описана в тикете, я бы не пошел к Бао, не знакомился бы с ним, не стал бы слушать его шизофренические истории о том, насколько он шарящий чувак. Он просто стал бы обычным кем-то, а не большим начальником и очень умным экспертом, который как никто другой может объяснить, в чем заключается проблема.

Нам, новичкам и молодым программистам, не нравится хаос, потому что он делает нас зависимыми от экспертов. Мы должны

выпрашивать информацию и чувствовать себя не в своей тарелке, если не понравимся Бао или Адриану.

Излишне говорить, что такая ситуация по многим причинам вредит компании. Во-первых, этим экспертам обычно переплачивают, и высшее руководство ничего не может с ними поделать. Тони не может легко уволить Бао, потому что только Бао знает, что такое «проблема многопоточности». Само наличие экспертов переворачивает менеджмент с ног на голову. Организацией управляет уже не Тони, а Бао. Во-вторых, присутствие этих хранителей информации или экспертов серьезно демотивирует тех, кто не хочет становиться экспертом или просить экспертов помочь и поделиться сведениями. Я один из таких. Я не хочу больше общаться с Бао. Я бы предпочел, чтобы он заболел или умер. Я очень зол, но если копнуть глубже, то вполне может быть, что он неплохой человек. Он просто защищает свою работу, свою зарплату, свою семью. Он может быть даже большей жертвой этого хаоса, нежели я.

А виноват в этом Тони. Он технический директор и отвечает за установление правил в своей организации. Он должен гарантировать, что поток информации хорошо разъяснен, формализован, задокументирован и легок для понимания. Ему этого сделать не удалось, поэтому команда создала свои собственные правила, которые в целом сводятся к «Бао знает, как это сделать». Такие правила контрпродуктивны и работают против Тони и организации. Я считаю, что виноват в этом в первую очередь он. Разве это не так?

Хорошо, что я знаю, что делать. Я должен жаловаться на беспорядок, а не на Бао. Я не буду спорить с Бао, потому что непременно проиграю, поскольку его позиция существенно сильнее моей. Однако я попробую переманить его на свою сторону вместе с Адрианом и внедрить правила отслеживания ошибок в нашей команде. Эти правила в итоге «уничтожат» их обоих: они больше не будут слыть экспертами, поскольку информация станет доступной и прозрачной. Система тикетов будет содержать сведения о программном обеспечении, ошибках, проблемах, рисках и решениях. Личное общение даже не потребуется, если все задокументировано официально, в письменном виде. Эти ребята лишатся своей прежней власти.

Я должен все сделать правильно, чтобы нейтрализовать их страх потерять работу. На сколько это возможно? Не знаю, но я постараюсь.

А тем временем что делать с этой задачей? Я думаю, следует найти другого козла отпущения, который будет готов заменить меня.

— Ты знаешь, кто лучше всех разбирается в проблемах многопоточности в нашем бэкенде? — спрашиваю у Адриана.

Вопрос довольно рискованный, поскольку может показаться, что я ничего не смыслю в многопоточности, а я не могу позволить себе подвергнуть сомнению мою компетентность в самом начале испытательного срока. Вот почему я подчеркиваю «в нашем бэкенде». Вопрос должен звучать так, будто я много знаю о многопоточности, но мне нужна помощь, чтобы понять проблему в данном конкретном случае.

— Поговори с Томом, он решил кое-какие проблемы месяц назад.

Адриан — наш парень, в отличие от Бао. Я нахожу Тома. Это молодой парень в очках Gucci и с несколькими слоями наклеек на своем макбуке. Ну-с, попробуем заставить его помочь мне и выполнить мою работу.

**Единственный способ избавиться от всезнающих экспертов — формализовать поток информации.**

— Привет, — улыбаюсь я, — ты мог бы мне помочь? Я не могу понять, с чего начать.

Я должен быть честным с парнем и объяснить ему все, чтобы он чувствовал себя ответственным за помощь другу.

— Бао сказал, что в нашем API для платежей есть определенные проблемы с многопоточностью, но по существу не объяснил, что не так. Ты не знаешь, о чем речь?

— Давай посмотрим, — Том сразу открывает новое окно IDEA и просматривает код. Он что-то говорит о каких-то классах, пытаясь объяснить мне, что где находится. Я не слушаю, но продолжаю кивать и повторять: «Ага». Все, что мне сейчас нужно, — чтобы он взял на себя ответственность за этот вопрос. Затем он отправится к Бао, и они что-нибудь придумают. Я уверен, что Бао уже рассматривает Тома как своего подчиненного и не будет играть в ту самую игру, которую затеял со мной.

— Как думаешь, это можно исправить? — спрашиваю я через десять минут.

— Да, я займусь этим, не волнуйся, — он себя прекрасно чувствует, потому что может помочь коллеге.

— Спасибо, чувак, ценю это, — и я направляюсь к своему столу.

Я возвращаюсь к Адриану.

— Знаешь, непросто понять, о чем говорит Бао! Сегодня мне пришлось снова обсуждать с ним эту проблему, хотя он уже все объяснил Тому несколько месяцев назад. Как насчет того, чтобы Бао и его программисты сообщали нам об ошибках только через систему тикетов?

Я произношу эти слова и понимаю, что это правильный ход, поскольку Бао — конкурент Адриана. Я помогу Адриану в этом бою, и он будет на моей стороне.

— Смотри, — продолжаю я, — он говорит программистам из нашего отдела, которые находятся только в твоем подчинении, что им делать. Он пытался сделать то же самое со мной. Вместо того чтобы правильно сформулировать проблему, он велел мне разбираться с ней самостоятельно, и только благодаря Тому мы справились.

Адриан слушает.

— Так не должно быть, — продолжаю я. — Нужно, чтобы все, что им не нравится в нашем коде и продуктах, проходило через систему тикетов. Давай сделаем так, чтобы у каждой проблемы и задачи был тикет. Кстати, я уверен, что Тони понравится эта инициатива, она дисциплинирует.

> Гибкость не может быть оправданием отсутствия правил и систем отслеживания задач.

Тони — его начальник, и Адриан определенно заинтересован в том, чтобы донести до него новые идеи.

— Мы пробовали это в прошлом году, но не сработало, — ответил Адриан без какой-либо заинтересованности.

Похоже, я не первый, кто предлагает это.

— Потому что никто не создает тикеты, а все просто решают вопросы лично. Может быть, мы не так дисциплинированны, как ты думал.

Адриан улыбается, и я понимаю, что самый недисциплинированный человек — прямо передо мной.

— А как ты знаешь, над чем сейчас работает команда? Как ты распределяешь задачи, как отслеживаешь результаты?

— Потому-то я и нахожусь в офисе весь день без выходных, — смеется он, очень гордясь этим. — Потому мы и проводим совещания каждое утро, чтобы все держать под контролем.

— Стало быть, официального отслеживания задач у нас нет, верно?

— Нет, мы Agile, — улыбается Адриан, и я не понимаю, сарказм это или он действительно думает, что Agile подразумевает отсутствие отслеживания задач.

Что еще я могу сказать? А ведь так работают в большинстве компаний. Этот бедный Адриан не одинок. Все-таки он не совсем руководитель. Он был программистом на протяжении нескольких лет, затем перешел на эту руководящую должность. Он почти ничего не знает об управлении проектами, но у него доброе сердце. Он просто не может подвести Тони. Он хороший человек, поэтому он начальник. Можно ли винить его в том, что он руководил командой без задач, отслеживания, правил, требований и дисциплины? Не думаю. Опять же виноват в этом Тони, который неправильно структурировал компанию.

Можно ли исправить эту ситуацию? Могу ли я что-то сделать, будучи архитектором или просто программистом? Очень сомневаюсь. Эта ситуация в этой конкретной компании неуникальна. Большинство команд разработчиков работают именно так и называют это Agile. У них вообще нет никакой дисциплины, никаких правил, планов, расписаний или формальностей. Каждое утро они просто становятся в круг вокруг руководителя, который раньше был программистом...

# Глава 2

# Деннис

Я работаю в этой компании уже почти два месяца. Как я и предполагал, здесь хаос не только в управлении, но и в коде и во всех технологических процессах. Хотя правильнее будет сказать *ad hoc (ситуативно)* — не беспорядочно или хаотично, как я думал первоначально. Не то чтобы тут царил полный беспорядок. Все-таки есть определенные правила, определенные соглашения, определенные привычки, но они созданы командой, а не руководством. Это может показаться не такой уж плохой идеей, но на самом деле это не так. Процесс, который задействован сейчас, мотивирован исключительно людьми, участвующими в нем. Например, Бао был автором сценария развертывания, поэтому он отвечает за развертывание продукта в производстве; Том использовал Hibernate на своей предыдущей работе, поэтому мы также используем Hibernate; Деннис не любит Бао, поэтому Деннис не работает над задачами, исходящими из отдела Бао; Бао не любит меня, поэтому он не приглашает меня на свои совещания, ну и так далее. Кто кем управляет?

## Автоматическое тестирование

Мы все делаем так, как делаем, потому что нам так удобнее или только так мы и можем. Деловые соображения стоят на втором месте, если они вообще есть. Тони, наш технический директор, не может сказать, чтобы мы что-то делали определенным образом, потому что так нужно нашим клиентам. Он способен разве что спросить нас, что мы можем сделать, а затем передать это клиентам. В большинстве случаев возвращается он ни с чем. Процесс не навязывается, он просто происходит. И всегда в нашу пользу. Нам, программистам, такой подход на руку, но компания страдает.

Деннис, программист из Греции, сидит рядом со мной. Он приехал в Калифорнию по рабочей визе шесть лет назад и за это время успел жениться, завести двоих детей и несколько раз поменять работу. Он хороший программист и очень эмоциональный парень. Ему где-то около 30. Похоже, что его не радует работа здесь, несколько раз он говорил мне, что собирается уходить. Он не рассказал почему, а я не спросил. Обычно лучше держаться подальше от тех, кто собирается уходить, если действовать стратегически. Но эти люди обычно наиболее честны, и вы можете многому у них научиться. Им практически нечего терять.

— Я исправил это две недели назад! Что происходит? — закричал Деннис и ударил кулаком по столу.

— Что случилось? — откликнулся я. Мне всегда интересно услышать историю от взбешенного программиста.

**Автоматизированное тестирование — это система безопасности, которая защищает программу от ее программистов.**

— Слушай, я потратил целый день, чтобы исправить эту тупую ошибку, и теперь она снова вернулась! — он разочарованно смотрит на меня. Что я могу сказать?

— Ты проверял эту функциональность модульным тестом? — спрашиваю.

— Что ты имеешь в виду, чувак?

— Ты знаешь, для чего нужны модульные тесты?[1] Ну, в общем, любые автоматические тесты, — я изображаю удивление, но на самом деле вовсе не удивлен. Я видел код и давно уже понял, что модульные тесты здесь скорее исключение, нежели правило. Они пишут их изредка и в основном для забавы[2]. Никто действительно не понимает истинной сути автоматизированного тестирования[3].

— А, ты имеешь в виду разработку через тестирование? Да, чувак, я знаю, для чего это!

— Нет, не то, забудь о TDD, просто тесты.

Опять же я не удивлен. Большинство людей путают их[4]. Автоматические тесты и разработка через тестирование — две разные вещи[5],

---

[1] *Shepard T. et al.* More Testing Should Be Taught. Communications of the ACM, Volume 44, Issue 6, 2001: «Тестирование обычно занимает не менее 50 % ресурсов для проектов разработки программного обеспечения. Любопытно, что в рамках типичной учебной программы для студентов, изучающих информационные технологии, на тестирование выделяется значительно меньше ресурсов».

[2] *Hunt A. et al.* Pragmatic Unit Testing in Java with JUnit. The Pragmatic Bookshelf, 2003: «Многие программисты считают, что тестирование — это своего рода неприятные хлопоты, отвлекающие от реального дела — клепания кода».

[3] ★ *Feathers M. C.* Working Effectively with Legacy Code. Prentice Hall, 2004: «Код без тестов — плохой код. Неважно, насколько хорошо он написан; не имеет значения, насколько он красив, объектно-ориентирован или инкапсулирован. С помощью тестов мы можем быстро и контролируемо изменить поведение нашего кода. Без них мы на самом деле и не знаем, становится наш код лучше или хуже».

[4] *Janzen D. et al.* Does Test-driven Development Really Improve Software Design Quality? IEEE Software, Volume 25, Number 2, 2008: «Заблуждение № 1: TDD = автоматизированное тестирование. ...Поскольку TDD помог продвинуть автоматизированное тестирование на передний план, многие, похоже, полагают, что TDD — это только написание автоматических тестов».

[5] *Janzen D. et al.* Test-Driven Development: Concepts, Taxonomy, and Future Direction. Computer, Volume 38, Number 9, 2005: «Автоматизированное тестирование включает в себя написание модульных тестов в виде кода и размещение этого кода в тестовом комплекте или фреймворке, таком как JUnit. Среды автоматизированного модульного тестирования предполагают минимум усилий по тестированию, сводя большое количество тестов к нажатию кнопки. ...С TDD программист пишет модульные тесты перед тестируемым кодом. В результате он может выполнить тесты сразу после их написания».

но они обычно рассматриваются в связке друг с другом, возможно, благодаря книгам[1].

— Но, чувак, как это связано с этой дурацкой ситуацией?

— Автоматизированные тесты защищают и тебя, и всех остальных от той самой ситуации, в которой ты сейчас находишься. Любую ошибку, которую ты исправляешь, или любую добавленную тобой функциональность ты должен покрыть тестом. Проще говоря, тебе нужно создать еще один фрагмент кода, который запускается и завершается неуспешно, если функциональность не работает или ошибка не исправлена.

Деннис слушает и смотрит в свой ноутбук.

— Если бы ты создал такой тест две недели назад, — продолжаю я, — никто не смог бы снова испортить этот код[2].

— Как так?

— Ну, они могли бы сделать это, но сборка не прошла бы. И в идеале они не смогли бы провести слияние своих изменений.

Мне кажется, что я говорю очевидные вещи[3], но не похоже, что он понимает меня.

— Что ты имеешь в виду, говоря, что не смог бы провести слияние? Как сборка связана со слиянием?

— В нашем случае они не связаны, но в идеальном мире изменения сливаются в ветку master только в том случае, когда они не нарушают сборку.

---

1 *Beck K.* Test-Driven Development by Example. Addison Wesley, 2003.

2 *Beck K.* Embracing Change With Extreme Programming. Computer, Volume 32, Number 10, 1999: «Вместо действий, которые улетучиваются сразу же после их завершения, вы пишете долговременные тесты. Эти тесты будут выполняться автоматически и сейчас, и вечером, и завтра, и на следующей неделе, и в следующем году. Уверенность, которую они подпитывают, накапливается, поэтому команда XP со временем все больше убеждается в правильном поведении своей системы».

3 *Williams L. et al.* Test-driven Development as a Defect-reduction Practice. 14th International Symposium on Software Reliability Engineering (ISSRE), 2003: «Новая функциональность считается реализованной должным образом только в том случае, если и новые, и все прежние модульные тестовые сценарии, написанные для кодовой базы, работают корректно».

— Сливаются кем?

— Должен быть скрипт, который делает это. Допустим, ты создал новую ветку, внес свои исправления вместе с тестом. Они не нарушают сборку, так как изменения и тест есть в твоей ветке. Ты убедился, что тест пройден. Затем мы запускаем скрипт, который сливает твою ветку с master и снова запускает так называемую pre-flight-сборку[1]. Если все чисто, скрипт отправляет изменения в master. Затем кто-то еще... Нет, не кто-то еще. Давай сделаем историю более реалистичной. Допустим, Том создает новую ветку и вносит какие-то изменения, нарушающие код, который ты только что исправил. Затем мы запускаем скрипт для ветки Тома. Скрипт сливает изменения Тома с master и запускает сборку. Твой тест, который ты добавил ранее, не прошел, потому что Том изменил код. Скрипт сообщает об этом Тому, и master остается нетронутым. Том должен убедиться, что его код соответствует твоему тесту, если он хочет, чтобы его код попал в master.

— Но Том может удалить мой тест.

— Это возможно. Но для этого и нужна проверка кода[2]. Прежде чем дать код скрипту, его нужно более или менее формально проверить, желательно двумя людьми. Если тест будет удален, рецензент заметит это и поинтересуется. Возможно, тебе следует зани-

---

[1] *Aiello B. et al.* Agile Application Lifecycle Management: Using DevOps to Drive Process Improvement. Pearson Education, 2016: «Предстартовые сборки позволяют разработчику запускать сборку в частном порядке на своем компьютере, чтобы убедиться, что код будет компилироваться на платформе сборки, прежде чем передавать его команде разработчиков. Предстартовые сборки экономят немало времени, выявляя аномалии без пинг-понга с забрасыванием сборки по сети — только для того, чтобы она была отклонена в случае сбоя».

[2] *Gousios G.* Work Practices and Challenges in Pull-Based Development: The Integrator's Perspective. Proceedings of the 37th International Conference on Software Engineering, IEEE Press, 2015: «Член основной команды проекта отвечает за проверку изменений и их интеграцию в основную ветку разработки проекта. Роль интегратора имеет решающее значение. Интегратор должен выступать в образе блюстителя качества проекта. ...Феномен качества проявляется в том, что интегратор в явном виде просит о проверке кода при запросе изменений, беспокоится о качестве на уровне исходного кода и наличии тестов».

маться такой проверкой, если изменения касаются созданного тобой кода[1].

— Что если Том просто внесет изменения в ветку master, не запустив сценарий?

— Мы должны сделать это технически невозможным с помощью конфигурации Git[2]. Никто не должен иметь возможность заливать изменения в master — только через сценарий.

**Ветка master должна быть доступна только для чтения всем, кроме машины, которая сливает все изменения.**

— Значит, чтобы что-то слить, каждый из нас должен запустить сценарий? Где будет установлен этот скрипт? Я не совсем понял, как это реализовать технически.

— В идеале это должен быть не простой bash-скрипт, а сервер или бот, установленный где-нибудь и доступный только через веб-интерфейс или что-то подобное. Ты должен не в буквальном смысле запускать скрипт, а просто нажимать где-то кнопку для этого.

Деннис пожимает плечами:

— В любом случае мы не пишем эти автоматизированные тесты, чувак[3].

— Это уже другая история. Конечно, не пишем, потому что нет мотивации. Даже если бы ты написал такой тест две недели назад, это все равно не помогло бы. Том, например, легко проигнорировал бы твой тест и протолкнул изменения в master. Не потому, что Том такой плохой, а потому, что скрипт не помешал бы ему это сделать.

---

1 *Bacchelli A. et al.* Expectations, Outcomes and Challenges of Modern Code Review. Proceedings of the 35th International Conference on Software Engineering (ICSE), IEEE Press, 2013: «Когда контекст ясен, а понимание [кода] такое же четкое, как если бы вы были владельцем измененных файлов, можно получить комментарии более подробные, действенные и уместные, а также обнаружить более труднонаходимые проблемы».

2 *Chacon S.* Pro Git. 2nd Edition, Apress, 2014: Customizing Git, An Example Git-Enforced Policy, Server-Side Hook.

3 *Beck K. et al.* Test Infected: Programmers Love Writing Tests. Java Report, Volume 3, Number 7, 1998: «Каждый программист знает, что он должен писать для своего кода тесты. Но пишут их немногие. Распространенный ответ на вопрос: “Почему не пишешь тесты?” — “Нет времени”».

— Такая система — дополнительная головная боль для программистов. Это никому не понравится, — делает вывод Деннис.

— Ну да, это головная боль. Когда система большая и уровень покрытия тестами достаточно высок, сложно вносить какие-либо серьезные изменения, не ломая чужие тесты.

— И ты думаешь, это хорошо? Кстати, что такое покрытие тестами?

**Повышая качество программного обеспечения, высокое покрытие тестами неизбежно замедляет саму разработку.**

— Это процент кода, затронутого модульными тестами, когда мы их запускаем[1, 2]. Допустим, у тебя есть десять классов Java и нет модульных тестов. Твое покрытие составляет 0 %. Затем ты создаешь свой первый тест, который вызывает отдельные методы в одном из этих классов и не касается других классов. Твое покрытие составляет 10 %. Затем ты создаешь еще один тест, который касается того же класса и никакого другого. Твое покрытие все еще составляет 10 %. В конечном итоге у тебя будет много тестов, которые затронут все десять классов, и твое покрытие составит 100 %[3].

---

[1] SWEBOK, глава 4: «Набор тестов создается исходя из покрытия всех условий и решений блок-схемы. В какой-то степени напоминает тесты на основе конечного автомата. Отличие — в источнике набора тестов. Максимальная отдача от тестов на основе блок-схемы наблюдается, когда тесты покрывают различные пути блок-схемы, по сути, сценарии потоков работ (поведения) тестируемой системы. Адекватность таких тестов оценивается как процент покрытия всех возможных путей блок-схемы».

[2] ISTQB, International Software Testing Qualification Board, version 2.0, 2007: «Покрытие показывает процент исходного кода программы, который был выполнен в процессе тестирования. Если охват не равен 100 %, то можно разработать больше тестов для проверки пропущенных элементов и, следовательно, для увеличения охвата».

[3] *Prause C. R. et al.* Is 100 % Test Coverage a Reasonable Requirement? Lessons Learned from a Space Software Project. Proceedings of the 18th International Conference on Product-Focused Software Process Improvement (PROFES), 2017: «100%-ный охват необычен, но достижим. ...100%-ное покрытие иногда необходимо. ...100%-ное покрытие приносит новые риски. ...Не оптимизируйте для 100%-ной метрики. ...100%-ное покрытие не является достаточным условием хорошего качества».

Обычно тестов больше, чем классов, которые они тестируют. И конечно, этот показатель касается не только классов. Он также учитывает строки кода, пути, методы и т. д.[1] — на самом деле объект не имеет значения. Ну, ты понял идею.

— Да-да, я понял, — он откидывается назад и кладет руки за голову. — Стало быть, какое у нас сейчас покрытие?

Деннис улыбается и смотрит в монитор.

— Около нуля, и это главная[2] проблема. Вот откуда твое расстройство. Без нормального тестового покрытия эта ситуация будет повторяться — и ты ровным счетом ничего не сможешь сделать.

— Я могу найти, кто испортил мой код, просто посмотрев на историю Git! — воскликнул он.

— Можешь, но что это тебе даст? Он скажет тебе, что исправил какую-то важную ошибку и случайно сломал то, что ты сделал. И что? Ему жаль. Однако это не помешает ему сломать твой код завтра. Он не делал этого специально. Это была случайность. Автоматизированные тесты были разработаны именно для предотвращения подобных ситуаций. Их иногда сравнивают с защитной сеткой, которую электрики протягивают над дорогой, когда устанавливают высоковольтные кабели. Если они случайно уронят кабель, он не упадет на землю и ничего не повредит — сетка поймает его. Модульные тесты выполняют точно такую же работу — ловят, когда ты случайно что-то роняешь. А ты предлагаешь поймать его после того, как он уже сломал код, — это будет слишком поздно. Автоматический тест поймает его раньше.

---

[1] *Shahid M. et al.* A Study on Test Coverage in Software Testing. Proceedings of the International Conference on Telecommunication Technology and Applications (CSIT), Volume 5, 2011: «Существует с десяток типов элементов покрытия, таких как оператор, ветвь, блок, решение, условие, метод, класс, пакет, требование и покрытие потока данных».

[2] *Inozemtseva L.* Coverage Is Not Strongly Correlated with Test Suite Effectiveness. Proceedings of the 36th International Conference on Software Engineering (ICSE), 2014: «Покрытие само по себе не является лучшим показателем эффективности тестовых комплектов; во многих случаях очевидная связь в значительной степени основана на том, что наборы с высоким охватом содержат больше тестовых сценариев».

— Да, в теории это хороший подход, но как же мы далеки от этого.

— Никогда не поздно начать.

**В большинстве проектов высокий охват тестированием невозможен из-за слабого менеджмента.**

Я хочу преподнести ему ситуацию как легкую задачу, хотя это совсем не просто. Я не могу себе представить, как команда без адекватного тестирования и с большим количеством «унаследованного кода»[1] может начать писать модульные тесты и получить достойный охват. Он не спрашивал меня, что такое приличное покрытие, но полагаю, что покрыто должно быть не менее 80 % кода, если смотреть по строкам, и не менее 60 % методов[2]. Так что на самом деле я более пессимистичен, чем пытаюсь выглядеть.

---

[1] Предположительно, этот термин впервые использовал Джордж Оливетти для описания кода, поддерживаемого администратором, который не разрабатывал этот код. Однако я не смог найти подтверждений.

[2] StackOverflow question: What is a Reasonable Code Coverage % For Unit Tests (and Why)? https://goo.gl/yZtFxp.

# Поощрения и наказания

Деннис несколько секунд думает и произносит:

— Чувак, ну никто не будет писать тесты. Никого же на самом деле не интересует код без ошибок. Нам платят, чтобы исправить его, и чем чаще он «ломается», тем больше мы нужны компании. Если код станет более стабильным, компании понадобится меньше программистов — и мы потеряем работу. Так что, чувак, нам не нужны никакие тесты.

Он смеется.

Конечно, Деннис шутит, но как же он близок к истине! Действительно, именно так происходит, и мы оба прекрасно это понимаем. Это своего рода гарантия работы[1]. Программистам на окладе не нужно, чтобы код было легко поддерживать, потому что это противоречит их основной мотивации[2]: оставаться в компании как можно дольше, быть востребованными и получать зарплату как можно выше[3]. Но вот компания мотивирована в обратном: иметь как можно меньше программистов и платить им как можно меньше.

Если компания явно не требует, чтобы программисты следовали определенным правилам и принципам разработки, гарантирующим качество кода и удобство его поддержки, программисты сделают

---

1 *Spinellis D., Security J.* IEEE Software, Volume 26, Issue 5, 2009: «Похоже, что написание кода, который никто другой не сумеет понять, может значительно повысить безопасность работы».

2 *Bhatt P.* Dynamics of Software Maintenance. ACM SIGSOFT Software Engineering Notes, Volume 29, Number 5, 2004: «В некоторых ситуациях отсутствие поддерживаемости напрямую связано с безопасностью работы программиста (-ов), занимающегося (-ихся) поддержкой».

3 *Carpenter A. L.* Programming for Job Security: Tips and Techniques to Maximize Your Indispensability. Proceedings of the Twenty-First Annual SAS Users Group International Conference, Volume 19, 1996: «В целом для легко обслуживаемых программ требуется меньше программистов, и этим лишним программистом можете быть вы. Можно писать программы так, чтобы никто другой не мог их поддерживать. Можно написать программы, выдающие результаты, которые нельзя предсказать ни при поверхностном, ни при тщательном анализе кода. Освоив эти методы и научившись правильно их применять, вы можете быть уверены, что сохраните за собой рабочее место до тех пор, пока используются ваши программы».

свой код настолько неудобным, насколько это возможно[1]. На первый взгляд, это может противоречить здравому смыслу, но Деннис абсолютно прав: качество само по себе не возникнет — его нужно обеспечивать.

— Ты прав, качество нужно обеспечивать и контролировать, само по себе этого не произойдет. Мы, программисты, должны писать тесты, но сами мы этого не станем делать, — говорю я.

— Хорошо, и как ты можешь заставить писать эти тесты, а?[2]

— Думаю, что единственный способ — не принимать твой код, если в нем недостаточно тестов.

> Качество само по себе не появится. Его нужно обеспечивать и контролировать.

— Как ты это проверишь?

— Должно применяться правило, что каждое изменение проходит проверку кода[3, 4, 5]. Ты пишешь код, а затем кто-то его просматривает. И отклоняет, если нет тестов.

---

1 *Davis H.* Visual Basic 6 Secrets. Wiley, 1998: «Есть армии программистов поддержки, которые до сих пор поддерживают старые программы мейнфреймов... Если обслуживание не считается очень важным аспектом какого-либо существенного проекта, это будет означать большие проблемы в ближайшие годы. Подобно тому как "Проблема 2000 года" всполошила легионы программистов на COBOL уже после их выхода на пенсию, программы на VB, написанные сегодня без учета технического обслуживания, могут считаться вашей гарантией длительной работы».

2 *Shafer J.* Proceedings of Pacific NW Software Quality Conference (PNSQC), 2011: «Как начать модульное тестирование, если ваша команда никогда не проводила его раньше? Первое, что вам нужно, как и в случае со многими из этих рекомендаций, — это поддержка руководства. Ваши начальники должны сообщить команде разработчиков, что модульное тестирование необходимо, а не выполняется по желанию. Чтобы упростить процесс адаптации, вы должны внедрить соответствующую систему измерений и поощрений».

3 *Gilb T. et al.* Software Inspection. Addison-Wesley, 1993: «Анализ программного обеспечения — проверка программного продукта на наличие дефектов, отклонений от стандартов разработки и других проблем, осуществляемая без запуска ПО».

4 ★ *Wiegers K. E.* Peer Reviews in Software: A Practical Guide. Addison-Wesley Professional, 2001.

5 *Cohen J.* Best Kept Secrets of Peer Code Review: Modern Approach. Practical Advice. Smart Bear, 2006.

— Допустим, ты просматриваешь мой код и отклоняешь его. Я тут же пожалуюсь Маше. И она скажет тебе принять его прямо сейчас.

— А кто такая Маша?

— Дамочка, доставляющая мне головную боль каждые пару дней, — гримасничает он. — Она то ли владелец продукта, то ли дизайнер. Она говорит мне, какую функциональность добавить, и указывает, что такая-то функция должна быть к пятнице. Результаты твоей проверки будут просто проигнорированы, чувак, я тебе точно говорю[1].

Он чертовски верно мыслит. Заказчик всегда прав, независимо от того, что мы, технари, думаем об этом. Мы работаем на компанию, она нам платит, и мы должны делать то, что нам говорят. Но это в краткосрочной перспективе. В долгосрочной же приоритеты должны быть разноплановыми.

— Да, и она будет совершенно права. Она — клиент, и ей наплевать на нашу проверку кода. Мы должны заботиться о ней и...

— Но она заставит нас нарушить наш процесс, — перебивает меня Деннис.

— Да, чувак, если он хлипкий!

— Что ты имеешь в виду?

— Ну смотри. Мы должны убедиться, что, когда она придет с какой-нибудь новой идеей, кто-нибудь из нас приступит к реализации этой идеи. Скорее всего это будешь ты. Верно?

**Цели перед сотрудниками нужно ставить с учетом системы поощрений и наказаний.**

---

[1] *Dustin E.* Implementing Automated Software Testing: How to Save Time and Lower Costs While Raising Quality. Pearson Education, 2009: «Основная проблема сегодняшнего тестирования программного обеспечения состоит в том, что клиенты хотят, чтобы больше функциональных возможностей программного обеспечения доставлялось быстрее и дешевле, и в то же время ожидают, что качество программного обеспечения будет хотя бы соответствовать их ожиданиям, если не превосходить их».

Он кивает.

— Хорошо, ты как-то реализуешь озвученную идею и отправляешь свои изменения на проверку. Я проверяю твой код и отклоняю его. Затем клиентка появляется и спрашивает, почему еще не реализована нужная функциональность. Ты скажешь ей, что это все из-за меня, верно?

— Конечно, — улыбается он.

— Без всякого сомнения, она попытается заставить меня проигнорировать проверку и принять твои изменения, верно?

— Готов поспорить, что именно так и будет.

— Что я сделаю? — риторически спрашиваю я. — А я скажу ей, пусть идет подальше.

— Серьезно? — подмигивает он.

— Сто процентов, чувак, — я немного преувеличиваю, — и тогда она пойдет к Адриану или Тони, чтобы пожаловаться на нас. В итоге они обвинят во всем и накажут тебя[1].

— Накажут меня? Кнутом или еще как? Серьезно, как меня могут наказать?

— Чувак, я не знаю, должен быть какой-то механизм наказания. Может, уволят?

— За одну неудачную проверку кода, что ли? — он удивленно смотрит на меня.

— Ну, не за одну, конечно, но должен быть какой-то механизм.

— Какой?

— Может быть, штрафные баллы, может, какие-то взыскания. В общем, что-то[2]. Дело в том, что компания должна отслеживать

---

[1] *Axelrod S.* Effects of Punishment on Human Behavior. Academic Press, 1983.

[2] *Andreoni J.* The Carrot or the Stick: Rewards, Punishments and Cooperation, American Economic Review. Volume 93, Number 3, 2003: «Одни только поощрения мало влияют на сотрудников — определенное влияние имеют наказания. При их объединении влияние существенно усиливается; это говорит о том, что поощрения и наказания улучшают сотрудничество. ...Таким образом, введение политики, основанной исключительно на поощрениях, и отказ от наказания может быть ошибкой».

твои результаты. И конечно, не только наказывать, но и поощрять[1].

— Это звучит гораздо лучше, чувак.

Хотя мне трудно представить, как в традиционной компании можно реализовать явную политику вознаграждения, я пытаюсь сымпровизировать.

— Да я готов поспорить! Представь, что ты получаешь 100 баксов за каждый тикет, закрытый вовремя, и не получаешь ничего, если он не закрыт в срок. Итак, твоя награда — те 100 баксов, а твое наказание — их отсутствие. Или наказание может быть еще сильнее, если, скажем, за каждое невыполненное задание ты теряешь шанс на получение награды в следующий раз. Что-то вроде того. Никто не возьмет деньги из твоего кармана, если ты не успеешь закончить вовремя, но заплатят тебе только в том случае, если ты уложишься в срок.

— Я не уверен, что мне бы такой подход понравился, — медленно отвечает он. — Это как-то... довольно жестко[2]. Я буду напрягаться весь день, думая лишь о том, как вовремя выполнить эту задачу[3].

---

[1] *Bateman T. S. et al.* Job Satisfaction and the Good Soldier: The Relationship Between Affect and Employee "Citizenship". Academy of Management Journal, Volume 26, Number 4, 1983: «Любая ковариация между удовлетворенностью работой и эффективностью работы возникает только тогда, когда есть удовлетворение от тех вознаграждений, которые зависят от результатов. Любое представление о том, что удовлетворение вызывает сама работа — это наивная народная мудрость, не подтвержденная эмпирическими данными».

[2] *Bergen Von C. W. et al.* Contemporary Workplace Punishment and Discipline Recommendations. International Journal of Interdisciplinary Research, Volume 1, Number 1, 2012: «Хотя общепризнанно, что руководители должны избегать наказания за предполагаемые негативные побочные эффекты, эта статья иллюстрирует, что в очередной раз общепринятая мудрость о том, что уважение скорректирует неправильное поведение работников, является неверной, а правда не всегда политкорректна или настолько обнадеживающая, как хотелось бы».

[3] См. сноску 1 на с. 24: «42 % всех ошибок проектирования были напрямую связаны со стрессом программиста».

Более того, деньги не единственное, что мотивирует меня[1]. А как насчет чего-то, что может отвлекать меня?

— Это твои проблемы. Ты должен сам с этим разобраться.

— Чувак, ну прекрати. Команды разработчиков программного обеспечения так не работают.

— Хорошо, расскажи, как работают команды разработчиков программного обеспечения. Они состоят из людей, которые ни за что не отвечают, которых ничем не награждают и никогда не наказывают? Это в буквальном смысле означает отсутствие управления и приводит к проблемам, с которыми ты сталкиваешься повсюду.

— К каким проблемам?

— Никаких тестов, чувак! С чего мы начали? Ты забыл?

— Прости, я уже потерял ход твоих мыслей. У нас нет тестов, потому что нас никто не наказывает?

— Ну, по сути, именно так. Смотри, у тебя нет мотивации[2], чтобы выполнить задачу вовремя, потому что нет никаких наград и никакого наказания. Проще говоря, ничего не происходит, если ты делаешь все правильно и вовремя, если делаешь немного позже или даже если вообще не делаешь. Единственное, что действительно может случиться, — тебя уволят. Но мы же знаем, что этого не произойдет, если у компании есть деньги. На рынке дефицит программистов, а ты хороший разработчик. Итак, ты будешь здесь работать, пока не уйдешь добровольно. В такой ситуации нет абсолютно

---

[1] *Bonnera Sarah E.* The Effects of Monetary Incentives on Effort and Task Performance: Theories, Evidence and a Framework for Research. Accounting, Organizations and Society, Volume 27, 2002: «Фундаментальная гипотеза, которая предсказывает положительную взаимосвязь между наличием денежных стимулов и эффективностью выполнения задач, заключается в том, что стимулы повышают усилия, а увеличение усилий приводит к росту производительности (как в краткосрочной, так и в долгосрочной перспективе). Кроме того, был предложен ряд механизмов для объяснения связи между стимулами и усилиями, включая ожидания, личный интерес, постановку целей и самоэффективность. В отличие от этой фундаментальной гипотезы эмпирические данные указывают на то, что денежные стимулы часто не связаны с увеличением усилий и производительности».

[2] *Vroom V. H.* Work and motivation. 1964.

никакого способа мотивировать тебя писать эти тесты, если только ты не ответственный по своей природе человек[1].

Но даже в этом случае ты быстро потеряешь мотивацию, потому что увидишь, что остальные не пишут тесты и коммитят испорченный код прямо в твои прекрасные классы. Проще говоря, для того чтобы заставить всех нас писать тесты, нужно разграничить тех, кто их пишет, и тех, кто не пишет. Нужно найти способ объективно оценивать людей по их результатам, а это значит, что мы вознаградим тех, чьи результаты лучше, и накажем тех, чьи результаты хуже. Если мы придумаем способ определить это и отреагировать с помощью какого-нибудь мотивационного инструмента, например денег, проблема будет решена. Как только ты не напишешь тест, не сможешь вовремя закоммитить свои изменения и расстроишь Машу — ты будешь наказан.

**Наказание демотивирует, если оно налагается определенными людьми, а не системой четко определенных правил.**

— Но это наказание заставит меня так сильно напрягаться, что я вообще перестану работать, — отвечает Деннис после небольшой паузы.

— Ну, возможно, но я сомневаюсь в этом. Ты не похож на человека, который наказанием будет выбит из колеи. Я говорю о наказании, у которого есть своя логика.

— Логика?

— Да, демотивирует людей именно нелогичное наказание[2]. Наказание, основанное на эмоциях, отдаляет сотрудников от наказывающего их человека. Но если у наказания есть своя логика и оно

---

[1] *Deci E. L.* Intrinsic Motivation. Plenum Press, 1975: «Внутренне мотивированные виды деятельности — те, за которые нет иного явного вознаграждения, кроме самой этой деятельности. Люди, видимо, работают ради себя, а не потому, что это предполагает награду. Такая деятельность — это цель, а не средство для достижения цели».

[2] *Sethi V. et al.* What Causes Stress in Information System Professionals? Communications of the ACM, Volume 47, Issue 3, 2004: «Многие наши респонденты отметили, что открытое обсуждение производительности и ожидаемых результатов полезны для снижения уровня стресса».

не спровоцировано гневом или какими-либо другими эмоциями, мы склонны воспринимать его как урок, а не как злоупотребление полномочиями.

— Интересно, — Деннис откидывается назад и делает глоток из чашки.

Мне тоже хочется пить, поэтому я открываю бутылку минералки.

— Позволь привести пример. У тебя же есть дети, верно?

— Двое сыновей, — улыбается он.

— Представь, что ты говоришь одному из них, что он должен убираться в своей комнате каждое воскресенье. При этом обращаешь внимание, что если он не будет этого делать, то получит вдвое меньше на еженедельные карманные расходы. Позже, в воскресенье вечером, ты видишь в его комнате беспорядок. Ты говоришь ему, что он наказан и получает только половину денег, как вы договаривались ранее. Это первый сценарий. Теперь второй. Ты ничего не обсуждаешь с ребенком. Ты просто заходишь в его комнату в воскресенье, начинаешь кричать, что кругом бардак и теперь он наказан, и уменьшаешь сумму карманных расходов на 25 %. В первом случае наказание в два раза больше, но ребенку будет легче. Более того, он чему-то научится. Во втором случае он просто возненавидит тебя за то, что ты наказал его, и никаких уроков не вынесет. Ну, еще он увидит, что его папа очень эмоциональный и по воскресеньям лучше держаться от него подальше. Он не свяжет тот факт, что его комната не убрана, с наказанием. Вместо этого он свяжет с этим твою ярость и гнев. Точно так же происходит и с нами, программистами. Мы вполне адекватно воспринимаем наказание, основанное на правилах, но ненавидим, когда нас наказывают ситуативно, когда руководство просто чувствует, что так нужно.

— Может быть, я приму это рационально, но большинство людей будет недовольно самой идеей наказания, я тебе это могу гарантировать, — он снова делает глоток и смотрит на меня.

**Поощрения и наказания должны быть сбалансированы, их нужно регулярно проверять на предмет соответствия.**

— Ну, может, кто-то и будет, но нужны ли нам такие люди? В любом случае они ленивы. Может, лучше просто избавимся от них? — я шучу, конечно, но мой собеседник не улыбается. — Слушай, чувак, надо помнить, что наказание всегда должно идти вместе с поощрениями. Они должны быть сбалансированы, как учил нас Макиавелли[1]. Одни лишь наказания разрушат команду так же быстро, как и поощрения. Здесь нужен баланс. Ты как владелец задачи должен знать, что получишь, когда все сделал правильно, и что потеряешь, если накосячил. И баланс должен быть максимально точным. Если кто-то жалуется на наказание, напомни ему о поощрении. Если он продолжает жаловаться, увольняй его[2]. Такой человек явно не ориентирован на достижение результатов, а заинтересован лишь в том, чтобы хорошо провести время в офисе[3].

---

[1] См. сноску 3 на с. 98.

[2] ★ *McConnell S.* Problem Programmers. IEEE Software, Volume 15, Number 2, 1998: «Даже один проблемный программист снижает моральный дух и производительность хороших разработчиков. Проблемные программисты часто рассматриваются как имеющие "низкую производительность", но исследования и опыт работы с программным обеспечением предполагают, что такая оценка даже слишком оптимистична. Когда вам понадобится повысить производительность, не ищите того, кого вы можете добавить, ищите того, кого вы сможете убрать».

[3] *Herzberg F. et al.* The Motivation to Work, 2nd Edition, John Wiley, 1959.

## Неопределенные требования и лень

Деннис некоторое время думает и говорит:

— Хорошо, допустим, я понял идею, но как это поможет компании, если я буду наказан за неудачно реализованную функциональность? Маше вряд ли понравится, что эта функциональность не реализована.

— Ну да, изначально никому это не будет нравиться, но мы точно будем знать, что в этом виноват ты. Мы накажем тебя, а затем посмотрим, что можно сделать с этой функциональностью. Возможно, мы пропустим проверку кода или согласимся с тем, что функциональность может быть запущена на производстве без каких-либо тестов; это будет зависеть от многих факторов. В любом случае мы будем делать то же, что и сейчас. Но урок ты усвоишь. Вероятность косяков при выполнении следующего задания будет ниже, поскольку ты будешь очень мотивирован, чтобы не повторить что-то подобное еще раз.

— Да уж, я буду очень мотивирован, если ты будешь всегда наказывать меня. Как-то нечестно получается.

**Наказание вредит и демотивирует только тогда, когда объем работы нестабилен или плохо определен.**

Кажется, он немного обижен.

— А что, если Маша не знает, чего она хочет?[1] Она очень капризная: сегодня ей это нравится, завтра — нет. Ты все еще будешь обвинять меня в плохом выполнении задачи? Разве это справедливо?

— Не удивляйся, но да.

— Нет, чувак, это просто глупо, никто не может так работать. Наши задачи слишком неопределенные, чтобы мы могли использовать

---

[1] ★ *Wiegers K. E.* Software Requirements, 3rd Edition, Microsoft Press, 2013: «Не ждите, что ваши клиенты представят краткий, полный и хорошо структурированный список своих требований. Аналитики должны классифицировать несметное количество требований по различным категориям, чтобы можно было задокументировать и использовать их соответствующим образом».

твою модель наказания. Мы не можем знать, сколько времени займет выполнение задачи, что именно нужно сделать и как именно удовлетворить запросы наших клиентов. Это задача со слишком большим количеством неизвестных.

— Ты прав, дай мне пару минут.

Я выхожу, пью чай и размышляю. О чем вообще этот разговор? Полагаю, мне удалось убедить парня в том, что автоматизированное тестирование — это хорошо. И чтобы оно гарантированно было в коде, нам нужно мотивировать программистов писать тесты. Мы должны перед этим проверять качество, анализировать код и запускать скрипт, сливающий изменения, только в том случае, если с кодом все хорошо. Проблема по-прежнему заключается в понимании механизма вознаграждения и наказания. Деннис прав, это совсем не просто, потому что у задач нет четких границ. Мы попросту не знаем, за что награждать и наказывать программистов. Если мы сделаем это без явной логики, они будут обижены, а не мотивированы.

Я возвращаюсь с чашкой чая. Деннис стоит возле окна и смотрит на улицу.

— У меня есть ответ, но тебе он может не понравиться, — говорю я ему. — Программист обязан убедиться, что у задачи, с которой он работает, есть явные границы[1].

**Это ваша вина, если требования, в соответствии с которыми вы работаете, недостаточно ясны.**

---

[1] *McConnell S.* Code Complete: A Practical Handbook of Software Construction. 2nd Edition, Microsoft Press, 2004: «Всем нам хотелось бы надеяться, что, как только клиент утвердил требования, никаких изменений не произойдет. Однако чаще всего клиент не может точно сказать, что ему нужно, пока не будет написана часть кода. Проблема не в том, что клиенты — более низкая форма жизни. Подумайте: чем больше вы работаете над проектом, тем лучше вы его понимаете; то же относится и к клиентам. Процесс разработки помогает им лучше понять собственные потребности, что часто приводит к изменению требований. Если вы планируете жестко следовать требованиям, на самом деле вы собираетесь не реагировать на потребности клиента. ...Если требования недостаточно прописаны, прекратите работу, вернитесь назад и исправьте их».

Есть хорошо известный термин, ты наверняка его слышал: «критерии готовности» (Definition of Done)[1].

— Да, что-то подобное слышал. Что это?

— Это просто другое название границ задачи. У каждой задачи должны быть четкие критерии выхода — мы должны знать, когда пора прекратить работу над ней. Если ты профессиональный программист, ты не должен начинать работу над задачей, в которой критерии готовности тебе недостаточно ясны. Как инженеры, мы обязаны убедиться, что у всего, с чем мы работаем, есть четкие границы и четко определенные критерии окончания[2].

Он поворачивается ко мне:

— Но Маша даже не может четко поставить задачу, вот в чем проблема. Ты не понимаешь? Ей очень часто нужна моя помощь, чтобы выяснить, как эта функция должна работать. Иногда она просто экспериментирует, и мы удаляем немало кода. Как мы можем здесь установить какие-либо границы? Это попросту невозможно.

— Она владелец продукта. Так они обычно думают и ведут себя. Они креативны и должны импровизировать и экспериментировать. Мы не сможем изменить Машу, да и не должны ее винить. У нее есть все права делать то, что она делает: использовать тебя как инструмент для достижения своих результатов[3].

---

1 *Rubin K. S.* Essential Scrum: A Practical Guide to the Most Popular Agile Process. Addison-Wesley Professional, 2012: «Концептуально критерии готовности — это контрольный список типов работ, которые группа должна успешно завершить, прежде чем она сможет объявить свою работу потенциально готовой для отправки».

2 *Wiegers K. E.* More about Software Requirements: Thorny Issues and Practical Advice. Microsoft Press, 2005: «Неправильное определение границ задачи может иметь серьезные последствия».

3 *Saiedian H.* Requirements Engineering: Making the Connection Between the Software Developer and Customer. Information and Software Technology, Volume 42, Number 6, 2000: «Мы должны понимать, как действительно происходит формирование и анализ требований. Пользовательские требования часто расплывчаты, двусмысленны и неполны. ...В конечном счете именно клиент решает нашу судьбу независимо от того, нравится нам это или нет и считаем ли мы, что он может справиться без нас. Без денег клиента мы перестаем существовать».

— Что? Инструмент?

— Да, мужик, инструмент. Они используют нас для создания необходимого программного обеспечения, чтобы угодить своим клиентам. Мы их ресурсы, инструменты, называй как хочешь, но не обижайся — это бизнес. Твоя работа — быть полезным. Если Маша приходит к тебе и просит создать машину времени, ты не должен обещать ей сделать это завтра. Тем не менее нельзя говорить, что ты этого не сделаешь. Ты должен знать, как иметь дело с клиентом и запросами любого типа. Это часть твоих обязанностей — работа с неопределенными требованиями.

**Хороший программист знает, как быть полезным инструментом для бизнеса, который за это платит.**

Пример с машиной времени может помочь мне проиллюстрировать мысль.

— Итак, ей нужна машина времени. В такой спецификации требований, очевидно, есть ряд серьезных проблем, поэтому работу нельзя реализовать немедленно. Если бы я был программистом, то сказал бы ей, что мы определенно можем попытаться помочь, но нам нужно проделать кое-какую предварительную работу.

— Изменить некоторые фундаментальные законы физики, верно?

— Нет. В первую очередь мы должны уточнить требования. Нам нужно знать, что такое машина времени, как далеко она должна перемещать во времени, вперед или назад, как она будет проверяться, чтобы проект был принят, каков наш бюджет, сколько дней придется потратить на эту работу, каковы доступные ресурсы и т. д. Требования должны быть четкими, как того требует стандарт IEEE[1].

— Хорошо, допустим, здесь все ясно, мы точно знаем, какая машина времени нам нужна. Что дальше?

— Ну, этого никогда не случится. Уже на первом этапе все развалится. Маша поймет, что у нее не хватит ресурсов для выполнения такой задачи. Она также поймет, что не имеет ни малейшего пред-

---

[1] IEEE Recommended Practice for Software Requirements Specifications, IEEE 830-1998, IEEE Computer Society, 1998.

ставления о том, как именно должна работать эта машина времени, чтобы результат был приемлемым для нее. Возникнет много других проблем, из-за которых проект развалится прямо на стадии спецификации требований.

— Я разрушитель проектов? — Деннис улыбается.

— Ну да, если создаешь для нее что-то, что в конце концов не работает. Вместо того чтобы потратить время на спецификацию требований и документацию, ты просто создаешь прототип и демонстрируешь его[1]. Затем Маша говорит, что это совсем не то, что она хотела, и, считай, ты зря потратил время на код. Думаю, это называется «спецификацией на примере», и именно так ты помогаешь Маше понять, что ей нужно что-то другое[2].

— Разрушитель идей? — не унимается он.

— Да. Не каждая идея имеет смысл с точки зрения бизнеса. Если мы не откажемся от большинства из них, то лучшие идеи никогда не увидят свет, потому что мы будем тратить наши ресурсы поровну между всеми ними — и хорошими, и плохими[3]. Вместо этого мы должны сделать все от нас зависящее, чтобы

**Скептически относясь к требованиям и новым идеям, программист помогает бизнесу быстро обанкротиться.**

---

[1] *Andriole S. J.* Fast, Cheap Requirements Prototype, or Else! IEEE Software, Volume 11, Number 2, 1994: «Быстрое создание прототипов (иногда называемое исследовательским прототипированием) всегда выгодно и улучшает технические характеристики. Процесс состоит из девяти этапов: выявления начальных требований, требований к модели, определения ограничений, установки приоритетности начальных требований, дизайна, оценки проекта, спецификации, интерактивного прототипирования и валидации требований».

[2] См. сноску на с. 84: «Если требования нестабильны и ни одно из приведенных выше предложений не работает — откажитесь от проекта».

[3] *Johnson J.* My Life is Failure, Lulu, 2016: «Отмечу, что компании не должны избегать рискованных проектов. Организация, у которой нет опыта неудач, никогда не будет лидером на рынке, поскольку она не продвигает свою технологию быстро или достаточно далеко. Переключатель kill (kill switch — механизм, используемый для отключения неработающей машины или программы. — *Примеч. пер.*) позволяет выполнять рискованные проекты, сводя к минимуму потери. Фактически это может увеличить количество проектов. Перефразируя Томаса Эдисона, "секрет успеха — быстро отказываться от того, что не работает"».

как можно скорее отбраковать плохие идеи. Эксперимент помогает нам сделать именно это. Но задача не в том, чтобы убивать идеи! Наша главная задача — сделать как можно меньше. Потому что мы лентяи[1].

Он улыбается и почесывает затылок:

— Что?

— То, что слышал, — я беру небольшую паузу. — Послушай, мы уже выяснили, что мы инструменты для бизнеса, верно? Другими словами, ты поставщик услуг, а Маша — твой клиент. Если все так работает, то твоя главная задача, как и в любом бизнесе, — тратить меньше и зарабатывать больше. Это буквально означает сделать как можно меньше и получить за это столько, сколько сможешь. Еще это называют получением прибыли.

— Ты уверен, что можно говорить об этом на работе, чувак?

— Не волнуйся, я не говорю ничего такого, что может навредить компании. Послушай, дай мне закончить, и ты поймешь, что я прав.

**Прибыльный программист пишет меньше кода и закрывает больше тикетов и проектов.**

— Да не можешь ты быть прав, говоря, что программисты должны быть ленивыми, это безумие. Мне нужно в туалет, подожди минутку.

И Деннис уходит.

---

[1] *Raymond E. S.* The Cathedral and the Bazaar, Knowledge, Technology & Policy, Volume 12, Number 3, 1999: «Хотя я не претендую на звание великого программиста — я стараюсь быть на него похожим. Важная черта великих людей — конструктивная лень. Они знают, что вы получаете высший балл не за усилия, а за результаты и что почти всегда легче начать с хорошего частичного решения, чем с нуля».

# Эгоизм и альтруизм

Он возвращается, и я продолжаю:

— Вот в чем дело. Мы поступаем так с Машей не потому, что мы хорошие или плохие. Мы просто не хотим тратить время на сомнительные идеи. Мы хотим работать как можно меньше, одновременно выполняя как можно больше запросов и проектов[1]. И твоя работа с Машей — прекрасный индикатор твоей лени. Ты не хочешь начинать писать код, не убедившись, что требования окончательны и утверждены клиентом. Так что лень — это вполне разумный деловой подход[2]. Если бы мы стремились написать больше кода и как можно скорее, мы бы только навредили компании[3].

— Я что-то не понял, чувак, — мой собеседник выглядит обеспокоенным.

---

[1] *Kelly F. C.* The Man of the 'One Best Way': How Frank Gilbreth Studies Men and Their Ways. Popular Science Monthly, Volume 97, Number 6, 1920: «Большинство случайных улучшений были выполнены людьми настолько ленивыми, что каждый лишний шаг считался утомительным. ...Профессионалы зачастую в силу своей энергичности и способности действовать быстро могут выполнять большие объемы работы и считаться экспертами, но они гробят себя подобными нагрузками».

[2] *Lenssen P.* Why Good Programmers Are Lazy and Dumb. https://goo.gl/KZcA2W, 2005: «Я понял, что, как это ни парадоксально, хорошие программисты должны быть и ленивыми, и тупыми. ...Только ленивый программист будет уклоняться от написания монотонного, повторяющегося кода, избегая таким образом избыточности, врага поддерживаемости программного обеспечения и гибкого рефакторинга. ...Хороший программист должен быть тупым, потому что если он умный и знает это, то он: а) перестанет учиться; б) перестанет быть критически настроенным по отношению к своей работе».

[3] *Atwood J.* How to be Lazy, Dumb and Successful. https://goo.gl/NQGw2B, 2005: «Быть ленивым и тупым — это не только хороший совет для продвижения карьеры, но и ключ к успешному ведению бизнеса в области программного обеспечения».

— Смотри, группа может достичь чего-то только тогда, когда ее участники вносят вклад в общий результат и получают некоторую ценность. Это же очевидно, так ведь? Ты пишешь код — платят тебе. Я пишу код — платят мне. Маша разрабатывает продукт — ей тоже кто-то платит. Конечно, это не только деньги. Есть немало нематериальных ценностей, которые мы даем группе и которые получаем от нее. Это довольно сложный набор транзакций[1] (это важно понимать, что бы там ни говорили адепты laissez-faire[2], харизматичного[3] и трансфор-

**Ключевой фактор успеха — согласованное взаимодействие между проектом и его участниками.**

---

[1] *Odumeru J. A.* Transformational vs. Transactional Leadership Theories: Evidence in Literature. International Review of Management and Business Research, Volume 2, Issue 2, 2013: «Транзакционное лидерство, также известное как управленческое лидерство, фокусируется на надзоре, организации и групповой производительности; транзакционное лидерство — это стиль лидерства, когда лидер держит в узде своих сторонников с помощью поощрений и наказаний».

[2] *Skogstad A. et al.* The Destructiveness of Laissez-Faire Leadership Behavior. Journal of Occupational Health Psychology, Volume 12, Number 1, 2007: «Результаты указывают на то, что лидерство в стиле невмешательства может быть в большей степени контрпродуктивным, чем вообще отсутствие руководства (это связано с высоким уровнем стресса и межличностных конфликтов). Когда со стрессовыми факторами на рабочем месте и межличностными проблемами ничего не делается, они могут даже перерасти в травлю, что приведет к высокому уровню психологического дистресса и среди участников, и даже среди тех, кто наблюдает за ситуацией. Организации должны не только предотвращать злоупотребления и агрессивное лидерство, но и осознавать потенциально негативные последствия для невмешивающихся лидеров, создающие рабочую среду с высоким уровнем межличностных стресс-факторов».

[3] *Conger J. A.et al.* Toward a Behavioral Theory of Charismatic Leadership in Organizational Settings. The Academy of Management Review, Volume 12, Number 4, 1987: «Администраторы действуют как надсмотрщики, отвечающие за поддержание статус-кво. Они влияют на других с помощью власти, данной им их положением в организации. Лидеры, в отличие от администраторов, подталкивают своих последователей в направлении общей цели. Харизматичные лидеры, однако, вместо того, чтобы направлять своих последователей, стремятся к радикальному их изменению, пытаясь достичь идеализированной цели».

мационного[1] лидерства[2]). Группа продвигается вперед, когда каждый ее участник что-то дает и что-то получает: вознаграждение, удовольствие, оценку, положительные эмоции, знания, безопасность, самореализацию или просто наличные[3].

— Ну да, и что?

— Теория справедливости гласит, что люди будут давать столько, сколько могут, только когда будут получать отдачу в таком же количестве[4]. Ну конечно, они не могут взять ровно столько же, иначе не было бы никакого бизнеса и никакой прибыли, но они должны чувствовать, что количество того, что они производят, вполне сопоставимо с количеством признательности, которую они получают в ответ.

---

1 *Bass B. M.* From Transactional to Transformational Leadership: Learning to Share the Vision. Organizational Dynamics, Volume 18, Number 3, 1990: «Трансформационные лидеры имеют лучшие отношения со своими руководителями и вносят больший вклад в работу, нежели транзакционные. Более того, сотрудники говорят, что сами прилагают немало дополнительных усилий в интересах руководителей — лидеров трансформации. Организации с транзакционными лидерами менее эффективны, чем те, чьи лидеры являются трансформационными, особенно если большая часть транзакционного лидерства направлена на пассивное управление-по-исключению (вмешательство происходит только тогда, когда не соблюдаются стандарты). Сотрудники говорят, что не прилагают много усилий для таких лидеров».

2 ★ *Yukl G.* An Evaluation of the Conceptual Weaknesses in Transformational and Charismatic Leadership Theories. Leadership Quarterly, Volume 10, Number 2, 1999: «Очевидно, что теории харизматического и трансформационного лидерства предоставляют важную информацию, но некоторые концептуальные слабые места нужно исправить, чтобы теории стали более полезными. Они не дают четкого описания основных процессов влияния и не определяют, как поведение лидера связано с этими процессами. Похоже, что инструментальное соответствие является наиболее важным для транзакционного лидерства, интернализация наиболее важна для трансформационного, а личная идентификация — для харизматического лидерства. Тем не менее соответствие этих и других процессов влияния каждому типу руководства все еще изучается».

3 *Chaudhry A. Q. et al.* Impact of Transactional and Laissez Faire Leadership Style on Motivation. International Journal of Business and Social Science, Volume 3, Number 7, 2012: «Поэтому работники более мотивированы в тех банках, где используется стиль транзакционного лидерства».

4 См. сноску 2 на с. 17.

Я делаю небольшую паузу и продолжаю:

— У тебя как у члена группы может быть два противоположных отношения: эгоизм или альтруизм[1]. Если ты эгоист, то будешь только требовать, ничего не отдавая взамен, ты будешь лишь воровать у группы. Конечно, группа не станет терпеть тебя слишком долго, если, конечно, ее члены в состоянии дать отпор и изгнать тебя. Они также могут заставить тебя внести свой вклад. Я уверен, что ты видел немало подобных ситуаций в индустрии разработки программного обеспечения, когда какой-то ленивый программист ничего не делает, но довольно долго получает достойную зарплату.

— Да, очень много, — вздыхает он.

— Вот, пожалуйста. Альтернативный подход — альтруизм. Это означает, что ты делаешь все возможное и почти ничего не просишь взамен. Группа, конечно, даст тебе что-то, но намного меньше, чем ты заслуживаешь. Я уверен, что ты сталкивался и с такими ситуациями, когда кто-то буквально живет в офисе, исправляет все критические проблемы, постоянно тушит пожары и при этом зарабатывает меньше, чем другие, кто едва прикасается к коду.

— Это я, — печально говорит он.

— Да, ты на альтруистической стороне шкалы, и это не очень хорошо[2]. Но я вообще считаю, что ни эгоизм, ни альтруизм в чистом виде не годятся. Должен быть баланс. Если кто-то ведет себя эгоистично и команда это терпит — это слабость команды. Обычно это означает, что руководство слишком глупое, или слабое,

**И альтруизм, и эгоизм членов команды — симптомы неработающей системы менеджмента.**

---

[1] *Davis J. H. et al.* Toward a Stewardship Theory of Management. Academy of Management Review, Volume 22, Number 1, 1997: «Как агенты, так и принципалы в теории принципала-агента стремятся получить как можно больше выгоды при минимальных затратах. Учитывая выбор между двумя альтернативами, рациональный агент или принципал выберет вариант, который увеличивает его индивидуальную полезность. ...При выборе между поведением в свою пользу и поведением в пользу организации поведение управляющего не будет идти в разрез с интересами его организации».

[2] См. сноску 2 на с. 130: «Выдающиеся результаты работы не только не гарантируют вам повышение в должности, но могут и нанести вред».

или и то и другое. Руководство не может организовать процесс так, чтобы эгоисты были наказаны раньше.

— О, снова наказание? — заулыбался он.

— Ну конечно. Если ты ведешь себя эгоистично и команда не наказывает тебя, это сбой в процессе и его нужно устранить. С другой стороны, если ты действуешь как альтруист — это еще один тип неисправности в системе, причем еще более критический, но менее заметный. И в этом таится опасность.

Концепция, похоже, его удивляет, но я продолжаю:

— Слышал о философии fail fast[1]?

— Да, речь о выбрасывании исключений вместо попытки восстановления после ошибки[2].

— Ага, но речь идет не только о кодировании. Концепцию быстрого отказа можно применить ко всему — от жизненных ситуаций до программирования на Java[3]. Вот, скажем, ты женат...

— Я женат! — перебивает он.

— Хорошо, допустим, твоя жена говорит, что ей не нравится, когда ты поздно возвращаешься с работы. Есть два варианта ответа. Отказоустойчивый: даже если в ваших отношениях что-то не так, ты пытаешься спасти их; ты обещаешь ей, что постараешься возвращаться раньше, — и конфликт будет скрыт до поры до времени. Быстрый отказ: ты спрашиваешь ее, действительно ли она тебя не любит и почему; ты пытаешься разрушить свой брак

---

1 *Ries E.* The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses. Currency, 2011.

2 *Shore J.* Fail Fast. IEEE Software, Volume 21, Number 5, 2004: «Failing fast может происходить в довольно хрупком программном обеспечении. ..Со временем все новые и новые ошибки будут быстро выводить систему из строя, и это приведет к снижению стоимости отладки и позволит улучшить качество вашей системы».

3 *Bugayenko Y.* Elegant Objects. Volume 1, CreateSpace, 2016: «Я считаю, что можно добиться стабильности и устойчивости приложения, только если немедленно сообщать о выявленных ошибках. Чем раньше мы обнаружим проблему и вызовем отказ, тем лучше со временем станет приложение. Напротив, чем дольше мы скрываем проблему, тем большими в итоге окажутся неприятности».

как можно скорее, поскольку между вами уже есть первые признаки разногласий.

— Потрясающая идея, чувак! Я бы разрушил свой брак еще три года назад, если бы услышал тебя тогда, — улыбается он.

**Качество повышается тогда, когда неисправности сразу ведут к устраняемым впоследствии сбоям.**

— Да ты бы нашел новую жену, которая была бы более любящей и понимающей. Без обид, я просто привел эту ситуацию в качестве примера. В случае быстрого отказа, когда есть даже небольшая причина для сбоя, мы используем его. Мы не пытаемся восстановить то, что уже сломано, а просто убираем его подальше. А вот отказоустойчивый (fail safe) подход порождает проблемы. В краткосрочной перспективе это помогает, но в долгосрочной снижает качество, потому что проблемы со временем накапливаются.

Деннис встает и разминает спину:

— Интересно. А что насчет альтруистов?

— Альтруисты — отказоустойчивые ребята. Эгоисты — быстроотказники. Мы уже обсуждали, что правильно функционирующая команда должна быть сбалансирована: члены группы вносят столько, сколько забирают. Затем однажды в такой замечательной команде обнаруживается «дефект»: кто-то перестает получать столько, сколько он заслуживает, в итоге мотивационная система нарушается. Эгоисты быстро обнаружат этот недостаток, потому что перестанут работать, в то время как деньги все равно будут поступать в их карманы. Команда очень скоро развалится, потому что у нее закончатся деньги и прогресса не будет. Или не развалится и оперативно устранит эту неисправность, введя новую и эффективную мотивационную схему. Теперь давай посмотрим, что произойдет, если неисправность возникнет у альтруистов. Они будут продолжать отдавать больше, чем получают, а система все равно не почувствует этого. Проект не умрет сразу, так как не будет никакого краткосрочного ущерба, бюджет не израсходуется впустую, сумма вклада альтруистов останется высокой. Однако в долгосрочной перспективе эта неисправность уничтожит и проект, и команду.

— Как?

— Ну, рано или поздно эти альтруисты уйдут[1]. Тебя не смущает подобная перспектива?[2]

— И то верно, — вздыхает он.

— То же самое произойдет со всеми лучшими сотрудниками, и система ничего из этого не вынесет. Потихоньку все будет ухудшаться, таланты начнут уходить. Чтобы выжить, система управления должна быть разработана с учетом философии быстрого отказа. Это означает, что эгоистическое поведение должно поощряться.

**Альтруизм членов команды разрушает проект, а эгоизм делает его сильнее и помогает ему выжить.**

— А как именно это должно выглядеть?

— Компания должна поощрять программистов быть ленивыми и постоянно отстаивать свои интересы. Под ленью я подразумеваю постоянное намерение делать меньше и получать больше. Если все будут вести себя так, руководство будет постоянно сталкиваться с проблемой: как управлять этими ленивыми и жадными людьми? Компания либо умрет, либо создаст очень эффективный и сильный механизм, который будет выгоден как для программистов, так и для владельцев. Все будет сбалансировано. В тот момент, когда нарушится равновесие, эгоисты немедленно воспользуются

---

[1] *Babula M.* Motivation, Altruism, Personality and Social Psychology. Palgrave Macmillan, 2013: «Из представленных данных мы можем увидеть, что наибольшая угроза нашему психологическому развитию в области экзоцентрической альтруистической мотивации — структурное насилие в форме неудовлетворенности потребностями. Это замкнутый круг, в результате которого большое количество людей может перейти от стремления к экзоцентрическому альтруизму к ксенофобии».

[2] *Moorman R. H.* Relationship Between Organizational Justice and Organizational Citizenship Behaviors: Do Fairness Perceptions Influence Employee Citizenship? Journal of Applied Psychology, Volume 76, Number 6, 1991: «Представления о справедливости, особенно основанные на принципах вознаграждения по заслугам, служат средством в прогнозировании появления социальной ответственности. Поэтому руководители должны осознавать преимущества такого поведения по отношению к подчиненным, которое воспринимается как справедливое. Они должны думать о том, как относятся к своим сотрудникам, потому что восприятие сотрудниками такого обращения может повлиять на проявление сознательности».

преимуществом — и компании придется исправить ошибку, чтобы выжить. То есть я хочу сказать, что профессиональный разработчик должен быть эгоистом и лентяем, чтобы помочь фирме дольше оставаться на плаву. Компании не нужно, чтобы мы были полезными альтруистами. Чтобы она могла положиться на нас, нам нужно быть эгоистичными и всегда готовыми к отступлению. Как сказал Стендаль, «опереться можно лишь на то, что оказывает сопротивление»[1].

Деннис берет со стола чашку кофе, делает глоток и задумывается.

— Вроде все логично, что тут скажешь. Но как это решает первоначальную проблему Маши, чьи требования расплывчаты и неясны?

— Ну, это несложно, тебе просто нужно возвращаться назад каждый раз, когда ты не видишь достаточной ясности в ее идеях. А если они понятны, тебе нужно разбить их на более мелкие части, которые твоя лень готова принять. Проще говоря, когда задача слишком большая или неясная, ты должен предложить вместо этого сделать что-то менее громоздкое и закруглиться, просто потому что ты эгоист и не хочешь никому помогать бесплатно, — улыбаюсь я.

— Значит, это я виноват в том, что у задач, которые ставит мне Маша, нет границ?

— Точно. И ты пытаешься их решить, потому что альтруист.

— То есть я должен стать эгоистом?

— Ну да.

— Занятная философия, но я не уверен, что Тони это понравится.

— Да, я не думаю, что он будет доволен.

— Можно попытаться убедить его, что мы станем ленивыми только с завтрашнего дня, — смеется он.

— Нет, это точно не сработает, но мы можем убедить его, что нам нужна система поощрений и наказаний. Сейчас, как ты видишь, никто не мотивирован выполнять какие-либо задачи. Мы работаем в основном в силу своего естественного альтруизма. Прямо как ты. В тот момент, когда компания начнет поощрять за результаты и на-

---

[1] *Stendhal.* Le Rouge et le Noir. 1830.

казывать за их отсутствие, эгоизм начнет расти, — отвечаю я, и тут звонит телефон Денниса.

**Метод кнута и пряника — лучшая мотивационная модель для поощрения эгоизма в команде.**

Он берет трубку, а я думаю о системе, которую мы можем внедрить в компании вроде этой. Можно ли реализовать такую идею? Готовы ли все к тому, чтобы их вознаграждали и наказывали? Честно говоря, все и везде общаются в соответствии с этим довольно старым методом кнута и пряника, но очень немногие готовы это признать[1]. Даже если это художник, который рисует на холсте и говорит, что им руководят только его чувства. У такого художника тоже есть кнут и пряник. Пряник — это общественное признание, кнут — это его отсутствие, если картина недостаточно хороша. Любая работа (независимо от того, в какой сфере она выполняется и какой уровень креативности ей нужен) выполняется потому, что существует потенциальное поощрение и нежелательное наказание. Мы созданы для такого поведения, нравится это нам или нет[2].

Руководители компании, если они достаточно умны, должны использовать правильные пряники и кнуты[3]. С простой работой,

---

[1] *Bass B. M.* Leadership, Psychology and Organizational Behavior. Harper, 1960: «Есть два способа изменить уровень мотивации других: угрозой наказания и обещанием вознаграждения».

[2] *Pink D. H.* Drive: The Surprising Truth About What Motivates Us. Riverhead Books, 2011: «Действительно, сама предпосылка внешних стимулов заключается в том, что мы всегда будем рационально реагировать на них. Но даже большинство экономистов теперь не верят в это. Иногда эти мотиваторы работают, но чаще всего — нет».

[3] *Kerr S.* On the Folly of Rewarding A, While Hoping for B. Academy of Management Journal, Volume 18, Number 4, 1975: «Это не значит, что все поведение определяется формальными поощрениями и наказаниями. Конечно, верно то, что при отсутствии формальной мотивации некоторые солдаты будут патриотами, некоторые президенты будут думать об экологии, а некоторые директора детских домов будут заботиться о подопечных. Но дело в том, что в таких случаях вознаграждение не вызовет желаемого поведения, а будет лишь восприниматься как счастливая случайность. Чтобы организация могла влиять на своих членов, система вознаграждений должна положительно подкреплять желаемое поведение, а не становиться препятствием, которое необходимо преодолеть».

такой как замена масла в автомобиле, это будет проще. Для более сложных заданий, таких как создание нового алгоритма шифрования, это будет сложнее. Но тем не менее это необходимо сделать. Мы не можем, как предположил Троцкий[1], просто полагаться на врожденное желание людей работать без всяких кнутов и пряников. Что бы вы ни думали, люди не такие[2]. Макиавелли еще 500 лет назад сказал, что люди «переменчивы, лицемерны и жадны до выгоды»[3]. С тех пор ничего не изменилось.

Я готов что-то предложить и надеюсь, что это сработает.

— Есть идея! — говорю я, когда он вешает трубку. — Мы просто делаем две вещи. Во-первых, все задачи должны пройти через тикеты. У каждого, кто что-то от нас хочет, должен быть номер тикета. Более того, мы вообще ничего не будем делать, если у нас нет номера тикета. И второе правило: за каждый закрытый тикет сотрудник получает бонус. Скажем, 20 долларов, — улыбаюсь я.

— Это не сработает, хотя можно попробовать.

— Почему бы и нет?

— Давай поговорим с боссом.

Адриан пододвигает свой стул ближе к нам.

— Я слышал ваш разговор, ребята, это не сработает, — говорит он.

— Ну вот, пожалуйста, — улыбается Деннис.

— Почему же?

— Потому что очень сложно определить задачи, особенно когда мы не знаем требований. Более того, мы часто меняем приоритеты. Утром Деннису приходится выполнять одну задачу, а через несколько часов — другую. Нам нужно быть более гибкими, чем ты предлагаешь, — объясняет он.

---

[1] *Троцкий Л. Д.* Моя жизнь: опыт автобиографии. — 1930.

[2] *Medinilla A.* Agile Management: Leadership in an Agile Environment. Springer-Verlag, 2012: «Деньги — это не основной мотиватор, а фактор, удерживающий на работе. Работники хотят гордиться своей работой и наслаждаться ею благодаря самоорганизации, обучению и связи с другими».

[3] ★ *Machiavelli N.* Il Principe. 1513.

— Ты действительно думаешь, что это неправильно само по себе? Или есть какие-то конкретные проблемы в нашем конкретном случае?

— Ну, я не знаю... у нас сейчас большие проблемы... наверное, мы можем попробовать в следующем году, — отвечает он, и я понимаю, что он прав.

**Чтобы эффективно бороться с хаосом управления, нужно быть готовым бороться с людьми внутри этого хаоса.**

Невозможно изменить систему управления без смены построивших ее руководителей. Менеджмент — это продукт создавших его людей. Беспорядок, который сейчас наблюдается в этой компании, был создан именно этими людьми. И Адриан, и Деннис просто часть системы. Благодаря им она работает так, как работает. Если я действительно хочу что-то изменить, я должен дождаться либо их ухода, либо изменения их образа мышления.

Исправление системы без исправления работающих в ней людей было бы огромной травмой для них; они сделают все возможное, чтобы помешать этому. Действительно ли я хочу начинать эту битву? Определенно нет. Быть может, в следующем году, как сказал Адриан. Я улыбаюсь и на этом закругляюсь. Разговор окончен.

— Ну хорошо, отложим пока что, — говорю я Адриану и улыбаюсь Деннису.

Он удивлен:

— Ты так быстро сдался?

— Я не сдался. Не волнуйся, позже мы вернемся к этому разговору.

# Конфликты

Прошли еще две недели, и я начинаю понимать проблему этой команды — отсутствие менеджмента. Здесь все делается *ситуативно* (это означает, что все происходит так, как оно, скорее всего, будет происходить в любой момент времени). Ситуация управляет нами, а не наоборот. Однако было бы несправедливо утверждать, что никакого процесса нет вообще. Он определенно существует и основан на ряде принципов, но они довольно примитивны и держатся на старом добром правиле: кто сильнее, тот побеждает, а мы делаем то, что нам говорит победитель.

**Конфликты, как катализаторы технического прогресса, должны поощряться; процедуры их разрешения должны быть явными.**

Вот пример: на прошлой неделе мы поставляли новую версию микросервиса обработки заказов. Мы добавили в него пару новых точек входа RESTful, которые Бао и его ребята собирались использовать для извлечения истории заказа. Несколько разработчиков сказали, что для способа, с помощью которого мы упаковываем полезную информацию в JSON, не хватает нужных данных. Они хотели собрать все возможные данные в одном ответе, в то время как мы дали доступ ко всем данным только с помощью ряда микрозапросов. Коллеги сказали, что такой метод слишком громоздкий и избыточный, и попросили, чтобы мы представили им все имеющиеся данные в одном ответе JSON. Они утверждали, что это наш внутренний интерфейс и данные нам точно понадобятся. Какой смысл делать пять запросов, если одного вполне достаточно? Короче говоря, мы были правы, а они — нет, во всяком случае, нам так казалось. Мнение Бао было противоположным.

Адриан говорил, что дизайн должен определяться сервером и должен быть универсальным. Бао утверждал, что микросервис должен делать то, что просят его клиенты. Они клиенты, а мы по-

ставщики услуг и должны удовлетворить их требования. Каждый имеет право на мнение, верно? Неудивительно, что созвали совещание. В комнате собрались восемь человек, и для разрешения этого технического конфликта потребовалось более трех часов[1].

Очевидно[2], что конфликты — двигатель прогресса[3]. Без конфликтов в этом мире никогда ничего не будет достигнуто[4, 5]. Гегель

---

[1] *Romano N. C. et al.* Meeting Analysis: Findings from Research and Practice. Proceedings of the 34th Annual Hawaii International Conference on System Sciences, 2001: «Многие обзоры и опросы показывают, что совещания отнимают много времени у работников и руководителей и при этом рассматриваются как дорогостоящее и непродуктивное занятие».

[2] *Levine J. M.* Social Foundations of Cognition. Annual Review of Psychology, Volume 44, 1993: «Социально-когнитивный конфликт возникает, когда люди по-разному реагируют на одну и ту же проблему и при этом нацелены на достижение совместного решения. Интеллектуальное развитие, вызванное социокогнитивным конфликтом, представляет собой обширную когнитивную реструктуризацию, а не простую имитацию, на что указывает способность субъектов применять ответы из одной области к другой, использовать новые аргументы, которые не были упомянуты ранее, и получать выгоду от взаимодействия со сверстниками на тех же или более низких уровнях когнитивного развития».

[3] *Mayer B.* The Dynamics of Conflict Resolution: A Practitioner's Guide. Jossey-Bass, 1946: «Конфликт сам по себе не является чем-то плохим. Есть много причин назвать его необходимым фактором развития людей, семей, сообществ и обществ. Конфликт может помочь создать сообщество, определить и сбалансировать потребности людей как личностей с их потребностями как участников более крупных систем, а также помочь им четко и осознанно принимать важные решения. ...Сталкиваясь с серьезными конфликтами и реорганизовываясь для их решения, социальные организации адаптируются к изменениям».

[4] *Cloke K. et al.* Resolving Conflicts at Work: Eight Strategies for Everyone on the Job. John Wiley & Son, 2005: «Каждый конфликт, который возникает у вас на работе, предоставляет вам возможность значительно улучшить личную жизнь, повысить эффективность организации, наладить отношения с вашими друзьями, коллегами и клиентами, вывести вас из тупика».

[5] *Schulz-Hardt S.* Productive Conflict in Group Decision Making: Genuine and Contrived Dissent as Strategies to Counteract Biased Information Seeking. Organizational Behavior and Human Decision Processes, Volume 88, Issue 2, 2002.

200 лет назад сказал[1], что «война — это прогресс, а мир — это стагнация»; я это понимаю так: без создания и разрешения конфликта ни одна из его сторон не будет заинтересована в росте, развитии, улучшении и даже выживании. Война как крайняя форма конфликта — вот что побуждает нас делать все это. Цель не в том, чтобы покончить с войной и установить мир, — это приведет только к стагнации. Цель в том, чтобы найти наиболее цивилизованные, конструктивные и эффективные формы решения конфликта[2].

Есть несколько приемов (и результатов), которые могут привести к разрешению конфликта[3]. Они делятся на три категории[4]: «выигрыш — проигрыш», «проигрыш — проигрыш» и «выигрыш — выигрыш»[5].

Когда мы начали разговор, Бао хотел решить конфликт путем «выигрыш — проигрыш», то есть применив силу. Он настаивал на том, что мы допустили ошибку и ее нужно исправить. Он и слышать ничего не хотел и даже повысил голос. Если бы мы согласились с ним, то проиграли бы, а он бы вышел победителем. И тогда вся логика, лежащая в основе нашего дизайна, была бы попросту

---

1 *Hegel G. W. F.* Phenomenology of Spirit. 1807.

2 *Schweiger D. M.* Group approaches for Improving Strategic Decision Making: A Comparative Analysis of Dialectical Inquiry, Devil's Advocacy and Consensus. Academy of Management Journal, Volume 29, Number 1, 1986: «С одной стороны, конфликт повышает качество принимаемых решений, с другой — может ослабить способность группы работать вместе».

3 *Bercovitch J.* Conflict Resolution in the Twenty-first Century: Principles, Methods, and Approaches. University of Michigan Press, 2009.

4 *Burgess H.* Encyclopedia of Conflict Resolution. ABC-CLIO, 1949: «В терминологии теории игр "выигрыш — проигрыш", "проигрыш — проигрыш" и "выигрыш — выигрыш" описывают возможные результаты двусторонней игры или спора. ...Можно ожидать, что люди добровольно урегулируют спор, только если все станут победителями. Вот почему зачастую целью разрешения споров является взаимовыгодное решение».

5 ★ *Mulcahy R.* PMP Exam Prep, Eighth Edition: Rita's Course in a Book for Passing the PMP Exam. 6th Edition, 2015.

проигнорирована. Кроме того, исход «выигрыш — проигрыш» повредил бы мотивации программистов Адриана, поскольку они увидели бы, что Адриан слаб. И как они смогли бы доверять ему в будущем?

Затем стороны попытались пойти на компромисс. Это типичная ошибка, которую допускают в ходе большинства конфликтов, потому что она ведет к потерям с обеих сторон (именно поэтому называется «проигрыш — проигрыш»). Было предложено уменьшить количество HTTP-запросов с пяти до трех. Такое решение не пошло бы на пользу ни нашей концепции дизайна, ни концепции Бао. Это обходной путь, который позволил сделать всех счастливыми и расслабленными. В этом суть компромиссов: успокоить всех собеседников и бросить им маленькую кость, чтобы они заткнулись. Но насколько это полезно для архитектуры?

Наконец, мы приняли решение «выигрыш — выигрыш». Мы согласились с тем, что у первого HTTP-запроса будет необязательный аргумент, который станет указывать серверу, следует ли передавать весь результат в одном ответе JSON, исключая необходимость дополнительных последовательных HTTP-запросов. Таким образом, обе стороны в выигрыше: Адриан победил, потому что API оставался «правильно» спроектированным, согласно его пониманию; Бао победил, потому что его команда получила возможность извлекать весь набор данных за один запрос.

**При компромиссе приходится жертвовать технической корректностью ради эмоционального спокойствия, что делает его худшим методом разрешения конфликтов.**

Каждый конфликт должен привести к обоюдному выигрышу и никогда не должен решаться путем компромисса, который заставляет обе стороны в чем-то страдать. Даже принуждение одной стороны делать то, что хочет другая, лучше компромисса. Если бы мы заставили Бао принять наш дизайн, это не нарушило бы подход, предложенный Адрианом, который, по мнению хотя бы одной группы программистов, был верным.

Однако основная проблема заключалась в том, что в нашем конфликте не было предсказуемого механизма урегулирования. Мы просто не знали, что делать. Мы не знали, у кого хватит сил доказать, что правильно, а что — нет. Бао повышал голос, Адриан почти плакал, Деннис смеялся и, скорее всего, думал о том, что нужно как можно скорее уйти. Я же пытался понять, почему подобная ситуация так часто возникает во многих командах. Короче, полный бардак. Мы не знали, кто именно был боссом. Не тем боссом, который платит зарплату, а техническим боссом, которого еще называют архитектором.

У любого программного проекта должен быть технический лидер, отвечающий за все технические решения, принимаемые командой, и обладающий достаточными полномочиями для их принятия[1]. Человека можно назвать архитектором, если он несет ответственность и имеет полномочия — это две обязательные составляющие. В нашей ситуации роль архитектора была вакантной, вот почему развернулись столь длительные баталии, которые, к счастью для нас, закончились несколькочасовым совещанием и более или менее правильным дизайнерским решением. Если бы у нас был архитектор, он просто выслушал бы мнения обеих сторон и принял решение, взяв на себя полную ответственность. Все, включая Адриана, Бао, меня и обе команды, должны были просто подчиниться этому решению.

Если этому архитектору нужно совещание, он может его организовать. Если ему требуется письменное объяснение проектных решений, которые мы предлагаем принять, он может попросить об этом. Если ему нужно доказательство концепции, он может потребовать от нас его разработку. Это его решение. И он знает, что если решение окажется неверным — он заплатит за это сполна.

**Просто добавив конкретную роль архитектора, команда может эффективно решать технические конфликты.**

[1] *Booch G.* Object Solutions. Addison-Wesley, 1996: «В каждом проекте должен быть один архитектор; в более крупных проектах у главного архитектора должна быть своя небольшая команда».

Ответственность означает неизбежное наказание за ошибки; власть означает полную свободу в том, чтобы делать их. Все это нужно не только архитектору — любому человеку в команде. Однако архитектор должен нести полную ответственность за все разрабатываемые технические решения.

В большинстве команд разработчиков программного обеспечения такой роли нет. Вместо этого программисты предпочитают принимать технические решения «демократическим» образом, порой даже включая в голосование каждого сотрудника[1]. Надо ли говорить, что в такой ситуации не может быть никакой ответственности? Никто не станет принимать вдумчивые решения, зная, что нет личного поощрения за успешное решение и наказания за неудачу[2].

Еще один отрицательный результат «демократического» процесса принятия командных решений — огромная трата времени. Мы потратили более 30 человеко-часов на то, что грамотный архитектор решил бы за час. Если бы я был этим архитектором и Адриан с Бао пришли ко мне с просьбой помочь им разрешить конфликт, я бы попросил их обоих объяснить свои позиции, причем письменно, а не устно. Они потратили бы максимум полчаса. Затем я попросил бы их показать мне, как спроектирован весь API микросервиса. Это заняло бы еще около получаса. Потом мне бы понадобилось понять, сколько времени уйдет на редизайн. Я бы

---

[1] *Lu J. et al.* Multi-objective Group Decision Making: Methods, Software and Applications With Fuzzy Set Techniques. Imperial College Press, 2007: «Правило большинства. Некоторые групповые решения принимаются голосованием (возможно, неформальным образом) за альтернативные варианты или отдельные мнения после обсуждения. Мнение большинства может помочь решить проблему. Этот метод позволяет быстро принять групповое решение и отличается наличием четкого правила демократического участия в процессе».

[2] *Trevino L. K.* Ethical Decision Making in Organizations: A Person-Situation Interactionist Model. Academy of Management Review, Volume 11, Number 3, 1986: «Если организации заинтересованы в поощрении нравственных действий, им следует поощрять индивидуальную ответственность за последствия поступков на каждом уровне».

собрал у них всю нужную мне информацию. Все написанное, конечно же, осталось бы в нашей проектной документации для будущих поколений. Затем я бы просто принял решение и сообщил им об этом[1]. Мое решение с обоснованием также осталось бы в документации к проекту.

Однако большинство команд и программистов не хотят, чтобы у них был такой структурированный и дисциплинированный процесс принятия решений просто потому, что это неинтересно. Это скучно. Нет совещаний, скандалов, интриг, расследований... А есть сплошная работа. Кому это понравится, если все на окладе? Мы все знаем, что обсуждение технических решений — самый трудоемкий процесс и в то же время наименее напряженный.

Совещания длятся часами, а участники не слишком активны просто потому, что решение еще не принято, никто ничего не должен боссу, никого еще ни в чем не винят и всегда можно притвориться, что вы думаете о проекте, мечтая в это время об отпуске[2].

**Для принятия технических решений команде, ориентированной на результат, нужен сильный архитектор и продуманный процесс принятия решений, а не совещания.**

---

[1] *Converse S. et al.* Shared Mental Models in Expert Team Decision Making. Individual and Group Decision Making: Current Issues, 1993: «В иерархически структурированных командах, в которых полномочия принятия окончательного решения сконцентрированы в руках одного человека, задача команды — предоставить информацию, имеющую первостепенное значение в данной ситуации».

[2] *Votta Jr. L. G.* Does Every Inspection Need a Meeting? Proceeding of the 1st ACM SIGSOFT Symposium on Foundations of Software Engineering, 1993: «Мы обнаружили, что аудиты не так полезны, как думают руководители и разработчики. Что еще хуже, они обходятся намного дороже с точки зрения скорости разработки продукта и стоимости времени разработчика, чем это кажется на первый взгляд. ...Хотя твердо считается, что синергия является основным оправданием для проведения аудитов, экспериментальные данные показывают, что на самом деле она практически отсутствует».

Наличие в команде профессионального архитектора программного обеспечения с полными полномочиями и ответственностью — гораздо более стрессовый для всей команды фактор, поскольку этот человек не будет собирать многочасовые совещания и не станет играть в демократию.

Вместо этого архитектор будет собирать информацию в письменном виде и всегда будет возлагать вину на тех, кто предоставляет неверные данные. Процесс будет формализован, и не все сойдет с рук его участникам — каждому придется ответить за свои ошибки, невозможно будет улизнуть от ответственности, просто присутствуя на совещании и притворяясь, что все это очень интересно. Вопрос только в том, действительно ли этого хотят программисты и их руководители.

# Ручное тестирование

Джиоти — руководитель небольшого отдела ручного тестирования. Я познакомился с ней несколько дней назад. Я подхожу к ее столу, чтобы поболтать о тестировании. Эта тема мне очень интересна, хотя я никогда не работал тестировщиком.

— Что нового? — начинаю я.

— Ничего особенного, а ты как?

— Слушай, — перехожу я сразу к делу, — я хотел бы узнать, что ты думаешь о процессе тестирования в этой компании. Можешь рассказать мне, как это происходит и есть ли проблемы?

Я сажусь на пустой стул рядом с ней.

— Да никаких проблем, мы только и делаем, что тестируем, тестируем и тестируем, — улыбается она.

— Ого, вам нужен отдых! А ты не думаешь о том, что качество нашего программного продукта оставляет желать лучшего и именно поэтому вы так много тестируете?

— Ну, не знаю, — осторожно отвечает она, — но вы всегда можете его повысить.

— Я думаю, что для этого и нужны тестировщики: рассказать нам, что не так, чтобы мы могли улучшить качество. Сколько тикетов вы обычно подаете, скажем, в неделю?

— Я лично?

— Нет, вся группа. У тебя три тестировщика, верно?

— Да, всего нас четверо, но мы обычно не считаем тикеты. Я могу посчитать их для тебя сейчас, если хочешь.

— Нет, я сделаю это сам позже. Но разве тебя не интересует этот показатель? Разве найденные ошибки не являются основным результатом вашей работы?

— Что ты имеешь в виду? — улыбается она.

Я абсолютно серьезен и немного удивлен тому, что она смеется.

**Самая большая ошибка тестировщика — думать, что целью тестирования программного обеспечения является подтверждение того, что программное обеспечение работает так, как это предполагалось.**

— Насколько я понимаю, цель тестирования — найти ошибки и сообщить о них. Чем больше, тем лучше.

— Нет! — восклицает она. — Цель состоит в том, чтобы убедиться, что в программном обеспечении нет ошибок, — вот для чего нужно тестирование. Кому нужен софт с ошибками?[1]

— Как вы в отделе тестирования можете убедиться, что в программном обеспечении нет ошибок?

— Когда новая версия готовится к выпуску, нас просят протестировать ее. Мы проверяем работу всех функций. Если обнаружим, что что-то сломано, то скажем разработчикам — и они это исправят. Затем, когда все готово, мы даем зеленый свет — и они выпускают продукт. Таким образом, мы гарантируем, что сломанный продукт не поступит в производство. Вот почему нас называют обеспечением качества (Quality Assurance, QA) — мы гарантируем качество. Понимаешь?

— Так что, в принципе, если вы скажете, что ошибок нет, продукт можно выпустить, верно?

— Верно.

— А откуда вы знаете, что все работает так, как ожидалось, и что вы не пропустили что-то критическое?

— У нас есть список тестовых процедур, которые мы всегда выполняем. Давай покажу тебе, — она протягивает мне несколько листов бумаги с длинным списком элементов, возле большинства из которых есть какие-то пометки и комментарии. — Мы просматриваем весь этот список и удостоверяемся, что каждый тест пройден.

Я почесал затылок, размышляя о следующем вопросе. Она настолько не права, что я даже не знаю, с чего начать. Я вижу три очевидные проблемы в том, что сейчас услышал.

---

[1] *Humphrey W. S.* Reflections on Management: How to Manage Your Software Projects, Your Teams, Your Boss and Yourself. Pearson Education, 2010: «Некоторые люди называют дефекты программного обеспечения ошибками. В этом случае возникает неверное восприятие ситуации. В итоге, когда инженер говорит, что в программе осталось всего несколько ошибок, первая реакция — облегчение. А теперь предположим, что мы назвали дефекты не ошибками, а минами замедленного действия. Почувствуете ли вы облегчение, если программист скажет вам, что он тщательно протестировал программу и в ней осталось всего несколько мин замедленного действия? Другой термин полностью поменял ваше отношение».

В первую очередь это ее понимание философии тестирования. Д-р Майерс сказал: «Несмотря на множество доступных книг по тестированию программного обеспечения, многие разработчики, похоже, придерживаются позиции, которая противоречит расширенному тестированию», а затем добавил: «Тестирование — это процесс выполнения программы с намерением найти ошибки»[1]. Похоже, Джиоти, как и многие другие[2, 3, 4, 5], не понимает этого принципа. Она считает, что работа тестировщика состоит в том, чтобы убедиться, что в программном обеспечении нет ошибок, тогда как должно быть наоборот: работа тестировщика заключается в том, чтобы доказать, что программное обеспечение не работает[6]. Чем лучше

---

[1] См. сноску 1 на с. 59.

[2] 🔻 *Sewell N.* How to Test a System That is Never Finished, Agile Testing: How to Succeed in an Extreme Testing Environment by John Watkins, Cambridge University Press, 2009: «Именно тестировщики принимают окончательное решение по поводу того, поступит ли проект в работу».

[3] 🔻 *Bertolino A.* Software Testing Research: Achievements, Challenges, Dreams. Proceeding of Future of Software Engineering (FOSE), 2007: «Тестирование — важный этап разработки программного обеспечения. Проще говоря, оно сводится к наблюдению за выполнением системы программного обеспечения, чтобы проверить, ведет ли она себя так, как было задумано».

[4] 🔻 *Whittaker J. A.* What is Software Testing? And Why is it so Hard? IEEE Software, Volume 17, Number 1, 2000: «Тестирование программного обеспечения — это процесс проверки системы программного обеспечения для определения того, соответствует ли она своей спецификации и выполняется ли в предназначенной среде».

[5] *Gelperin D.* The Growth of Software Testing. Communications of the ACM, Volume 31, Number 6, 1988: «Мы выделяем четыре основные модели тестирования»; «Согласно одной модели мы тестируем, чтобы продемонстрировать, что какая-то версия программного обеспечения соответствует спецификации; согласно двум другим моделям мы тестируем, чтобы обнаружить ошибки; согласно последней мы тестируем, чтобы предотвратить ошибки. Эти три цели не должны противоречить друг другу; и фактически все они характерны для каждой модели тестирования».

[6] ★ *Black R.* Pragmatic Software Testing: Becoming Effective and Efficient Test Professional. Wiley, 2007: «Тестирование программного обеспечения не сводится к окончательному подтверждению того, что нет никаких дефектов, или к обнаружению дефектов. Такую задачу команда тестировщиков не в состоянии выполнить».

тестировщики выполняют свою работу, тем о большем количестве найденных ошибок они сообщают[1].

Кроме того, чем более критическими и серьезными будут обнаруженные ошибки, тем лучше. Тестировщик должен быть настроен «сломать» программное обеспечение. Майерс даже назвал тестирование «деструктивным, даже садистским, процессом», потому что всегда нужно предвзято относиться к тестируемому продукту, ожидать кучу ошибок и сбоев.

Джиоти и ее подопечные делают прямо противоположное, просматривая список процедур, которые они называют тестовыми скриптами, в ожидании того, что программное обеспечение сработает так, как от него этого ожидают. Более того, поскольку на выходе от них ждут зеленый свет, они, очевидно, хотят провести тестирование как можно скорее. Логично, что для этого они найдут как можно меньше ошибок[2]. Как при таком подходе можно обеспечивать высокое качество?

Допустим, человек сдает кровь на сифилис. Результаты, которые выдает ему врач, называются отрицательными, если человек здоров. Если в справке, не дай Бог, написано «положительный», это означает, что он заражен. Именно наличие сифилиса делает тест положительным, а не его отсутствие. Точно так же должно происходить и в программной инженерии: наличие ошибки должно сделать тест «положительным» и «пройденным».

Тем не менее Джиоти только что сказала мне, что они «удостоверяются, что каждый тест пройден». Видите, насколько неверное это

---

[1] *Chauhan N.* Software Testing: Principles and Practices. Oxford University Press, 2010: «Непосредственная цель тестирования — поиск ошибок на каждом этапе разработки программного обеспечения. Чем больше ошибок будет обнаружено на ранней стадии, тем выше показатель успешности тестирования».

[2] *Beizer B.* Black-Box Testing: Techniques for Functional Testing of Software and Systems. Wiley, 1995: «Ошибки есть во всем, что написано людьми. Не тестировать что-либо равносильно утверждению, что ошибок нет. Программисты не могут думать обо всех возможных взаимодействиях между функциональностями и между различными частями программного обеспечения. Мы пытаемся взломать программное обеспечение, потому что это единственный способ быть уверенными в работоспособности продукта».

утверждение? Она ставит все с ног на голову. Но если бы она выразилась правильно, то сразу поняла бы, что ее подход неверный. Она должна была сказать, что «мы проверяем, что каждый тест не проходит», но ведь это случается, когда ошибок нет! Таким образом, чтобы тесты оказались непройденными, следует ничего не трогать и не обнаруживать ошибок. Это сразу же гарантирует вам нужные результаты: все тесты будут провалены![1]

У тестируемого продукта и самого теста всегда должны быть обратные статусы. Когда у продукта есть проблемы, тест пройден успешно. Когда продукт работает нормально, тест не пройден. Он провалился, потому что не смог сделать то, для чего он был предназначен: сломать продукт. То же самое относится ко всем тестам — и автоматическим, и ручным, работающим с программным обеспечением или образцами крови.

**Цель тестирования — подтвердить, что программное обеспечение не работает, и показать, где именно.**

Однако все фреймворки xUnit, как было предложено в известной книге[2], используют обратную терминологию и сообщают, что эти тесты «проходят», когда не находят ошибок. Они также используют зеленый цвет для пройденных тестов и красный для тех, которые завершились неуспешно (их они маркируют как «проваленные»). Проблема здесь в формулировках (говоря о тестируемом продукте, эти слова связывают с самим тестом). Более подходящими словами будут «работает» и «сбой», потому что они будут описывать поведение продукта, а не теста. Тестируемый продукт не «проходит тест», он «работает»; он «не выходит из строя», он «валится». Тест «проходит», когда продукт «валится», и «не проходит», когда продукт «работает». Вот такая логика была бы правильной.

---

[1] ★ *Beizer B.* Software Testing Techniques. 2nd Edition, International Thomson Computer Press, 1990: «Вероятность продемонстрировать рабочее программное обеспечение уменьшается с увеличением времени тестирования; иначе говоря, чем больше вы тестируете, тем больше у вас шансов найти ошибку. Поэтому, если ваша цель — продемонстрировать рабочее программное обеспечение, ее можно достичь, вообще ничего не тестируя».

[2] См. сноску 1 на с. 68.

Вторая проблема заключается в том, что тестирование воспринимается как синоним обеспечения качества, хотя это разные вещи[1]. Тестирование — это процесс: 1) проведения экспериментов; 2) сравнения фактических результатов с ожидаемыми; 3) документирования наблюдаемых несоответствий.

Обеспечение качества (QA) — гораздо более широкая концепция[2], которая, согласно ISO 9000[3], «нацелена на обеспечение уверенности в том, что требования к качеству будут выполнены». Тестирование — не единственное, что нам нужно сделать для гарантии того, что требования к качеству выполнены[4]. Есть много других действий при разработке программного обеспечения, которые должны быть выполнены, чтобы гарантировать качество, например проверка кода, статический анализ, проверка и подтверждение требований, даже планирование и документирование технических решений. Все, что мы делаем, должно быть ориентировано на качество, и именно за это отвечают специалисты по обеспечению качества (чтобы все, что мы делаем в компании, имело требуемый уровень качества, включая собственно тестирование).

Мы не называем лаборантов, которые берут наши анализы крови, «инженерами по обеспечению качества», верно? Они ничего не обеспечивают, они просто берут кровь на анализ. Они тоже помогают обществу оставаться здоровым, но это лишь малая часть процесса «обеспечения здоровья». Они только диагностируют тот же сифилис — вот и все. Лаборанты не лечат, они не знают, что пациенты

---

1 *Hom'es B.* Fundamentals of Software Testing. Wiley, 2012: «Тесты иногда ошибочно принимают за обеспечение качества. Эти два понятия не идентичны: 1) обеспечение качества гарантирует, что процессы организации реализованы и правильно применяются; 2) тестирование выявляет дефекты и сбои и предоставляет информацию о программном обеспечении и рисках, связанных с его выпуском на рынок».

2 *Gillies A.* Software Quality: Theory and Management. 3rd Edition, Lulu.com, 2011.

3 The International Organization for Standardization, Quality Management Systems / Fundamentals and Vocabulary, ISO 9000:2005.

4 *Kaner C.* Lessons Learned in Software Testing: A Context-Driven Approach. Wiley, 2001: «Ваша команда может называться командой обеспечения качества. Это не важно. Результаты ваших тестов и отчеты об ошибках предоставляют информацию, которая способствует обеспечению качества проекта, но это результат усилий всей команды».

собираются делать с результатами теста, они даже не пытаются повторно проверить заболевших пациентов.

Если бы эта компания когда-нибудь решила получить сертификат ISO 9001[1], то слов о том, что мы тестируем наше программное обеспечение, было бы недостаточно. Работа группы ручных тестировщиков, которые проводят регулярные проверки и сообщают об ошибках, — лишь малая часть системы обеспечения качества, которую нужно организовать для получения этого сертификата.

Так что, услышав, как Джоти говорит, что их называют инженерами по обеспечению качества, я понял, что ни она, ни ее руководство в действительности не понимают, что такое обеспечение качества и как его правильно организовать. Может быть, это название QA заставляет их гордиться своей работой?[2] Возможно, руководство считает, что при разработке программного обеспечения тестирование является чем-то второстепенным? Посмотрите на количество программистов в нашей компании (их определенно больше 50) и количество тестировщиков, которых всего четыре. Это явный дисбаланс. Я не говорю, что должно быть 50 тестировщиков, но четыре по сравнению с 50 — абсолютно неправильно. Тони, очевидно, считает, что программисты создают продукт, а тестировщики помогают им[3], в то время как правильный подход состоит в том, что программисты и тестировщики создают продукт вместе. И их роли одинаково важны.

**Обеспечение качества гарантирует, что все процессы соответствуют требуемым стандартам качества, включая программирование, проектирование и тестирование.**

---

1 *Cochran C.* ISO 9001 in Plain English. Paton Professional, 2008.

2 *Cohen C. F.* Managing Conflict in Software Testing. Communications of the ACM, Volume 47, Issue 1, 2004: «Отсутствие статуса и поддержки делает работу тестировщика более сложной и трудоемкой, поскольку борьба за признание становится частью работы».

3 *Hutcheson M. L.* Software Testing Fundamentals: Methods and Metrics. Wiley, 2003: «В последние годы движущей силой разработки программного обеспечения является предпринимательское давление, график выпуска и постоянно меняющееся определение продукта. По этим и другим причинам руководство не всегда уверено, что есть смысл в тестировании. Во многих видах разработки тестирование независимой группой тестировщиков не считается целесообразным».

Более того, работа тестировщика требует больше навыков и опыта, чем работа программиста, если мы говорим о профессиональных тестировщиках и программистах[1]. Однако, к сожалению, руководство не понимает этого и не знает, какие обязанности должны быть у тестировщиков, чего от них ожидать, как измерить качество их работы[2], как мотивировать их и как платить им, чтобы они оставались заинтересованными, — в общем, зачем вообще нужны тестировщики[3]. Вот почему руководство просто бросает им кость в виде звучного «титула» — «инженер по обеспечению качества».

Третья проблема заключается в том, что Джиоти считает, что программное обеспечение можно выпускать, когда тестировщики подтверждают его работоспособность, но это распространенное за-

---

[1] *McGregor J. D.* A Practical Guide to Testing Object-Oriented Software. Addison Wesley, 2001: «Во многих отношениях быть хорошим тестировщиком сложнее, чем хорошим разработчиком, потому что тестирование требует не только отличного понимания процесса разработки и самого продукта, но и способности предвидеть возможные неисправности и ошибки».

[2] *Kitchenham B.* Software Quality: the Elusive Target. IEEE Software, Volume 13, Number 1, 1996: «Качество — сложное понятие. Не существует универсального определения качества, оно означает разное для разных людей, следовательно, сильно зависит от контекста. Таким образом, не может быть одной единицы измерения качества программного обеспечения, которая была бы приемлемой для всех. Чтобы оценить или улучшить качество программного обеспечения в вашей организации, вы должны определить те его характеристики, в которых заинтересованы, а затем решить, как вы собираетесь их измерять. Определяя качество измеримым образом, вы облегчаете другим людям понимание вашей точки зрения и соотносите ваши представления с их собственными. В конечном итоге ваше представление о качестве должно быть связано с целями вашего бизнеса. Только вы можете определить, является ли программное обеспечение хорошим для вас».

[3] ★ *Farrell-Vinay P.* Manage Software Testing. Auerbach Publications, 2008: «Тестирование — очень дорогостоящий процесс, и его результаты часто озадачивают. Есть много всезнающих людей, готовых привести правдоподобные аргументы на тему, почему можно сократить тестирование или скрыть его последствия. Если вы думаете, что на таких всезнаек можно не обращать внимания, то наверняка вы либо только что окончили университет, либо вели очень замкнутую жизнь, либо работали только в действительно хороших компаниях».

блуждение[1]. Большинство разработчиков программного обеспечения считают, что роль тестировщика заключается в том, чтобы быть «привратником» и предотвращать внедрение в производство неработающего программного обеспечения[2], а такой подход ошибочен. Д-р Вест сказал, что «программное обеспечение выпускается не тогда, когда известно, что оно корректно работает, а тогда, когда скорость обнаружения ошибок снижается до уровня, который руководство считает приемлемым»[3]. Этот принцип вытекает из постулата о том, что у любого программного обеспечения (неважно, насколько оно простое) есть немалое количество проблем, которые еще не обнаружены и не устранены[4]. С точки зрения философии я бы даже сказал, что существует неограниченное количество неисправностей, которые еще предстоит обнаружить.

Кто-то может поспорить с этим утверждением, но, если мы посмотрим на определение качества программного обеспечения, приведенное в IEEE, то все встанет на свои места: это «степень соответствия системы определенным требованиям или ожиданиям

**Решение о выпуске продукта должно приниматься не тестировщиками или программистами, а менеджером, который следит за динамикой поиска и исправления ошибок.**

---

1 *Naik K. et al.* Software System Testing and Quality Assurance. Wiley, 2008: «С одной стороны, мы улучшаем качество продукта, поскольку повторяем цикл "тест — обнаружение дефектов — исправление во время разработки". С другой стороны, мы оцениваем, насколько хороша наша система, когда проводим тесты системного уровня перед выпуском продукта».

2 *Dustin E.* Effective Software Testing: 50 Specific Ways to Improve Your Testing. Pearson Education, 2003: «В большинстве организаций, занимающихся разработкой программного обеспечения, программа тестирования словно последние "врата качества" для приложения: разрешает или предотвращает его переход из комфортной среды разработки программного обеспечения в реальный мир».

3 ★ *West D.* Object Thinking. Microsoft Press, 2004.

4 *Hetzel B.* The Complete Guide to Software Testing. Wiley, 1993: «Никогда не будет способов достичь уверенности в том, что мы полностью понимаем то, что программа должна делать (ожидаемые или требуемые результаты), и что какая-либо тестовая система, которую мы создадим, не окажется неудачной. Короче говоря, мы никогда не сможем добиться 100%-ной уверенности, независимо от того, сколько времени и сил потратим!»

пользователей»[1]. Другими словами, во всем, что работает не так, как написано в документации, есть ошибка. Более того, во всем, что не работает так, как мы ожидаем, — тоже ошибка. Видите, какое широкое определение? Если пользователей всего несколько, их ожидания более или менее предсказуемы и могут быть в какой-то степени удовлетворены. Если пользователей тысячи или даже миллионы, то можете представить, насколько разнообразными окажутся их ожидания.

Невозможно обнаружить и задокументировать все неисправности, сколько бы у нас ни было времени и ресурсов[2]. Мы можем найти самые важные, наиболее заметные и самые очевидные ошибки, но никогда не сможем зафиксировать абсолютно все. Поэтому результаты тестирования не могут быть дискретными. Нельзя сказать либо «да, оно работает», либо «нет, оно не работает», если мы говорим о программном обеспечении любой сложности.

Вот почему позиционирование фазы тестирования сразу после реализации — ужасная ошибка, которую делают большинство разработчиков программного обеспечения. Они ожидают, что тестировщики будут эдакими привратниками[3], которые всего за несколько часов или дней смогут объявить, что продукт либо хороший, либо плохой. Эффективность такого тестирования достаточно низка,

---

[1] IEEE Std 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology. IEEE Computer Society, 1990.

[2] *Ammann P. et al.* Introduction to Software Testing. Cambridge University Press, 2017: «Тестирование программного обеспечения по своей сути сложный процесс, а наша конечная цель — абсолютно правильное программное обеспечение — недостижима. ...Один из основных фактов, которые должны знать все тестировщики программного обеспечения, заключается в том, что тестирование может показать только наличие сбоев, а не их отсутствие. Это фундаментальное теоретическое ограничение; точнее говоря, проблема поиска всех сбоев в программе неразрешима».

[3] См. сноску 4 на с. 113: «Некоторые тестировщики мечтают о праве вето на выпуск продукта. За исполнение такого желания они явно заслуживают наказания. Проблема в том, что, когда тестировщики контролируют выпуск, они также должны нести полную ответственность и за качество продукта. Остальная часть команды расслабится — может быть, немножко, а может — и сильно. Если какая-то ошибка улизнет от тестировщика — остальные члены команды могут (и будут) пожимать плечами и обвинять тестировщиков. В конце концов, почему тестировщики поставили такой глючный продукт? С другой стороны, если тестировщики откладывают выпуск, их действия подвергаются тщательному разбору и давлению за то, что они такие фанаты качества».

потому что время ограничено, а мотивация окажется обратной по отношению к необходимой[1].

Правильное позиционирование подразумевает параллельную работу: когда программирование выполняется одновременно с тестированием. Программисты создают новый код и исправляют имеющийся, в то время как тестировщики «ломают» продукт и сообщают об ошибках. Когда продукт еще свежий, его легче «сломать», ошибок много, тестировщики быстро находят их. Когда код станет более стабильным и большинство критических ошибок будет исправлено, повредить его станет сложнее, и скорость нахождения ошибок упадет. В этот момент руководство принимает решение о выпуске[2], глядя на все показатели[3].

---

[1] *Lewis W. E.* Software Testing and Continuous Quality Improvement. 3rd Edition, Auerbach Publications, 2009: «"Парадокс тестирования" преследует две основные, но противоречивые цели: 1) уверенность в том, что продукт работает хорошо; 2) выявление ошибок в программном продукте до его поставки заказчику (или до следующей стадии разработки). Если первая цель состоит в том, чтобы доказать, что программа работает, она определена как "нас подсознательно направят к этой цели, то есть мы склонны использовать тестовые данные, которые с низкой вероятностью могут приводить к сбою программы". Если вторая цель состоит в выявлении ошибок в программном продукте, то как можно быть уверенными в том, что продукт хорошо работает, после того, как было показано, что он на самом деле не работает? Сегодня хорошие тестировщики признают, что вторая задача более продуктивна, чем первая, поскольку, если принять первую, тестировщик подсознательно игнорирует дефекты, пытаясь доказать, что программа работает».

[2] *Rothman J.* Release Criteria: Is This Software Done? STQE Magazine, March/April 2002: «Можно запланировать создание программного продукта, который нужен вашим клиентам, и сразу определить срок его готовности. Вам не нужно играть в игру под названием "догадайся, когда мы сможем выпустить продукт". Если вы используете критерии выпуска, чтобы знать, когда проект будет завершен, то теоретически вы сделали общедоступным и понятным обычно скрытое решение. Сделайте ваши критерии выпуска объективными и измеримыми, чтобы все участники проекта знали, к чему нужно стремиться. Сверяйтесь с ними по мере прогресса в работе над проектом, вплоть до финальной версии. Тогда вы с гордостью сможете сказать "Выпускаем!"».

[3] *Kan S. H.* Metrics and Models in Software Quality Engineering. Addison-Wesley, 2002: «Проанализировав все метрики и модели, измерения и данные, а также качественные показатели, команда должна сделать шаг назад, чтобы получить общее представление, и пропустить всю информацию через свой накопленный опыт, чтобы прийти к окончательному анализу. Окончательная оценка и принятие решений должны основываться на анализе, а не на данных. Показатели помогают принимать решения, но не заменяют его».

Адриан скажет, что все отлично и релиз готов. Джиоти скажет, что много ошибок и нужно больше времени для их обнаружения[1]. Тони скажет: «Пришло время» — и нажмет кнопку[2].

---

[1] *Xihui Z. et al.* Sources of Conflict between Developers and Testers in Software Development. Proceedings of Americas Conference on Information Systems (AMCIS), 2008: «Процесс тестирования программного обеспечения по своей сути является состязательным и создает почву для неизбежного конфликта разработчиков и тестировщиков. Между разработчиками и тестировщиками может возникнуть напряженность, потому что у них разные и зачастую противоположные цели, которые сложно соединить. Разработчики программного обеспечения и тестировщики отличаются друг от друга способом мышления, опытом, целями и восприятием их относительной важности как сотрудников.

Во-первых, разработчики и тестировщики часто придерживаются разных взглядов, так как первые всегда думают о "создании" чего-либо, в то время как вторые разрабатывают средства "сломать" то, что создали разработчики. Во-вторых, у разработчиков и тестировщиков обычно разные цели из-за различий в их функциональных возможностях. Разработчики обычно стремятся максимизировать свою эффективность, то есть выполнить работу с наименьшими усилиями, например создать ту же функциональность с наименьшим количеством строк кода или в кратчайшие сроки. А тестировщики стремятся максимизировать свою эффективность, то есть поставить конечный продукт с наилучшим качеством. В-третьих, разработчики и тестировщики могут отличаться опытом работы; например, нередко более опытные разработчики трудятся бок о бок с менее опытными тестировщиками. И наконец, разработчики и тестировщики могут по-разному воспринимать их относительную важность в организации, например, тестировщики часто считают, что им нужно постоянно работать, чтобы добиться того же уважения, которое проявляется к разработчикам».

[2] ★ *McConnell S.* Software Project Survival Guide. Microsoft Press, 1998: «Вопрос о выпуске программного обеспечения весьма непростой. Ответ должен быть на стыке между выпуском некачественного программного обеспечения на ранней стадии и выпуском высококачественного программного обеспечения на более поздней стадии. Вопросы "Достаточно ли хорошо наше программное обеспечение для выпуска сейчас?" и "Когда программное обеспечение будет достаточно хорошим для выпуска?" могут стать критически важными для выживания компании. Несколько методов помогут вам найти ответы на эти вопросы; не стоит строить инстинктивные догадки. ...Сравнивая количество новых дефектов с количеством дефектов, устраняемых каждую неделю, вы можете определить, насколько проект близок к завершению. Если количество новых дефектов за конкретную неделю превышает количество дефектов, устраненных на этой неделе, проекту еще есть куда развиваться».

Теперь я хочу убедить Джиоти, что она заблуждается.

— Как думаешь, что является твоей главной целью как начальника отдела тестирования? — спрашиваю я.

— Убедиться, что наше программное обеспечение высокого качества. Мы подтверждаем, что все функции работают как ожидается, — отвечает она и улыбается. — Я показывала тебе список наших тестовых сценариев, помнишь?

— Да, но как ты оцениваешь свой вклад в качество продукта, который мы создаем? Позволь мне сказать это иначе: каков ваш показатель успеха? Нам, программистам, определить его легко: если мы выпускаем новые функции или вовремя исправляем ошибки, у нас все хорошо. Если мы задерживаем работу, то наши показатели падают, а Тони расстраивается. Когда и почему Тони рад или не рад вашим результатам?

— У нас нет результатов, мы проверяем ваши результаты. К чему ты клонишь?

— Я думаю, что результаты тестировщиков — это ошибки, которые они находят. Мы, программисты, пишем код, это наш результат. Но код нельзя назвать завершенным и готовым к выпуску, если в нем не было найдено ошибок. Вот почему...

— Что? — перебивает она. — Вам нужны ошибки, чтобы выпустить продукт?

**Основные результаты работы группы тестирования — найденные и задокументированные ошибки.**

Кажется, Джиоти очень удивлена.

— Конечно. Если я создаю продукт и не нахожу в нем ошибок, это только даст мне понять, что ошибки найдут мои пользователи.

Я делаю паузу и думаю, как объяснить ей все это.

— Смотри, ты определенно согласишься, что в любом программном обеспечении немало ошибок, независимо от того, насколько хороши программисты, верно? Они делают ошибки, как и все люди. Не все можно протестировать с помощью автоматических тестов, да и не всегда их пишут. Вот почему, когда код написан, мы осознаем, что в нем есть некоторое количество критических ошибок, некоторое количество незначительных ошибок и очень большое ко-

личество косметических ошибок. Вопрос только в том, кто найдет их первыми — наши тестировщики или наши пользователи, хотя количество ошибок, конечно, не единственный показатель качества[1]. Следовательно, если бы я был генеральным директором своей компании и имел бы два отдела — один с программистами, а второй с тестировщиками, — то заплатил бы первому за предоставляемую функциональность, а второму — за найденные ошибки.

— Ты платил бы за ошибки? — улыбается она.

Я тоже улыбаюсь, но вовсе не шучу.

— А почему бы нет? Если результатом работы тестировщиков будут найденные ошибки, почему я не должен платить за них? Это помогло бы им понять, каков их вклад в общий результат. Они бы точно знали, насколько высоко качество их работы. Если бы компания подсчитала найденные вами ошибки, классифицировала их и каким-то образом увязала с вашими зарплатами, вы были бы очень заинтересованы в том, чтобы обнаружить больше ошибок и проверить, насколько они критичны и важны, не правда ли?

— Ну, я не знаю. Количество найденных нами ошибок зависит от многих факторов, — Джиоти приводит самое популярное оправдание, которое я обычно слышу[2]. — Мы просто не можем этого

---

[1] Sunita Chulani et al., Deriving a Software Quality View from Customer Satisfaction and Service Data. European Conference on Metrics and Measurement, 2001: «Как правило, ожидания клиентов в отношении качества основаны не на объеме или сложности покупаемого ими продукта; на их впечатление могут существенно повлиять другие характеристики продукта, обычно не связанные с дефектами (например, простота установки и использования, своевременная поддержка и т. д.)».

[2] *Black R.* Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing. Wiley, 2009: «Если вы рассматриваете другую цель тестирования — найти ошибки, — то реальным вопросом будет: "Достаточно ли ошибок мы нашли?" К сожалению, мы не можем знать того, чего не знаем. Если бы можно было знать обо всех ошибках заранее, все усилия по тестированию были бы излишними. Однако предположим, что вы можете оценить общее количество ошибок в тестируемой системе. Или, возможно, вы могли бы измерить эффективность поиска ошибок в вашей тестовой системе. Есть три метода решения этих проблем...»

предсказать. Цифры сами по себе недостаточно говорят о нашей работе. Перед каждым выпуском мы также проводим регрессионное тестирование, и оно отнимает много времени, хотя обычно мы не видим слишком большого количества ошибок. Поиск ошибок не единственный результат нашей работы. Мы делаем много чего другого.

Когда Джиоти говорит это, я чувствую, как она гордится.

— Но поиск ошибок — это самое важное, чего я ожидал бы от своих тестировщиков, если бы был генеральным директором. На самом деле не только от тестировщиков[1]. Я хотел бы, чтобы они занимались именно ошибками, все остальное для меня имело бы меньшую ценность. Знаешь, я бы даже попросил их не проводить регрессионное тестирование и сосредоточиться только на поиске ошибок в функциональности, которую создают программисты. Я думаю, что тестировщики не должны делать ничего другого: только искать ошибки. Более того, я уверен, что тестировщики и программисты не должны взаимодействовать, — я говорю это просто для того, чтобы слегка ее шокировать, — и должны быть полностью независимыми. Сейчас Адриан, Бао и другие программисты в прямом смысле говорят вам, что делать, просят вас проверить изменения, когда им это нужно, так ведь?

— Ну да, — кивает она.

— Я бы не позволил им это делать. Я бы назначил ваш отдел

**Лучший способ мотивировать тестировщиков находить все больше и больше ошибок — это платить за каждую.**

[1] *Lieberman H. et al.* Will Software Ever Work? Communications of the ACM, Volume 44, Issue 3, 2001: «Будущие разработчики программного обеспечения должны все больше ориентироваться на то, чтобы сделать программное обеспечение более взвешенным, прозрачным и адаптивным. Программное обеспечение по-прежнему будет содержать определенные ошибки (хотя, возможно, меньше), но пользователи смогут сами их исправить. Разработчики программного обеспечения будут иметь более совершенные инструменты для систематического выявления проблем, а само программное обеспечение поможет им исправлять ошибки. Взаимодействие с программным обеспечением, содержащим ошибки, будет совместным решением проблем конечного пользователя, системы и разработчика».

ответственным только передо мной, генеральным директором, и попросил сообщать мне о вашем ключевом показателе: количестве найденных ошибок. Программисты не говорили бы, что вам делать. Они просто знали бы, что вы проверяете их код, и им придется исправить ошибки, о которых вы сообщите, но не могли бы говорить вам, когда проверять и что проверять.

— И моим единственным показателем были бы найденные ошибки?

— Ну, это не такой уж простой показатель. Это не просто число, ведь ошибки разные. Они имеют разную степень серьезности и приоритетности, некоторые из них могут быть отклонены, поскольку они неверно зафиксированы или дублируют предыдущие. В показателе все это должно быть учтено[1, 2]. Кроме того, я принимал бы во внимание процент исправленных ошибок[3].

— Ты думаешь, что наша нынешняя работа не особенно эффективна? — грустно спрашивает она.

---

[1] См. сноску 2 на с. 121: «Общий показатель эффективности отдела тестирования определяет, удается ли группе тестировщиков найти большую часть ошибок до выпуска. Ошибки на продуктовом сервере или у клиента иногда называют тестовыми выходами (test escapes). Смысл в том, что ваша команда тестирования пропустила эти ошибки, но могла бы обнаружить их во время выполнения теста. Вы можете количественно оценить этот показатель следующим образом: эффективность определения дефектов = ошибки (тест) / (ошибки (тест) + + ошибки (производство))».

[2] *Jones C.* Software Defect Removal Efficiency, Computer. Volume 29, Issue 4, 1996: «Серьезный контроль качества программного обеспечения предполагает измерение эффективности устранения дефектов (defect removal efficiency, DRE). Эффективность устранения дефектов — это процент дефектов, обнаруженных и исправленных до выпуска. В принципе, измерить DRE просто. Ведите учет всех дефектов, обнаруженных в процессе разработки. По истечении фиксированного периода 90 дней добавьте обнаруженные клиентом дефекты к дефектам, найденным в процессе разработки, и рассчитайте эффективность внутреннего устранения. Если команда разработчиков обнаружила 90 дефектов, а клиенты сообщили о 10 дефектах, то DRE соответственно составляет 90 %».

[3] ★ *Kaner C. et al.* Testing Computer Software. 2nd Edition, Wiley, 1999: «Лучший тестировщик — не тот, кто найдет больше ошибок или затроллит больше программистов. Лучший тестировщик — тот, у которого большинство ошибок будут исправлены».

— Увы.

— Хм... правда? — она не расстроена, просто обеспокоена (если я правильно распознал ее эмоции).

— Ну смотри. Я никоим образом не хочу сказать, что вы плохие тестировщики или инженеры по обеспечению качества, как вы сами себя называете. Отнюдь. Я просто считаю, что, к сожалению, ваши приоритеты неправильно расставлены. Я думаю, что вас занимают чем-то менее важным, нежели ошибки, которые могут найти ребята из вашего отдела.

— Ты говорил об этом с Тони?

— Пока нет, я сначала хотел поговорить с тобой. Подумай обо всем этом, и давай попробуем поговорить с Тони на следующей неделе, хорошо?

— Хорошо, — соглашается она.

## Равенство и рабство

Деннис сидит рядом со мной:

— Слушай, чувак, можешь объяснить такую вещь? Сегодня утром Бао сказал мне добавить новые поля в этот JSON. Затем чуть позже Адриан попросил помочь Тому подготовить релиз к пятнице. Я сослался на занятость, но он сказал, что это важно. Теперь Бао решит, что я не делаю того, что нужно. Ну не могу я так работать. Они дают мне задания одновременно — и как мне понять, на кого я работаю? В конце концов я сижу допоздна, но по-прежнему получается, что для них я ненадежный человек, — он начинает рисовать в записной книжке. — Черт возьми, я все равно скоро уйду.

— Разве ты не можешь просто отправить их подальше?

— Как это?

— Просто скажи одному из них, что ты занят и у тебя нет времени. Да соври, что здесь сложного-то!

— Моя мама меня такому не учила, — кажется, он удивлен. — Не шути, чувак, я действительно хочу с тобой посоветоваться.

— Я абсолютно серьезно. Ты не должен быть хорошим мальчиком, если хочешь помочь проекту.

— Как иначе я мог бы помочь, если бы не программировал так, как они хотят? Они же уволят меня, если я не буду работать. Ты знаешь это не хуже меня, — Деннис показывает мне лист с рисунком: кулак со средним пальцем, указывающим вверх.

**Силовое управление, также известное как рабство, не исчезнет только из-за того, что мы заявим о своей цивилизованности и свободе.**

— Да, это именно то, что ты должен показать им, когда чувствуешь, что они делают что-то не на благо проекта. Послушай, вот ужасная правда: ты, как и почти все остальные, привык к системе управления путем принуждения, также известной как рабство. Люди управляют другими людьми целую вечность, и единственный известный на сей день способ заставить кого-то работать и создавать прибыль для другого — использовать старую добрую

стратегию принуждения[1]. Когда-то давным-давно основной формой было физическое принуждение и буквальная угроза смерти. И даже если подобные инструменты больше не применяются, в основном из-за социальной справедливости и лучшего образования[2], фундаментальный принцип все еще действует[3], просто в другом виде.

— Ты хочешь сказать, что я раб?

Я улыбаюсь.

— Ну, с помощью христианства[4], гуманизма, а затем и социализма мы чуть усовершенствовали наши инструменты порабощения и больше не используем физическое насилие или телесные наказания. Теперь мы пропагандируем ответственность, сочувствие, командный дух, сострадание и даже любовь. Боссу нет надобности лупить тебя кнутом, если ты не уложишься в срок. Достаточно сказать тебе, что ты разочаровываешь команду и предаешь ее ценности[5]. Благодаря своему христианскому воспитанию

---

[1] *Schermerhorn Jr. J. R. et al.* Exploring Management. 5th Edition, Wiley, 2015: «Стратегия силового принуждения основана на легитимности, поощрении и наказании в качестве основных побуждений к изменениям».

[2] *Williams E.* Capitalism and Slavery. The University of North Carolina Press, 1944: «Взлет и падение меркантилизма — это взлет и падение рабства».

[3] *Kotter J. P.* Choosing Strategies for Change. Harvard Business Review, July 2008: «Когда скорость важна и изменения нежелательны, использование принуждения, хотя и рискованно, но может быть единственным вариантом».

[4] См. сноску 1 на с. 152: «Христианская вера есть с самого начала жертвоприношение: принесение в жертву всей свободы, всей гордости, всей самоуверенности духа и в то же время отдание самого себя в рабство, самопоношение, самокалечение».

[5] ★ *Willmott H.* Strength is Ignorance; Slavery is Freedom: Managing Culture in Modern Organizations. Journal of Management Studies, Volume 30, Number 4, 1993: «Когда корпоративная культура укрепляется, сотрудникам рекомендуется посвятить себя ее ценностям, а также оценить собственную значимость согласно им. Пропагандируя эту форму преданности, сотрудники одновременно должны признавать взаимосвязь между надежностью их работы и их вкладом в конкурентоспособность товаров и услуг, которые они производят, а также нести за них ответственность. ...Поскольку их привлекает очарование технократического неформализма, сотрудники начинают дисциплинировать себя с помощью беспокойства, стыда и вины, которые возникают, когда они винят себя в том, что не оправдали святые ценности корпорации».

ты будешь чувствовать себя не лучше замученного раба в Древнем Риме[1].

— Чувак, ты уверен, что вообще в тему это сказал? Во-первых, я атеист. Во-вторых, я поступаю правильно, потому что мне это нравится больше, чем лениться[2].

— Конечно, нравится. Это то, что столетия образования и обучения сделали с тобой, твоими родителями, твоими бабушкой и дедушкой, ну и так далее вплоть до тех древних солдат и рабов, которые жили тысячи лет назад. Ты приучен к тому, что тебе нравится, когда ты поступаешь правильно и когда ты полезен[3]. Раньше же, давным-давно, людям это было не по нраву; они нуждались в регулярных наказаниях и поощрениях. Теперь нам нравится быть рабами, на современный лад — хорошими работниками[4].

**Врожденное желание всегда «поступать правильно» — главная заслуга хорошего раба.**

— И тебе?

— И мне. Мы все в одной лодке, включая Адриана, Бао, Тони и всех остальных. Старый добрый метод кнута и пряника не менялся целую вечность, но теперь он принял самую уродливую форму. Мы скрываем свое настоящее лицо за масками «всегдаполезности» и «правильнопоступаемости». Мы рабы, приятель, как и наши предки, или даже хуже.

---

1 ★ *Toner J.* How to Manage Your Slaves by Marcus Sidonius Falx. Profile Books, 2015.

2 *Organ D. W.* Organizational Citizenship Behavior: The Good Soldier Syndrome. Lexington Books, 1988: «Гражданское организационное поведение — это индивидуальное поведение, которое явно или неявно не признано формальной системой поощрений и которое в совокупности способствует эффективному функционированию организации».

3 ★ *Nietzsche F.* Zur Genealogie der Moral: Eine Streitschrift. 1887: «Рабская мораль — это, по сути, мораль полезности».

4 См. сноску 3 на с. 97: «Вне зависимости от того, имеете ли вы дело с обезьянами, крысами или людьми, вряд ли можно утверждать, что большинство организмов ищут информацию о том, какие виды деятельности вознаграждаются, а затем пытаются заниматься (или по крайней мере притворяются, что занимаются) этой деятельностью, зачастую практически исключая безвозмездные поступки».

— И что ты думаешь — есть ли свет в конце тоннеля?

— Конечно, это то, к чему я и веду. Чтобы быть полезным, эффективным и счастливым, ты должен перестать работать на босса и начать работать на себя. Это означает...

— Четырехчасовую рабочую неделю[1], — перебивает он.

— Нет, я сейчас не об этом. Я говорю о твоем отношении к проекту, его владельцу и самому себе. Ты не должен чувствовать, что обязан им чем-то или что они выше тебя. Мы вместе работаем над этим проектом, они вносят свой вклад в управление, а мы с тобой — в программирование на Java. У нас разные роли, но мы равны в своих правах и вкладе. Однако мир функционирует не так[2]. В общем, прими все сказанное за мой личный совет: не работай только из-за чувства вины.

Секунду он думает:

— А при чем тут вина?

— Вина — это современный кнут. Мы не избиваем сотрудников, мы играем на чувстве вины. На чем еще основана философия «поступай правильно»? Чувство вины толкает тебя вперед и заставляет переживать, когда Адриан или Бао разочарованы. Без вины ты бы просто показал им средний палец и пошел домой. Кстати, с минимальными последствиями, потому что ты хороший программист и они не смогут быстро найти замену. Ты знаешь это не хуже меня.

— Думаю, ты прав. Но в конце концов меня уволят.

— В конце концов да. Но ты же не такой глупый. Ты не позволишь ситуации зайти так далеко. Ты будешь держать ее под контролем и время от времени давать им то, что они хотят. Некоторые именно так и поступают, и ты ненавидишь их за это. Ты сидишь в офисе допоздна, а они возвращаются домой пораньше. Ты усердно ра-

**Свободные люди работают, когда видят для себя выгоду, а не когда чувствуют себя обязанными или виноватыми.**

---

[1] *Ferriss T.* The 4-Hour Workweek. Harmony, 2009.

[2] См. сноску 5 на с. 126: «При внедрении программ корпоративной культуры/HRM/TQM используются все мыслимые возможности для того, чтобы привить основные ценности организации ее (тщательно отобранным) сотрудникам. В той мере, в которой достигается эта миссия, корпоративная культура становится средством зарождающегося тоталитаризма».

ботаешь, а они ленивы. А вот зарплаты у вас одинаковы или у них даже больше. Они работают из-за жадности, а не из-за вины.

— Жадности? Хм... Что бы это значило?

— Это означает, что свободные люди, то есть не рабы, работают, когда им это выгодно. Я думаю, справедливо называть их жадными, а тебя — виноватым. Насчет тебя я более или менее уверен, раз ты чувствуешь себя подавленным из-за того, что Бао с Адрианом не рады твоим результатам.

— Нет, чувак, я не расстраиваюсь, когда вижу, что не такой я и хороший инженер, каким хотел бы быть. Они просто дают мне обратную связь о моей работе — положительную или отрицательную. Это не вина, а нормальные отношения между начальником и сотрудником. Если они никогда не скажут мне, где я ошибаюсь, то я перестану работать или получу плохие результаты. Я думаю, ты просто не понимаешь, в чем здесь дело, — он садится прямо на мой стол.

В принципе, он прав, но я продолжаю:

— Ну смотри. Ты думаешь, что пишешь плохой код? Или мало работаешь?

— Да нет же! Я пашу на двух начальников и задерживаюсь на работе каждый день! — восклицает он.

— Видишь! Если мы спросим у проекта — как думаешь, что мы услышим о работе Денниса?

— Что Деннис — лучший программист, в этом нет никаких сомнений! — саркастически восклицает он, и его слова недалеки от правды. Он действительно хороший программист и работает сверхурочно почти каждый день[1]. Сверхурочная работа сама по себе ни о чем не говорит, но я видел его код: он определенно один из лучших в этой компании.

— Вот видишь! Проект любит тебя, проект платит тебе, проект ставит тебе задачи, проект — твой начальник. Тем не менее тебе очень не нравится, когда два деятеля загружают тебя своими

---

[1] *Yourdon E.* Death March. Pearson Education, 2003: «Если бонусы и длительные отпуска являются мотиватором, то сверхурочные во время проекта обычно считаются демотиватором. Но это почти неизбежно в длительных проектах; действительно, это зачастую единственный способ, с помощью которого руководитель проекта может рассчитывать на выполнение работы в сжатые сроки».

требованиями и приказами. Где логика?[1] Почему твой реальный начальник, проект, доволен твоими результатами, но ты в депрессии?[2]

— Эти два деятеля... — он оглянулся. — Надеюсь, они нас не слышат. Но ведь они же представители проекта?

— Очевидно, нет, если они отправляют тебе неправильный месседж. Как мы оба знаем, проект оценивает тебя на пять звезд, а они постоянно дают тебе негативную оценку. Похоже, они понятия не имеют, что проекту на самом деле нужно от тебя, или они скрывают эту информацию. Я хочу сказать, что ты должен работать для проекта, а не для босса. Это трудно, потому что боссы всегда пытаются заставить тебя вкалывать на них, вызывая у тебя чувство вины. Я думаю, очевидно, почему они это делают.

**Очень часто руководители — это просто помехи, а настоящий начальник — это проект, над которым мы работаем и который приносит нам деньги.**

— Не совсем. Почему?

— Потому что они понятия не имеют, как иначе управлять людьми. Все, что они умеют делать, — заставлять тебя бояться их. Тогда ты просто «поступишь правильно», а они пожнут плоды. Прочти книгу Оуэна о менеджменте[3] — в ней обобщено большинство популярных методов, которые используют руководители, чтобы заставить сотрудников бояться их. Помнишь фильм «Офисное пространство»?[4]

— Конечно, он просто шикарный, — улыбается он.

---

[1] *Organ D. W.* A Reappraisal and Reinterpretation of the Satisfaction-causes-performance Hypothesis. Academy of Management Review, Volume 2, Number 1, 1977: «Удовлетворение в целом больше коррелирует с организационным просоциальным или гражданским поведением, чем с традиционной или ролевой производительностью».

[2] *Pfeffer J.* Power: Why Some People Have It and Others Don't. Harper Collins, 2010: «Одна из самых больших ошибок, которые делают люди, — думать, что хороших результатов — рабочих достижений — достаточно, чтобы получить власть и избежать организационных трудностей».

[3] См. сноску 3 на с. 15.

[4] «Офисное пространство» — комедия 1999 года Майка Джаджа.

Пусть этот фильм немного примитивен, но все же он один из моих любимых.

— В этом фильме немало горькой правды. Начальник всегда непредсказуем, играет за твоей спиной, игнорирует твои результаты, распространяет слухи, раздает ненужные задачи, требует сверхурочной работы, выставляет тебя в неприглядном свете и т. д. Арсенал его средств огромен[1].

— Похоже на нашу команду, — вздыхает мой собеседник.

— Большинство команд и руководителей выглядят именно так, поэтому фильм такой смешной — мы узнаем в его героях себя. Но первопричина не в злой натуре начальника, а в сочетании двух факторов: нашем врожденном желании иметь хозяина[2] и отсутствии у них управленческих навыков. С одной стороны, начальники попросту не знают, как составлять планы, определять стандарты качества, решать конфликты, рассчитывать бюджеты и оценивать риски. С другой стороны, мы привыкли быть мальчиками для битья и не можем представить какой-либо альтернативный вариант управления.

**Лучший способ работать с непрофессиональным руководителем — относиться к нему как к плохой погоде и сосредоточиться на том, что нужно проекту.**

Я чувствую, что говорю то, что Деннис еще не совсем готов услышать.

— И что ты предлагаешь?

— Мы не можем изменить систему, но можем изменить наше отношение и поведение. Я уже сказал тебе: не работай из чувства вины. Проще говоря, забудь о начальниках и о том, насколько ты им

---

[1] *Bugayenko Y.* How Do You Punish Your Employees? (Blog post). https://goo.gl/AaVDN4, 2016.

[2] *Kojève A.* Introduction to the Reading of Hegel. Basic Books, 1969: «На ранней стадии развития человек никогда не бывает просто человеком. Он всегда, обязательно и по сути либо Хозяин, либо Раб».

нравишься[1]. Не работай на них. Не обращай внимания на их жалобы. Не беспокойся о том, что они могут быть чем-то разочарованы. Не воспринимай их всерьез. Они просто обезьяны, распределяющие задания и пытающиеся заставить тебя чувствовать себя виноватым. Игнорируй их. Вместо этого работай на проект. Ты знаешь, что нужно клиентам, каковы технические проблемы, какой вклад от тебя требуется. Адриан, конечно, постарается скорректировать твои действия, будет давать указания, мешать тебе и отвлекать — это раздражает, но это неизбежно. Воспринимай его как плохую погоду. Или как фоновый шум. Твой настоящий босс — это проект, а не Адриан.

— Мне нравится твой подход, чувак. Хочешь поговорить об этом с Адрианом? — смеется он.

— Может быть, в следующий раз. Уже пора домой, — я улыбаюсь и встаю, пакуя свой ноутбук.

---

[1] *Bolino M. C.* Citizenship and Impression Management: Good Soldiers or Good Actors? Academy of Management Review, Volume 24, Number 1, 1999: «Есть две причины, по которым мотивы управления впечатлениями могут снизить влияние поведения граждан на эффективность организации или рабочей группы. Во-первых, когда люди предпринимают действия, направленные на произведение впечатления, их способность посвятить все свое внимание выполнению поставленной задачи снижается. Во-вторых, когда поведение мотивируется проблемами управления впечатлениями, люди могут сознательно прилагать меньше усилий или тратить меньше энергии на выполнение работы».

# Архитектор программного обеспечения

Профессиональное управление проектами основано не на чувствах или эмоциях. Оно должно основываться на логическом мышлении и планировании[1]. Однако сейчас ценится совсем не это[2]. В большинстве команд руководители — плохие родители, а сотрудники — избалованные дети. Когда все замотивированы и полны энергии, мы говорим о «слаженных» командах[3] и «горизонтальных»[4] организациях без злого начальства[5]. Однако когда кто-то ленится, босс вдруг становится боссом с правом голоса и с властью в руках.

Система управления на базе силы доминировала целую вечность в основном потому, что люди были необразованными, механизмы социальной защиты и правосудия были довольно при-

---

1 *Pfeffer J.* Leadership BS: Fixing Workplaces and Careers One Truth at a Time. Harper Business, 2010: «Если вы — лидер, стремящийся реально изменить условия труда, чтобы улучшить вовлеченность сотрудников, их удовлетворенность или эффективность, или если вы желаете наметить курс для более успешной карьеры, то вам не нужно вдохновение. Вам нужны факты, доказательства и идеи».

2 *Teal T.* The Human Side of Management. Harvard Business Review, November 1996: «Великих руководителей отличает нечто большее, чем проницательность, честность, лидерство и воображение, и это скорее напоминает героизм».

3 *DeMarco T. et al.* Peopleware: Productive Projects and Teams. 3rd Edition, Addison-Wesley Professional, 2013: «Как только начинается кристаллизация команды, вероятность успеха резко возрастает. Команда может стать неумолимой силой, стремящейся к успеху. Управлять этой стихией — одно удовольствие. Вы проводите большую часть рабочего дня, убирая препятствия с их пути, расчищая тропинку, чтобы зевак случайно не затоптали: "Так, народ, минуточку внимания. Сделайте шаг назад и придержите свои шляпы". Управление в традиционном смысле этого слова им не нужно, и уж точно не нужны дополнительные стимулы. Они уже обладают собственным импульсом».

4 *Edwin E. Ghiselli et al.* Leadership and Managerial Success in Tall and Flat Organization Structures. Personnel Psychology, Volume 25, Number 4, 1972.

5 *Foss N. J. et al.* Why Managers Still Matter. MIT Sloan Management Review, Fall 2014: «В современной наукоемкой экономике управленческий авторитет предположительно падает. Но по-прежнему есть острая необходимость, чтобы кто-то определял и внедрял организационные правила игры».

митивными[1] и у нас не было компьютеров. Теперь люди с каждым годом становятся умнее, использование любой формы принуждения затруднительно, а автоматизация вступает в свои права. Это подходящий момент, чтобы заменить злых начальников объективными показателями. Меритократия — новая философия управления[2], основная идея которой в том, что каждый получает то, что заслуживает[3], независимо от того, кто он и какова его должность[4].

Чтобы претворить в жизнь меритократию, нам нужны метрики[5]. Для объективного и бесстрастного управления требуются цифры, которые будут явными показателями наших успехов и неудач. А вот неявные показатели — это наши слова, слезы, обещания и оправдания. Если мы даем нашим руководителям только неявные показатели, то не можем рассчитывать на получение чего-то логичного и разумного. Мы ведем себя как дети, стараясь угодить родителям, а руководители ведут себя как мамы и папы,

---

1 *Young M.* The Rise of the Meritocracy. Transaction Publishers, 1994: «Социалисты ускорили рост крупных организаций и, в отличие от малого бизнеса, поощряли продвижение с учетом заслуг».

2 *Highsmith J.* Agile Project Management: Creating Innovative Products. Addison-Wesley Professional, 2nd Edition, 2009: «Основной принцип эгалитарной меритократии особенно популярен в движении Agile. Кроме того, это основополагающий принцип, который определяет то, как воспринимают себя большинство приверженцев Agile».

3 *Fielding R. T.* Shared Leadership in the Apache Project. Communications of the ACM, Volume 42, Number 4, 1999: «Проект Apache — это меритократия: чем больше работы вы проделали, тем больше вам разрешено делать».

4 *Eckhardt E. et al.* The Merits of a Meritocracy in Open Source Software Ecosystems. Proceedings of the 2014 European Conference on Software Architecture Workshops, 2014: «В рамках истинной меритократии можно ожидать, что люди будут выполнять меньше работы, чем их подчиненные, которые их сменят».

5 *Fenton N. E. et al.* Software Metrics: A Rigorous and Practical Approach, Revised. 2nd Edition, Course Technology, 1998: «Трудно представить электроинженерию, машиностроение и строительное дело без системы измерений. Действительно, наука и техника без измерений не могут быть ни эффективными, ни практичными. Но в разработке программного обеспечения измерение считалось роскошью... Даже если измерения выполняются — обычно это делается нечасто, непоследовательно и частично».

грозящиеся наказать и дающие шоколадные батончики в награду. Если бы мы вместо этого выглядели для них как роботы, а на наших головах всегда светились зеленые или красные огоньки, им было бы гораздо удобнее говорить нам, что делать, поощрять и наказывать нас[1].

Конечно, все мы люди и у нас есть чувства и эмоции. Мы любим и ненавидим, у нас бывает хорошее и плохое настроение, мы полны энергии или иногда ленимся, но все это просто помехи для руководителя, чья работа — отфильтровывать их и фокусироваться на наших результатах. Хороший начальник может проигнорировать помехи, а плохой возьмет их за основу для принятия решений. То же самое с детьми: если мы эмоционально реагируем на их плохое поведение, они не извлекают из этого уроков и ведут себя еще хуже; если мы отвечаем логически и разумно, они уважают нас и учатся.

> **Меритократия — это управленческая парадигма будущего, когда все будут вознаграждены и наказаны за то, что они делают, а не за то, кем являются.**

Наши боссы злятся, повышают голос и вводят различные наказания не из-за плохого характера, а потому, что они просто не знают, что еще могут сделать при отсутствии каких-либо сведений о нас. Все, что они способны сказать о нас как о сотрудниках, — это то, что мы «делаем все так» или «что-то не так». Как еще они могут координировать нашу работу в условиях отсутствия данных?

Деннис прерывает мои размышления:

— Вот ты говорил, что мой настоящий начальник — это проект. Что это означает в практическом смысле? Кто что делает? Какова роль лидера команды?

— Все просто: руководитель проекта, или тимлид, как его называют, является архитектором проекта.

---

[1] *Kerzner H.* Project Management, a Systems Approach to Planning, Scheduling and Controlling. 12th Edition, Wiley, 2017: «Проблема в том, что стандартные планы повышения заслуг и бонусов основаны на индивидуальной ответственности, в то время как над проектом чаще всего работают команды с общей подотчетностью, ответственностью и контролем. Обычно очень трудно приписать успех или неудачу проекта одному человеку или небольшой группе».

Он вскинул брови, собираясь спросить, но я продолжаю:

— Да, архитектор. Обрати внимание: архитектор проекта, а не продукта.

— Я на самом деле не вижу никакой разницы.

— Прочитай PMBOK[1] или еще лучше — эту замечательную книгу Риты[2].

— Я программист, а не управленец. Зачем мне это? — улыбается он и говорит то, что я так часто слышу.

**Знание правил управления — один из ключевых факторов успеха в технической карьере.**

— Зря ты так. Если ты не разбираешься в управлении, то никогда не займешь должность выше обычного программиста. Возможно, станешь старшим, но никогда не будешь руководителем отдела, вице-президентом или техническим директором.

— Я и не хочу. Я хочу продавать клубнику, — говорит он абсолютно серьезно.

— Что?

— Чувак, посмотри, что мы делаем. Мы пишем код, зарабатываем деньги, покупаем все эти макбуки и айфоны, но это не делает нас счастливыми. Смысл жизни в том, чтобы быть счастливым, так ведь?

— Ну, думаю, да. Ты не счастлив?

— Я был бы намного счастливее, если бы продавал клубнику, — вздыхает он. — Просто открой фруктовый магазин на углу и делай людей счастливыми каждый день всю жизнь. Ты можешь себе это представить?

— Да ты бы умер со скуки! Тебе нужно что-то делать.

— Возможно, ты прав. Но я действительно люблю клубнику!

— Пока ты не продаешь клубнику, ты обречен работать в программном бизнесе — и лучше быть на вершине. Пока ты програм-

---

[1] Project Management Institute (PMI), A Guide to the Project Management Body of Knowledge. 6th Edition, 2017.

[2] См. сноску 5 на с. 102.

мист, ты находишься на нижней ступени. Ты много работаешь и мало зарабатываешь. Когда ты становишься тимлидом или руководителем отдела, ты работаешь намного меньше, а рискуешь и зарабатываешь больше. Затем в конце концов ты становишься директором, рискуешь намного больше, седеешь, каждые несколько месяцев проверяешь простату, но зарабатываешь много. Потом, если ты достаточно успешен, становишься вице-президентом или даже владельцем компании, прекращаешь понимать что-либо в разработке программного обеспечения, начинаешь играть в гольф, твой проктолог становится твоим лучшим другом, и ты покупаешь «Бентли». На самом деле ты можешь подняться по всем этим ступенькам быстрее, если ты рисковый и тебе повезет. Кроме того, если ты выберешь правильный стартап, то выиграешь джекпот и доберешься до «Бентли» всего, скажем, лет за шесть. Но это крайне редкая ситуация, я бы на нее не ставил.

— Что-то я не вижу клубники в твоей истории, — улыбается он.

— А нет ее! Это жалкая жизнь. Обрати внимание, я даже не упомянул трех жен, которых ты поменяешь, нескольких адвокатов по бракоразводным процессам, которые будут обманывать тебя даже больше, чем твои близкие, пятерых отпрысков, которые по разным причинам будут ненавидеть тебя, а еще антидепрессанты, которые ты будешь принимать, чтобы сохранить хоть какой-то интерес к жизни.

**Быть богатым и успешным неизбежно означает быть частью системы с ее иерархическими законами власти и подчинения.**

— Что ты хочешь этим сказать? — он больше не улыбается.

— Да тебе нужно обмануть систему, прежде чем она убьет тебя. Ты должен стать достаточно умным для того, чтобы взять от нее лучшее и не угодить в западню.

— И что же?..

— Деньги! Деньги — это хорошая часть. Рабство — плохая. Под рабством я имею в виду именно тех жен, детей, гольф и твою простату. Научись брать деньги и ничего не отдавать взамен. Иными словами, не становись рабом, а зарабатывай как хороший раб.

— Рабы мало зарабатывают, чувак.

— На самом деле не так уж мало. Ты представляешь себе рабов бедолагами, строящими пирамиды. Современные рабы другие. Посмотри на меня и на себя. Вот прямо здесь такие себе современные рабы: умные, интеллигентные, профессиональные, политкорректные, хорошо выглядящие — и подавленные.

Адриан подходит к моему столу и прислушивается. Я понимаю, что пришло время сменить тему:

— Вернемся к моей точке зрения: руководитель проекта — это архитектор проекта. Продукт и проект — это две разные вещи. Продукт — это то, что мы разрабатываем, часть программного обеспечения. Это материальная вещь. А вот проект — это деятельность, а не вещь. Проект — это то, что происходит, когда мы сидим все вместе за компьютерами и пишем программное обеспечение. Проект может производить несколько продуктов, а продукт может быть создан несколькими проектами. Пока все правильно?

— Думаю, да, — отвечает Деннис.

Адриан возвращается к своему столу, разворачивает стул, чтобы видеть нас, садится и обращается в слух.

Я ПРОДОЛЖАЮ:

— У программного продукта есть архитектор, то есть человек, принимающий ключевые технические решения и несущий за них полную ответственность. Задача архитектора — не разрабатывать продукт, а наладить все так, чтобы каждый мог внести свой вклад...

— Наладить? Что ты имеешь в виду? — перебивает Адриан.

— Смотри, архитектор не создает программное обеспечение[1], хотя он может и должен уметь писать код[2], а устанавливает границы

---

[1] *Fowler M.* Who needs an architect? IEEE Software, Volume 20, Number 4, 2003: «Ценность архитектора обратно пропорциональна количеству принимаемых им решений».

[2] *Langsworth A.* Should Software Architects Write Code? November 2012, https://goo.gl/vGN61n: «Понимание кода означает, что архитектор может эффективнее продвигать свое мнение, а не просто полагаться на мнение того разработчика, который более убедителен».

и ограничения в дизайне, принимает решения, которые не позволяют продукту развалиться. Допустим, мы должны создать какую-то простую программу — калькулятор. Ты программист и можешь написать ее менее чем за час. Я архитектор. Я говорю тебе, что калькулятор должен быть написан на Java, иметь тестовое покрытие более 80 % и в работе мы не будем использовать фреймворки. В этом суть деятельности архитектора. Я не пишу код, но я инструктирую тебя, как это сделать, наложив некоторые ограничения на твою работу. Но вот если я делаю это в устной форме — я плохой архитектор. Хороший архитектор настроит среду разработки так, чтобы ты как разработчик не мог нарушать правила.

— Правила? — Адриан, похоже, обеспокоен.

— Да, правила. Смотри, это может прозвучать грубо, но для создания хорошего продукта всегда нужно уметь говорить «нет». Ты не можешь создать ничего приличного, если всегда принимаешь то, что предлагает твоя команда. Ты должен стать плохим парнем, который почти всегда отвергает действия всех остальных и требует чего-то лучшего. Иначе зачем ты нужен на проекте, верно? Если ты любишь всех и принимаешь почти все, что делают другие, ты не нужен. Они могут выжить и без тебя. А хороший архитектор — это всегда плохой парень, который привык к тому, что его ненавидят.

**Основная задача архитектора программного обеспечения — определить технические правила, принципы и ограничения, которых должна придерживаться команда.**

Деннис улыбается:

— Да ладно! Ненавидят?

— Я знаю, звучит странно, но подумай об этом. Обеспечение качества всегда сводится к тому, чтобы отрезать то, что плохо, и дать жизненное пространство тому, что хорошо. В программном проекте «отрезание» заключается в отклонениях изменений кода, прекращенных задачах, отстраненных программистах, жестких правилах, строгом контроле и постоянных спорах. В большинстве случаев, особенно когда проект еще молодой, а команда недостаточно зрелая,

плохое есть везде. И тогда архитектор неизбежно становится всем ненавистен, если он действительно хорош в своем деле, потому что он должен удалить все плохое, чтобы позволить хорошему выжить и процветать. Конечно, плохое будет сопротивляться и найдет способы избавиться от архитектора.

— Ага, прямо как супергерой! — Деннис не хочет воспринимать меня всерьез.

— Именно к этому и клоню, — серьезно отвечаю я. — Но хороший архитектор умнее супергероя. Он не будет ежедневно сражаться со злом. Или, может быть, он слишком ленив для этого, но он не будет этого делать. Он «настраивает» продукт так, чтобы тот боролся за себя. Он возводит вокруг продукта стену качества, которая не позволяет проникнуть ничему плохому.

— Стену? — Деннис чешет нос.

— Это метафора. Архитектор устанавливает правила и обеспечивает их соблюдение, создавая программное обеспечение, которое поможет в этом. Программное обеспечение, контролирующее качество, — это стена. Вернемся к примеру с калькулятором. Я архитектор и хочу, чтобы у программы всегда было 80 % покрытия тестами. Если я просто скажу тебе, программисту, сделать это, ты согласишься, а завтра по каким-то причинам проигнорируешь это правило. Может быть, тебе просто будет лень создавать новый юнит-тест, или ты будешь спешить, или я буду в отпуске, или меня уже уволят. Неважно. Рано или поздно это случится. Недостаточно просто попросить тебя соблюдать покрытие тестами более 80 %. Мне нужно найти способ обеспечить соблюдение правила. Первый и самый примитивный способ — каждый день находиться в офисе и поднимать кипиш всякий раз, когда ты нарушаешь правило. Ты будешь бояться меня или ненавидеть, но правила будут соблюдаться. Так ведут себя плохие архитекторы.

**Хороший архитектор программного обеспечения большую часть времени тратит на возведение автоматизированной стены качества вокруг артефактов проекта.**

— А как ведут себя хорошие? — спрашивает Адриан.

— Хорошие архитекторы следят за тем, чтобы было технически невозможно нарушить их правила, — мне приятно, что Адриан заинтересовался. — Например, путем технического запрета любых коммитов или слияний Git в ветку master, если эти изменения не нарушают правила контроля покрытия. Архитектор просто следит за тем, чтобы сборка останавливалась, если покрытие меньше 80 %, и чтобы никто не смог закоммитить, если сборка остановлена. Реализовав это однажды, архитектор может спокойно отдыхать, зная, что его правила всегда будут соблюдаться. Ну, пока технические механизмы контроля на месте. Не нужно бродить по офису, проверяя, кто чем занимается и сколько правил собирается нарушать. Вместо этого архитектор реализует их один раз, обеспечивает их соблюдение и приступает к разработке новых правил. Задача архитектора состоит в том, чтобы постоянно работать с правилами. Точнее, анализировать, какие технические риски наиболее критичны в проекте, предотвращать их возникновение, приводить правила в действие с помощью программных сценариев и триггеров, смотреть, как они соблюдаются или нарушаются командой, и начинать все сначала с новыми рисками. Вот это я и называю настройкой проекта.

Адриану моя концепция явно не по душе:

— Это, конечно, интересно, а как насчет помощи команде, обучения младших программистов, проверки кода, встречи с владельцами продукта? Разве архитектор не должен всем этим заниматься?

— Думаю, все это не так важно. Более того, я бы сказал, что все это может быть только в том случае, когда архитектор не выполняет свою работу должным образом, то есть недостаточно тщательно настраивает продукт.

— Тщательно? — Деннис любит ловить на слове.

— Да, вполне подходящее слово. Чем тщательнее разработаны правила, тем более стабилен и самодостаточен проект. Его будет трудно повредить или вывести из строя.

— Можешь ли ты привести несколько примеров правил, о которых говоришь? — Адриан по-прежнему заинтересован.

— Конечно. Например, сборка всегда должна быть чистой; покрытие кода всегда должно быть на определенном уровне; статические анализаторы не должны выдавать ошибок; каждая модификация кода должна пройти две проверки; изменения в базе данных должны быть версионными; весь исходный код должен быть в Git; дизайнерские решения должны быть задокументированы как минимум с двумя возможными альтернативами; развертывание должно быть автоматическим; рабочая среда не должна быть доступна для команды разработчиков. Я, наверное, могу придумать много других правил. Важно то, что большинство правил можно и нужно реализовывать с помощью программных средств. Работа архитектора состоит в настройке продукта таким образом, чтобы он защищал себя от поломок.

**Работа архитектора состоит в том, чтобы постоянно придумывать и внедрять новые правила контроля качества, делая их как можно более строгими.**

Несколько секунд все молчат. Деннис встает и разминает спину. Адриан продолжает крутиться в кресле. Я делаю глоток из своей чашки. Чай холодный. Пора заварить новый. Я забыл, с чего мы начали, но, похоже, дал им пищу для размышлений. Может, я запамятовал о каких-то важных правилах?

Проблема в том, что в нашей команде нет такого понятия, как архитектор программного обеспечения. Это очень типично и для многих других команд — у них просто нет архитекторов[1]. У них

[1] *Keeling M.* Design It!: From Programmer to Software Architect. Pragmatic Bookshelf, 2017: «В одних командах архитектор — конкретная должность. В других командах этого нет, и сотрудники делят между собой обязанности архитектора. Некоторые команды говорят, что у них нет архитектора, но если вы внимательно посмотрите, то увидите, что кто-то выполняет его обязанности, не осознавая этого... Если в вашей команде нет архитектора, поздравляю, вы получили работу!»

есть программисты, тестировщики, начальники, дизайнеры, но не архитекторы. Ну, у них иногда используется это звание[1]. Его дают некоторым людям, когда они собираются уйти, а у компании не хватает денег, чтобы удержать их, повысив зарплату. Вместо повышения зарплаты эти бедные кодеры получают звание архитектора и продолжают писать код.

На самом деле я не прав: наличие такого звания позволяет им больше не писать код. Архитекторы обсуждают, разговаривают, винят программистов в ошибках, натаскивают младших программистов и делают как раз то, что сделал бы плохой архитектор, о чем я только что сказал Деннису: поднимают кипиш, когда что-то идет не так. В современных командах нет настоящих архитекторов. По крайней мере я не видел ни одного.

Архитектор должен иметь достаточно полномочий для принятия любого технического решения и нести личную ответственность за последствия.

Настоящий архитектор похож на режиссера[2]. Он несет личную ответственность за продукт. Если фильм скучный, мы обвиняем режиссера. Если актеры плохо играют, мы обвиняем режиссера. Если камера дрожит, мы обвиняем режиссера. Мы всегда обвиняем режиссера. С другой стороны, мы отдаем ему должное, если фильм отличный. Фильм — дитя режиссера. Конечно, чтобы правильно воспитать это дитя, режиссер должен обладать всей необходимой властью и полномочиями для принятия любого решения в процессе съемки фильма. Как еще можно нести ответственность за что-то, если не можешь принять все необходимые решения, верно?

---

[1] *Kruchten P.* What Do Software Architects Really Do? Journal of Systems and Software, Volume 81, Number 12, 2008: «Кент Бек на семинаре OOPSLA в Ванкувере осенью 1992 года: "Хм, архитектор программного обеспечения — это новое напыщенное название, которое программисты хотят видеть на своих визитных карточках, чтобы оправдать свои шикарные зарплаты"».

[2] *Eeles P.* Characteristics of a Software Architect. The Rational Edge, IBM Resource, 2006: «Если использовать в качестве аналогии киноиндустрию, то руководитель проекта является продюсером (следит за тем, чтобы все было сделано), а архитектор — режиссером (следит, чтобы все было сделано правильно)».

Подобную роль в программном проекте должен играть архитектор программного обеспечения: быть режиссером с абсолютными полномочиями и ответственностью. Если что-то пойдет не так, это личная вина архитектора. Всему, что говорит архитектор, мы беспрекословно подчиняемся. Эта роль весьма авторитарна, но так и должно быть.

В большинстве команд в настоящее время играют в техническую демократию, принимая решения коллегиально, после долгих обсуждений и стояний у доски. Это приводит только к хаосу и ни к чему более. Группа людей не может создать ничего, если нет лидера, диктатора, вождя, автора. Посмотрите на компанию Apple во времена Стива Джобса и посмотрите на нее сейчас. Я думаю, что Стива многие коллеги обвиняли в «грубости»[1] по одной простой причине — он был сильным и ярким режиссером продуктов Apple. Он брал на себя полную ответственность за них и хотел иметь возможность принимать любое решение, которое, по его мнению, будет правильным, независимо от того, насколько такое решение не понравится окружающим.

> Если вы хороший архитектор программного обеспечения, будьте готовы к ненависти коллег.

Хорошие архитекторы программного обеспечения создают хорошие программные продукты, но они также наживают множество врагов, если увлечены своим делом, а не только своей зарплатой (как и хорошие режиссеры или любые другие профессионалы).

Конечно, недостаточно быть сильным и иметь врагов, чтобы создать высокопроизводительный сайт или оскароносный фильм. Нужно также разбираться в сути программирования или видеосъемки. Конечно, архитектор не может знать все о разрабатываемом

---

[1] *Isaacson W.* Steve Jobs. Simon & Schuster, 2011: «Если честно, я думаю, что, когда он очень расстроен, то пытается избавиться от этого чувства, обидев кого-то. И я думаю, он считает, что у него есть на это право. Он думает, что правила, касающиеся социальных обязательств, к нему не относятся. Поскольку он очень чувствителен, он точно знает, как обидеть кого-то наверняка. Это он и делает».

продукте. Для этого и нужна команда: помочь архитектору заполнить пробелы в знаниях и сделать то, чего он сделать не может, потому что у него есть только 24 часа в сутках. Умный архитектор знает, как правильно использовать команду, чтобы она произвела как можно больше[1]. Или, быть может, здесь слово «использовать» звучит оскорбительно?

Совещания, записи на доске, общение по электронной почте, в чатах, экспериментирование и многие другие методы должны использоваться для сбора информации и выяснения мнения команды. Архитектор должен организовывать и поддерживать членов команды, но окончательное решение всегда за ним. В разработке программного обеспечения нет места демократии. Это должна быть диктатура, и точка.

Сильный диктатор всегда лучше, чем демократия. Диктатор может ошибаться и будет делать это очень часто, но он заплатит за все промахи[2]. И он знает, какую цену. Ошибки — неизбежная часть любого творческого процесса. Вопрос в том, кто ошибается и какую ответственность несет. Групповая ответственность — это вообще не ответственность. Важна именно индивидуальная ответственность.

**Пока архитектора не уволят, у него должны быть все полномочия для принятия любых технических решений, независимо от того, что говорит команда или начальник.**

---

[1] *Zhou Y.* Take Responsibility for Your Decisions в книге ★ *Monson-Haefel R.* 97 Things Every Software Architect Should Know: Collective Wisdom from the Experts. O'Reilly Media, 2009: «Многие архитекторы ошибочно полагают, что они должны принимать каждое архитектурное решение. Поэтому они считают себя всезнайками. На самом деле универсального технического гения не существует. Какого архитектора ни возьми, есть области, в которых он будет достаточно хорошо разбираться, и области, в которых он окажется некомпетентен. Архитекторы делегируют команде решения в тех сферах, в которых они не специализируются».

[2] *Brown S.* Are You a Software Architect? February 2010. https://goo.gl/nSBTW6: «Существует большая разница между вкладом в архитектуру программной системы и личной ответственностью за ее самостоятельное определение».

Проект должен быть готов заменить архитектора, если он начинает совершать слишком много ошибок. Кроме того, для архитектора должны быть предусмотрены санкции, которые скорректируют его поведение, пока не стало слишком поздно. Как именно это будет реализовано, зависит от проекта, его политики компенсации и от того, какой риск допустим. Но отсутствие архитектора и вера в способность команды принимать правильные решения в демократическом режиме — это самоубийственный путь для проекта[1]. Даже худший архитектор лучше, чем его отсутствие[2] (в крайнем случае его можно уволить и заодно преподать урок другим). Если нет архитектора — никто никакого урока не усвоит. Единственный урок, который мы вынесем, — никакого порядка здесь нет, мы можем сидеть здесь до тех пор, пока у этих толстосумов есть деньги нам на зарплату. Действительно ли это плохо для нас, программистов на окладе, — отдельный вопрос.

---

[1] *Bass L. et al.* Software Architecture in Practice. 2nd Edition, Addison-Wesley Professional, 2003: «Архитектура должна быть продуктом одного архитектора или небольшой группы архитекторов с конкретным лидером».

[2] *McBride M. R.* The Software Architect. Communications of the ACM, Volume 50, Number 5, 2007: «Без строгого контроля со стороны архитектора программного обеспечения проекты и принимаемые решения имеют тенденцию разваливаться из-за увеличивающейся сложности».

# Проигравшие и победители

Деннис снимает наушники и поворачивается ко мне.

— Чувак, ты обещал дать мне совет, как заработать кучу денег, — хихикает он.

— Деньги тебе не помогут. Чтобы быть счастливым, нужна власть, а не деньги. Деньги — это просто бензин в твоей машине, а не цель, к которой ты стремишься. Знаешь, что сказал об этом Рокфеллер?[1] Он сказал, что если ваша цель — просто стать богатым, то вы никогда ее не достигнете. Думаю, он имел в виду, что если ты просто хочешь получить полный бак бензина, то никогда не выиграешь гонку. В гонках цель — кубок, а не топливо.

— Чувак, ну ты же помнишь, что я хочу продавать клубнику? — смеется он.

— На самом деле ты не хочешь ее продавать. Ты просто продолжаешь рассказывать себе эту историю, чтобы объяснить, почему ты несчастен и вынужден сидеть за компьютером десять часов в день, разрабатывая какой-то Java-код для каких-то капиталистов. Без этой клубничной истории ты бы, наверное, уже повесился в туалете. Хочешь знать, почему ты несчастен?

**Бедные люди зарабатывают деньги, богатые отнимают деньги у других.**

— Почему? — он перестает веселиться.

— Потому что у тебя нет гена победителя.

— Что?

— Послушай, не обижайся, я расскажу тебе кое-что, чего никогда не пишут во всех этих книгах о том, как разбогатеть[2]. Все люди разные с рождения. Даже Аристотель сказал[3], что одни

---

[1] *Braude J. M. et al.* Complete Speaker's and Toastmaster's Library. Prentice Hall, 1992.

[2] *Ferriss T.* Tools of Titans: The Tactics, Routines, and Habits of Billionaires, Icons, and World-Class Performers. Houghton Mifflin Harcourt, 2016.

[3] *Аристотель.* Политика, часть V, 350 до н. э.: «Очевидно, во всяком случае, что одни люди по природе свободны, другие — рабы, и этим последним быть рабами и полезно, и справедливо».

из нас рождены, чтобы быть победителями и отдавать приказы, а другие — чтобы работать на победителей. Врожденное это качество или развивается в течение первых нескольких лет нашей жизни, я на самом деле не знаю. Но знаю, что ключевое различие между победителями и проигравшими заключается в их отношении к деньгам, собственности, власти, результатам, успеху и т. д. Проигравшие знают, что им нужно много работать, чтобы получить все это, но они никогда не достигают ничего значительного. Это то, чему нас учат книги: усердно трудитесь, будьте сосредоточенны, просыпайтесь в шесть утра, не пейте, не курите, помогайте другим, экономьте каждый доллар — и вуаля, в конечном итоге вы станете богатыми. Правда[1] в том, что богатые люди так себя не ведут. Они пьют, избивают жен, спускают деньги в Вегасе, редко появляются в офисе, лгут своим партнерам и сотрудникам, обманывают и воруют. Некоторые из них даже не платят налоги, — надеюсь, он понял сарказм. — Они не делают ничего из того, о чем пишется в книгах. Они ведут себя совершенно противоположным образом, но при этом выигрывают. Потому что они не неудачники. Победители, в отличие от проигравших, считают, что все уже принадлежит им. Им нужно только взять.

Он встает и щелкает пальцами:

— Вот это — «мир принадлежит мне», да?

Полагаю, Деннис ссылается на Тони Монтану[2].

— Да, чувак, именно это. Мир уже принадлежит тебе, если у тебя есть ген победителя. Некоторые люди рождаются такими. Другие — большинство из нас — нет. Мы с тобой, очевидно, неудачники. Сколь усердно бы мы ни работали всю жизнь — мы никогда не станем по-настоящему богатыми. Потому что мы верим в то, что должны заслужить богатство.

— Но как еще получить деньги, если не заработать их? Ограбить банк?

---

[1] *Greene R.* The 48 Laws of Power. Penguin Books, 2000.

[2] Scarface (1983) by Brian De Palma.

— Ты не понял. Посмотри на таких кумиров компьютерного бизнеса, как Билл Гейтс или Ларри Эллисон. Они зарабатывают несколько тысяч долларов в минуту. Ты думаешь, они в миллион раз умнее нас с тобой? Да ни в коем случае. Они обычные люди, не волшебники, у них нет волшебной палочки. Они так же чистят зубы каждое утро, как я и ты, они ленятся, смотрят телевизор, едят, занимаются сексом и ходят в туалет. Но по какой-то причине в год они зарабатывают в несколько тысяч раз больше, чем ты. Как думаешь, в чем причина? Чистое везение? Их необычайная работоспособность мозга? Их исключительный талант? Может быть, но не в такой же степени. Тогда что у них есть из того, чего нет у нас с тобой? Как думаешь, вот ты мог бы догнать их, просто написав код на Java быстрее и лучше?

**Сама вера в то, что ресурсы в мире распределены справедливо, делает вас бедным.**

— Хм...

— Да, чувак, — я делаю паузу на секунду. — Ты не можешь. Ты уже проиграл, когда родился. Ты родился для того, чтобы работать, а они — для того, чтобы иметь. В этом разница. Это наша ошибка: мы верим в справедливость, а они верят во власть. И мир основан на власти, а не на справедливости. Ты будешь усердно работать, изучать новые технологии, ты даже будешь создавать собственные стартапы, получать хорошую зарплату, но никогда не станешь по-настоящему успешным и богатым. Потому что ты веришь[1] в справедливость утверждения, что для того, чтобы что-то получить, необходимо вкалывать. Учти, когда я говорю «ты», я имею в виду и себя тоже. Я в том же положении, что и ты, друг мой, — грустно улыбаюсь я.

---

[1] *McNamee S. J. et al.* The Meritocracy Myth. 3rd Edition, Rowman & Littlefield Publishers, 2013: «Некоторым недостаточно просто иметь больше, чем имеют другие. Для того чтобы система неравенства была стабильной в долгосрочной перспективе, имущие должны убедить малоимущих о том, что такое распределение справедливо или является естественным порядком вещей. Чем выше уровень неравенства, тем более убедительными и неопровержимыми должны быть эти объяснения».

— Да я понимаю, — говорит Деннис, — не беспокойся, продолжай. Какое же есть решение?

Я УЛЫБАЮСЬ:

— А вот решения нет. Я бы уже стал миллионером, если бы знал секрет. Как ты думаешь, я менее талантлив, чем Билл Гейтс? Отнюдь. Я просто лузер. Может быть, талантливый, — вздыхаю я. — Талант, образование и интеллект не помогут тебе разбогатеть. Более того, они обычно оказываются препятствиями. Чем ты умнее, тем очевиднее, что у тебя есть пробелы в знаниях, что тебе нужно учиться чему-то другому, тебе нужно работать усерднее, исправить свои недостатки и т. д. И в то время, пока ты все это делаешь, менее образованные, менее умные, менее талантливые люди побеждают. Я бы сказал так: победители видят в работе препятствие между ними и призом, а проигравшие — инструмент для получения приза. Это типичные симптомы лузера: ему нравится работать, нравится быть занятым и уставшим, у него много вопросов на повестке дня, он планирует свое рабочее время.

— Ты меня совсем расстроил. Теперь я буду плохо спать, — вздыхает он.

**Тяжелая работа не ведет к успеху, а чаще всего становится препятствием на пути к нему.**

— Что ж, сожалею. Слушай, я скажу тебе кое-что еще. Я слышал эту теорию несколько лет назад[1]. Допустим, мы берем все деньги мира и делим их поровну между всеми людьми на этой планете. Каждый из нас получит где-то около 10 тысяч долларов[2]. А вот что произойдет дальше: очень скоро деньги будут накапливаться в руках тех же самых людей, которые ими владеют сейчас. Конечно, это теория, но я считаю ее верной. Толстосумы очень быстро вернут свои деньги, пока мы с тобой начнем строить бизнес-планы и делать разумные инвестиции. Я не говорю, что бизнес-планы не нужны, но мы им проиграем. По разным причинам. Они победят — тоже по другим причинам. Основной причиной,

---

1 *Anthony R.* Advanced Formula for Total Success. Berkley, 1988.

2 CIA, The World Factbook. https://goo.gl/EEY6cQ.

конечно же, будет то, что они уже богаты и никогда не сомневаются в этом, в то время как мы ожидаем, что мир вознаградит нас деньгами, если мы будем все делать правильно.

— Я понял, — перебивает он. — Так решения все же нет?

— Ну, ты можешь попытаться изменить себя, но, скорее всего, это не сработает. По-твоему, делать то, что делают богачи, плохо. Ты хочешь, чтобы мир был справедливым, и для тебя всегда будет важнее делать что-то хорошее, а не то, что приносит прибыль. И я не говорю об ограблении банка. Я говорю о наших фундаментальных моральных убеждениях, которые превращают нас в лузеров.

Деннис уставился на меня.

— То есть, чтобы стать богатым, я должен быть плохим?

— Ну, не совсем, хотя, может, и так. Я уже говорил тебе, что стремление к деньгам не поможет, потому что это ложная цель. Победители на самом деле не стремятся заработать деньги, они стремятся к власти[1]. Они хотят быть лучше других, быть на вершине, побеждать и доминировать. Ты не хочешь проиграть, поэтому тебе нужны деньги. Видишь разницу между «выиграть» и «не проиграть»?

— Думаю, да, — он выглядит очень озадаченным. — Нужно ли мне быть плохим парнем, чтобы стать богатым? Что нового в этой идее, чувак?

— Это та самая проблема, о которой я говорю. Ты негативно относишься к идее власти. Ты веришь, что быть наверху и побеждать — это зло. Ты не хочешь принимать участие в этой игре, ты думаешь, что ты лучше, что ты хороший человек и все такое. На самом деле ты просто искалечен современной моралью, как сказал бы Ницше[2]. Твои

**Быть хорошим и честным человеком и быть богатым и успешным — противоречащие друг другу цели.**

---

[1] *Tomassi R.* The Rational Male. Amazon, 2013: «Определение власти — это не финансовый успех, статус или влияние на других, а степень, в которой мы контролируем нашу собственную жизнь».

[2] ★ *Nietzsche F.* Beyond Good and Evil. 1886.

фундаментальные убеждения неверны, и я не думаю, что их можно быстро изменить. Благодаря христианскому воспитанию[1] ты веришь, что те, кто относится к другим справедливо и с уважением, — хорошие люди и заслуживают быть королями мира. В действительности все наоборот. Эгоизм, хладнокровие, жадность, нечестность и лживость — качества королей мира. Ты знаешь, что Макиавелли сказал[2]: тот, кто держит слово, не заслуживает того, чтобы быть королем?

— Во-первых, я атеист. Во-вторых, ты просто зол, потому что у тебя нет девушки, — смеется он. — Возможно, мне пора повеситься в туалете.

---

[1] ★ *Nietzsche F.* The Antichrist. 1895.

[2] См. сноску 3 на с. 98, глава 18, 1532: «Разумный правитель не может и не должен оставаться верным своему обещанию, если это вредит его интересам и если отпали причины, побудившие его дать обещание».

# Хакеры и дизайнеры

Когда я изучал программирование в университете, почти никто даже не упоминал об управлении разработкой программного обеспечения. Хотя какие-то лекции по разработке программного обеспечения были; нам рассказывали об Agile и экстремальном программировании, и мы немного работали в группах, но это всегда было на втором месте после теории алгоритмов[1], булевой алгебры[2], теории отношений[3], теории конечных автоматов[4] и других вещей, о которых я, честно говоря, сейчас уже и не помню. Мне кажется, что значительная часть этой информации была нам не нужна, так как теперь я не пользуюсь ею.

Кроме того, кому сейчас нужны разработчики алгоритмов? Ну, есть парочка научных организаций и стартапов Кремниевой долины, которые изобретают какие-то новые методы шифрования или высокоскоростные протоколы, но сколько их там? Из миллионов программистов в мире большинство просто зарабатывают на жизнь, «склеивая» уже имеющиеся компоненты с открытым исходным кодом. Им нужно понимать архитектуру и дизайн, знать, как правильно выполнять сборку, как избегать антипаттернов,

**Большинство программистов уже не ученые и даже не инженеры. Они специалисты, как сантехники или хирурги.**

---

1 *Cormen T. H. et al.* Introduction to Algorithms. 2nd Edition, The MIT Press, 2001

2 *Whitesitt J. E.* Boolean Algebra and Its Applications. Dover Publications, 2012.

3 *Date C. J.* An Introduction to Database Systems. 8th Edition, Pearson, 2003.

4 *Carroll J. et al.* Theory of Finite Automata with an Introduction to Formal Languages. Prentice Hall, 1989.

но куда важнее знать, как работать с людьми, а не с байтами и битами.

Университеты и школы должны не только учить нас кодировать, но и делать так, чтобы наши результаты можно было поддерживать. И поддерживать не только своими силами, но и силами других программистов, которые будут работать после нас или трудятся вместе с нами. Иными словами, сделать наш код читабельным для людей, а не только для компьютеров. В то время, когда вышел первый том Кнута[1], компьютеры были очень дорогими, а программисты были сумасшедшими фанатами с недооцененной зарплатой. Это была эпоха «хакинга», хотя в то время это слово имело негативный смысл. Программисты составляли небольшую элитную группу профессионалов, которые говорили на своем особом языке и принимали душ раз в неделю. Их основной задачей было выяснить, как научить этот тупой и медленный компьютер делать то, что они хотели. Во многих случаях они это делали впервые в истории человечества. Это была эпоха программных и аппаратных открытий. Меня тогда еще даже не было.

В эпоху хакеров[2] компьютеры были редкими, дорогими и экзотическими игрушками, к которым бизнесмены обычно не относились серьезно. Чтобы выжить, программисты должны были создать программное и аппаратное обеспечение, которое бы работало достаточно быстро, именно это являлось ключевым критерием успеха. Это было время таких языков, как Algol, Fortran, C, Assembly, Perl, довольно загадочных и требовавших серьезных умственных способностей для создания чего-то приличного или для понимания того,

---

[1] *Knuth D. E.* The Art of Computer Programming, Volume 1: Fundamental Algorithms. 3rd Edition, Addison-Wesley Professional, 1997.

[2] *Levy S.* Hackers: Heroes of the Computer Revolution. O'Reilly Media, 25th Edition, 2010: «Наиболее эффективные сотрудники, работающие в S&P, с большой гордостью называли себя хакерами».

что создал другой программист[1]. Требовалось точно знать, как функционировало аппаратное обеспечение, чтобы создать быстро работающее программное обеспечение.

Кроме того, тогда не было такого количества проектов с открытым исходным кодом. Большинство программных пакетов были коммерческими, люди работали в маленьких помещениях, подолгу оставаясь в одной компании. Они хорошо знали свои продукты и боялись потерять работу. Но за последнее десятилетие ситуация резко изменилась.

Прежде всего, стоимость вычислительной мощности с каждым годом падает. Например, 1 Гбайт памяти компьютера в 2000 году

---

[1] См. сноску на с. 84: «В зрелых технологических средах — таких как среда веб-программирования в середине 2000-х — нам доступны все достоинства богатой инфраструктуры разработки ПО. Такие среды предоставляют широкий выбор языков программирования, мощные средства поиска ошибок, эффективные инструменты отладки и надежные автоматизированные средства оптимизации производительности приложений. Компиляторы почти не содержат ошибок. Инструменты хорошо описаны в документации производителей, в книгах и статьях и на многочисленных сайтах. Инструменты интегрированы, благодаря чему вы можете разрабатывать UI, модули работы с БД, составления отчетов и бизнес-логики в одной среде. Решения проблем можно легко найти в ответах на "часто задаваемые вопросы". Кроме того, доступны разнообразные услуги консультантов и программы тренинга.

В ранних средах — таких как веб-программирование в середине 1990-х — ситуация противоположная. Языков программирования мало, при этом они часто полны ошибок и плохо документированы. Вместо написания нового кода программисты тратят кучу времени только на то, чтобы разобраться в особенностях языка. Бесчисленные часы уходят на борьбу с ошибками в языках, ОС и других инструментах. Инструменты программирования часто примитивны. Отладчиков может не быть вообще, а об оптимизаторах компиляторов программистам приходится лишь мечтать. Производители часто выпускают новые версии компиляторов, при этом очередная версия отказывается поддерживать значительные части вашего кода. Инструменты не интегрированы, из-за чего UI, модули работы с БД, составления отчетов и бизнес-логики приходится разрабатывать с помощью разных средств. Из-за плохой совместимости инструментов и частого появления новых компиляторов и библиотек программисты тратят много усилий только на поддержание работоспособности имеющейся инфраструктуры».

стоил около 1000 долларов. Теперь его цена ниже пяти долларов. Цена упала в 200 раз всего за 17 лет. То же самое верно и для жестких дисков, мониторов, процессоров и всех других аппаратных ресурсов. Проще говоря, с каждым годом компьютеры становятся все более доступными.

Во-вторых, рост программ с открытым исходным кодом огромен. Большая часть программного обеспечения теперь доступна бесплатно вместе с ее исходным кодом, включая операционные системы, графические процессоры, компиляторы, редакторы, платформы, инструменты криптографии и все, что вы можете себе представить. Программистам больше не требуется писать много кода, им просто нужно уметь собрать уже имеющиеся компоненты[1].

В-третьих, количество программистов в мире продолжает расти и при этом они все еще в дефиците. В некоторых европейских странах спрос на квалифицированный ИТ-персонал вдвое выше предложения, которое могут обеспечить их рынки[2]. Программисты уже не элитная группа гиков. Профессия превратилась в товар. Люди становятся программистами просто потому, что им нужна работа, чтобы прокормить свои семьи, тогда как много лет назад это были в основном хобби или наука.

**Поскольку индустрия вычислений сейчас существенно отличается от той, что была 30–40 лет назад, ей нужны разные виды человеческих ресурсов.**

В-четвертых, языки программирования сейчас намного более высокого уровня, чем раньше. Они менее связаны с оборудовани-

---

[1] *Raymond E. S.* The Cathedral and the Bazaar. Knowledge, Technology & Policy, Volume 12, Number 3, 1999: «Хорошие программисты знают, что написать. Великие знают, что переписать (и повторно использовать). ...Важной чертой великих людей является конструктивная лень. Они знают, что вы получаете высший балл не за усилия, а за результаты и что почти всегда легче начать с хорошего частичного решения, чем с нуля».

[2] *McCallum E.* Gaps in Dutch Labour Market: ICT, Tech and Sales Skills in Demand. Iamexpat.nl, July 2015: «Существует большой спрос на работников ИКТ с техническим образованием. Около 76 % сотрудников отделов кадров сообщили, что им трудно найти необходимых кандидатов с такой квалификацией».

ем, чем 40 лет назад. Программистам не нужно больше знать, как работает оборудование. Они могут кодировать на JavaScript, даже не задумываясь о таких вещах, как управление памятью, стеки, указатели или регистры процессора. Недавно мой друг рассказал мне о собеседовании, которое он провалил. Он довольно неплохой фронтенд-разработчик, в его портфолио много проектов на Ruby и JavaScript. Он с негодованием сообщил, что на собеседовании его попросили вычислить $2^8$. «Ну откуда мне это знать?» — кричал он мне. Я был удивлен, но потом понял, что, возможно, он был прав в своем разочаровании. Нужно ли фронтенд-разработчикам понимать двоичный код? Кстати, ему 22 года и у него нет никакого формального компьютерного образования. Возможно, он не знает, что такое байт, но его проекты работают и люди платят ему 80 долларов в час, а то и больше за графический дизайн и кодирование на Ruby. Суть этой истории в том, что для того, чтобы в наши дни стать успешным программистом, не обязательно знать, как работает аппаратное обеспечение.

В-пятых, программисты теперь работают удаленно, а не в офисах. Благодаря развитию высокоскоростного Интернета, программного обеспечения для проведения конференций, средств обмена сообщениями и распределенных систем управления репозиториями, таких как Git, наряду со многими другими инновациями удаленная работа стала удобнее, чем работа в традиционных офисных условиях. Несмотря на всю критику и проблемы[1], процент людей, работающих удаленно, растет с каждым годом[2].

---

[1] *Felstead A. et al.* Assessing the Growth of Remote Working and its Consequences for Effort, Well-being and Work-life Balance. New Technology, Work and Employment, Volume 32, Number 3, 2017: «Несмотря на эти недостатки и некоторые перемены со стороны работодателей, представленные данные свидетельствуют о том, что удаленная работа в целом выгодна и работодателям, и работникам. Это также говорит о том, что, хотя мы, видимо, не станем свидетелями полноценной революции, отсутствие привязанности к рабочему месту, несомненно, становится важной характеристикой работы в XXI веке».

[2] *Vanderkam L.* Will Half of People Be Working Remotely By 2020? FastCompany, August 2014: «В ходе опроса лидеров бизнеса на Саммите глобального лидерства в Лондоне 34 % респондентов указали, что более половины штатных сотрудников их компаний к 2020 году будут работать удаленно».

Наконец, шестое изменение — это зарплаты программистов, которые взлетели до небес за последние несколько десятилетий. В 2000 году, когда 1 Гбайт памяти стоил 1000 долларов, старшие программисты в Кремниевой долине зарабатывали 80 тысяч долларов в год. Сейчас, в 2017-м, они зарабатывают в три раза больше, а оперативная память стоит в 200 раз дешевле. Работа программистов сегодня намного дороже компьютеров.

Если мы объединим все шесть тенденций, то станет очевидным, что наиболее важное качество хорошего программного обеспечения в настоящее время — удобство его поддержки. Наступает эра дизайнеров, когда быстродействие или оптимальность алгоритма не так важны, как то, насколько легко его понимают коллеги-программисты[1]. Компания тратит много денег на очень дорогой человеческий ресурс и вполне может позволить себе несколько новых серверов или более мощный мобильный телефон для решения проблем, связанных с производительностью не самого совершенного алгоритма.

Конечно, есть области, где хакеры все еще необходимы. Нам по-прежнему нужны некоторые алгоритмы, нам нужно летать в космос, изобретать новых роботов, но это лишь малая часть всех программных продуктов, которые разрабатываются в мире. Большинство программного обеспечения не требует наличия сложных навыков.

> **Самое важное качество хорошего программиста — умение создавать простые и понятные решения.**

Однако необходим другой набор навыков. Программисты должны уметь создавать код — читаемый, тестируемый, задокументированный, тот, который не только работает, но и объясняет сам себя.

Более того, программисты должны уметь поддерживать среду вокруг кода в чистоте и порядке. Теперь важен не только код, но и отчеты об ошибках, требования, запросы на включение, сценарии развертывания, журналы и т. д. Сам по себе код — лишь малая

---

[1] *Bugayenko Y.* We are Done with 'Hacking'. Communications of the ACM, Volume 61, Number 7, 2018: «По всей видимости, будущее программирования не столько в математике, сколько в социотехнических отношениях между людьми».

часть того, что мы сейчас называем программным продуктом. Я полагаю, Брукс[1] сказал что-то подобное много лет назад. Путь от работающего кода до программного продукта, который можно выпустить на рынке, довольно долог.

Программный продукт — это живой организм, среда, экосистема, территория с ее законами, местами обитания, союзниками и врагами — используйте любую метафору, которая вам нравится. Очень важно понять, что код сам по себе — лишь элемент программного продукта. Хакеры, предпочитающие алгоритмы, не поддерживают этот подход. Они не придерживаются правил и норм, без уважения относятся к руководителям и младшим программистам. Они любят писать то, что расшифровать и понять смогут только очень серьезные разработчики, не создают автоматических тестов и предпочитают говорить, а не составлять документацию.

**Хакерам нравится важничать и владеть кодом, а дизайнерам нравится, что их кодом делятся.**

У дизайнеров, людей будущего, отношение полностью противоположное. Они стремятся использовать простые структуры кодирования, которые легче понять. Им нравится видеть, как их код совершенствуется другими программистами, они пишут автоматические тесты для поддержки будущих разработчиков, приветствуют начинающих программистов и тестируют их код, видя, сколько вопросов задает новичок при его использовании (они предпочитают документацию встречам и телефонным звонкам).

Ключевое различие в том, что хакерам нравится владеть кодом, а дизайнеры воспринимают себя как заменяемый компонент программного проекта, который не владеет ничем, кроме своих ноутбуков. Для хакера разработчик — король, а дизайнер понимает, что король — это код.

Очевидно, что представление о себе как о заменяемом ресурсе серьезно угрожает безопасности работы дизайнера — его легко

---

[1] *Brooks F.* The Mythical Man-Month. Addison-Wesley, 1975: «Программный продукт стоит как минимум в три раза дороже отлаженной программы с той же функциональностью».

заменить, если проект настолько хорошо организован, что не зависит от каких-либо конкретных экспертов, все задокументировано, регламентировано, а код — читабельный и поддерживаемый. Так что позиция программиста действительно становится шаткой.

В будущем разработки программного обеспечения не будет «рабочих мест» на полный рабочий день, а будут только проекты, к которым разработчики станут подключаться тогда, когда понадобятся их навыки, и из которых будут уходить, когда их миссия окажется выполнена. Не будет организаций или специализированных бодишопов с программистами, годами делающими то, что нужно фирме, только потому, что они регулярно получают за это зарплату.

Вместо этого будут программные проекты (или, может быть, не только программные?), которые станут собирать свои команды по запросу и платить людям с учетом их вклада. Взаимозаменяемость станет одним из ключевых достоинств разработчика программного обеспечения.

Хакеры — неуправляемый и опасный ресурс, от которого критически зависит проект, потому что код сложный, трудночитаемый, неоднородный и... высокомерный. Умные руководители проектов постараются максимально избавиться от хакеров, это определенно станет тенденцией будущего. С другой стороны, дизайнеры, в отличие от хакеров, — эксперты в работе с людьми, а не только с компьютерами. Они понимают важность создания хороших команд, а не просто правильного написания кода.

> **Программисты будущего будут знать: лучшее, что они могут сделать для своих проектов, — это сделать их легко заменяемыми.**

Программисты должны научиться заранее думать о том, что произойдет с их кодом, когда они покинут проект, или кто-то новый подключится к нему, или потребуются изменения, или компания решит открыть исходники продукта для общественности, или случится что-то еще. Проще говоря, они должны «кодировать наперед».

# Глава 3

# Тони

Политика — вот что такое современный менеджмент. Речь идет не о числах, планах, показателях, результатах или сценариях развертывания. Речь об эмоциях, страхах, слезах и борьбе за власть. Люди слабы и неорганизованны, глупы и неуверенны. По крайней мере большинство. Некоторые из нас сильнее, дисциплинированнее, увереннее в себе и агрессивнее. Они лезут друг на друга, образуя социальные иерархии. Компании по разработке программного обеспечения, создающие программные продукты, — не что иное, как социальные иерархии, украшенные причудливыми досками Kanban и наклейками для ноутбуков, сдобренные мотивационными речами и лозунгами о равенстве. Однако основной принцип остается тем же — более сильные животные порабощают более слабых. Но так ли должно быть в эпоху роботов и космических путешествий?

## Показатели производительности

Я возвращаюсь к своему столу. Деннис и Бао о чем-то спорят. Бао стоит, Деннис сидит, а Адриан развалился в кресле. Я люблю офисные споры, — это всегда весело и можно узнать что-то новое.

— Ну а как это будет работать? — спрашивает Адриан у Денниса, когда я сажусь на стул.

— Точно еще не знаю, но мы можем что-нибудь придумать.

Похоже, Деннис растерялся и разочарован.

— Чувак, что ты думаешь о метриках? — спрашивает он меня.

— Думаю, они не помешают, если в них есть смысл.

— Было бы здорово, если бы у нас были показатели производительности для нашей работы, верно? — говорит Деннис.

— Абсолютно. Но для чего?

— Вот и я думаю точно так же! — восклицает Бао. — Для чего они нам нужны? Чем они помогут? Они будут только отвлекать и создавать ненужное напряжение между нами всеми![1]

**Эффективное управление может основываться только на фактах, цифрах и показателях и ни на чем другом.**

— А напряжение — это плохо? — спрашиваю я.

— Ну да, — кажется, Бао озадачен. — Что ты имеешь в виду? Кто захочет работать в условиях стресса?

— Ну, некоторые люди работают. Некоторым людям даже трудно работать без стресса и давления.

— Ты, наверное, шутишь, а я говорю серьезно. Это не соревнование! Нам не нужны никакие метрики. Нам нужна комфортная рабочая среда, где мы сможем быть творческими и продуктивными. Метрики и все эти методы кнута и пряника подходят для

---

[1] *Dreu De C. K. W. et al.* Task Versus Relationship Conflict, Team Performance and Team Member Satisfaction: A Meta-Analysis. Journal of Applied Psychology, Volume 88, Number 4, 2003: «Небольшой конфликт стимулирует обработку информации, но при эскалации конфликта когнитивная система отключается, обработка информации затрудняется, а производительность команды, как правило, падает».

более примитивного труда. Программирование — творческий процесс.

Кажется, он гордится тем, что относится к числу таких творческих программистов.

— Некоторые могут даже сказать, что это искусство. Как ты можешь заставить художников рисовать картины под давлением каких-то показателей? Одна картина в месяц или тебя увольняют?

Все смеются.

— Согласен, — говорю я, — но независимо от того, насколько ты творческий человек, показатели все равно есть. Они есть даже у лучших художников. Они есть и у нас.

— Что ты имеешь в виду?

— Смотри, мы получаем зарплату каждые две недели, верно? Вот, это уже показатель. Если я перестану работать или мой руководитель, — я смотрю на Адриана, который рисует, опустив голову, — недоволен моими результатами, я перестану получать зарплату. Это показатель. Очень примитивный, но все равно показатель: я либо получаю зарплату, либо нет. У художников и артистов аналогично. Они рисуют, они поют, они создают скульптуры, но кто-то в конечном итоге платит им за эту работу. И если они производят то, что никому не нравится, значение их показателей будет отрицательным. Отдельный вопрос, обратят ли они внимание на такое значение или нет, но показатель — вот он, перед ними.

— Полностью согласен! — восклицает Деннис.

— Может быть, и так, — говорит Бао после нескольких секунд раздумий, — и что ты по этому поводу думаешь?

**Ежемесячная зарплата — это показатель производительности, но довольно грубый, примитивный и неэффективный.**

— Я хочу сказать, что этот показатель очень грубый. Он слишком примитивный, упрощенный и даже оскорбительный. И он совершенно не помогает людям, особенно творческим. Вообще никому не помогает. Он превращает нас в рабов, потому что мы работаем не на результат, а на хозяина. Вместо того чтобы достичь каких-то измеримых и объективных целей, мы должны угодить человеку, который платит зарплату.

— Чувак, прекрати, это действительно рабство, — злится Бао.

— Не пойми меня неправильно, я никого не виню. Просто вся система так работает. Виноваты не Адриан или Тони, — Адриан грустно на меня смотрит. — Отнюдь! Они в том же положении, что и мы. У них тоже есть хозяева. Они также должны работать на кого-то, а не для достижения каких-то целей. Мы все виноваты, потому что слишком ленивы или слишком глупы, чтобы придумывать более эффективные показатели, которые заставят нас работать на проекты, а не на людей.

— Именно это я им и говорил, чувак, — встает Деннис. — Давайте создадим какие-то показатели и посмотрим, как они работают.

— Какие показатели? Успокойся! — восклицает Бао. — Или вы планируете считать мои строки кода?

Интересно, написал ли он хоть пару строк за последние несколько лет?

— Почему каждый раз, когда я упоминаю показатели, все думают о строках кода? После этой шутки Билла Гейтса[1], я думаю, все знают, что подсчет строк — в корне неправильная идея.

— Что еще вы предлагаете посчитать? — спрашивает Бао.

— Ну давайте подумаем о чем-нибудь... — я секунду смотрю на Денниса. — Как насчет э-э-э... количества закрытых тикетов в неделю?

**Любой показатель лучше, чем его отсутствие, при условии, что их формулы и результаты регулярно пересматриваются.**

— Точно, — подтверждает Деннис.

— Так что, получается, если я не закрываю тикеты, то я плохой программист? — похоже, идея не нравится Бао уже потому, что он никогда ничего не закрывает.

— Ну, не совсем так, но это что-то нам скажет. Измерение не означает немедленных действий. Сначала важно понять, что обозначают цифры, а затем снова сядем вместе и решим, что с ними делать. В любом случае почему ты так боишься показателей?

---

[1] Цитата выглядит примерно так, хотя я не смог найти источник: «Измерение прогресса в программировании по строкам кода похоже на измерение прогресса в самолетостроении по весу самолета».

— Да потому, что они контрпродуктивны! Прочитай Майка Кона[1] или любого другого автора по Agile. Все они утверждают, что ответственность за скорость поставки или любые другие цифры убивают их мотивацию, — говорит Бао.

Я делаю паузу, прикидывая, как их убедить и с чего начать.

— Людям не нравится испытывать стресс, как ты уже сказал. Я согласен с этим. Никому не по душе жить и работать в страхе. И стресс вызывает страх, если ты не знаешь, откуда он взялся и когда закончится. Иными словами, если ты не знаешь правил игры, это очень напряженная для тебя ситуация. С другой стороны, если правила ясны, даже если ты проиграешь, игра тебе понравится. Офисная жизнь — это еще одна игра, в которую мы играем. Когда правила четко определены и мы точно знаем, что нужно сделать, чтобы выиграть, мы наслаждаемся игрой и становимся очень продуктивными, мотивированными, эффективными. Однако если у нас есть одно-единственное правило: «Обрадуй своего босса или вылетишь из игры», игра превращается в кошмар из-за непредсказуемости и неопределенности[2]. Показатели и объективные достижимые цели абсолютно ясны, и неважно, насколько они неэффективны. Даже просто тупо считать строки кода лучше, чем радовать босса. Естественно, в конечном итоге программисты начнут писать дополнительные стро-

---

[1] *Cohn M.* Succeeding with Agile: Software Development Using Scrum. Addison-Wesley Professional, 2009.

[2] См. сноску 5 на с. 126: «Авторитаризм, будь то межличностный (деспотический) или институциональный (тоталитарный) по форме, вызывает чувство солидарности и свободы (от ответственности) посредством систематического подавления неопределенности и амбивалентности. В идеальной типичной бюрократической организации, управляемой правилами, сотрудникам по крайней мере разрешается думать, что им нравится, если они действуют технически компетентным образом. Бюрократические работники могут быть склонны вкладывать свое чувство реальности и идентичности в авторитет организации, но они не получают за это систематического поощрения и вознаграждения. В принципе, общение между работниками, при котором они бросают вызов бюрократической власти или иронизируют над ней, допускается, если сами правила открыто не нарушаются. Напротив, в организациях с сильной корпоративной культурой такое "нелояльное" общение в лучшем случае строго закодировано, если не табуировано: "либо вы соглашаетесь с их нормами, либо уходите"».

ки кода, чтобы соответствовать показателю, но это легко исправить, заменив показатель чем-то более значимым, например оттоком кода[1]. Но отсутствие показателей намного хуже, чем наличие нелепых показателей, независимо от того, насколько это может быть противоречивым. Перефразируя Каспарова[2], я бы сказал, что плохой набор показателей лучше, чем их отсутствие.

— Я не согласен, — перебивает Бао, — плохие показатели будут отправлять разработчикам неправильные сигналы. Они начнут думать, что результаты этих показателей влияют на отношение руководства к ним, а ведь это не так[3].

**Количество закрытых тикетов — лучший показатель, который может быть у программиста.**

— Почему это не так?

— Что значит?.. — восклицает Бао. — Ты думаешь, что мы ценим разработчиков по количеству строк кода, которые они пишут?

— Ну, это был плохой пример. Возьми другой показатель. Скажем, количество тикетов, которое программист закрывает в неделю. Чем выше это количество, тем ценнее человек для компании, так ведь?

Бао реагирует весьма эмоционально:

— Нет! Есть много других задач и вообще куча всего, кроме закрытия тикетов!

— Действительно ли компании нужно, чтобы они были выполнены?

— Да, конечно.

---

1 *Hall G. A. et al.* Software Evolution: Code Delta and Code Churn. Journal of Systems and Software, Volume 54, Issue 2, 2000.

2 *Kasparov G.* How Life Imitates Chess: Making the Right Moves, from the Board to the Boardroom. Bloomsbury, 2010: «Лучше плохой план, чем его отсутствие».

3 *Milosevic D. et al.* Standardized Project Management May Increase Development Projects Success. International Journal of Project Management, Volume 23, Number 3, 2005: «Неверно полагать, что стандартизация факторов управления проектом автоматически повысит его успешность. Стандартизация управления проектом не обязательно будет способствовать его успеху, и мы не можем утверждать, что повышение уровня стандартизации факторов управления проектом автоматически сделает его успешным».

— Почему мы не можем превратить их в тикеты?

— Как ты можешь превратить обсуждение новой настройки сервера в тикет?

— В тикете указывают: номер, ответственного за этот тикет и список участников. Когда обсуждение заканчивается, тикет закрывается и каждый получает, например, один балл.

— Хм... — Бао пожимает плечами. — Как-то странно. Я никогда не видел, чтобы компании так работали.

— Чувак, ну какое нам дело до других компаний? — спрашивает Деннис.

**Эффективная управленческая команда знает, как превратить свои цели в измеримые задачи для сотрудников.**

— Такое, что они на плаву и знают, что делают, ведь они делают это годами. Ребята, никто не управляет разработкой программного обеспечения по этим глупым тикетам! — не сдается Бао. — Это ну очень примитивно! Программисты же не скаковые лошади, они не будут бегать с прикрепленными ко лбу тикетами. Кроме того, как ты предлагаешь решать проблему с разными размерами тикетов? Одни задачи более сложные, другие — очень маленькие. Что, если я закрою десять простых задач на этой неделе, а ты закроешь только две, но большие. Кто выиграет?

— Никто не выиграет, это просто число. Мы проанализируем ситуацию. Например, мы можем измерять каждый тикет в «попугаях» (story points), чтобы сбалансировать результаты разных программистов, — я пытаюсь на ходу что-то придумать.

— Народ, я не думаю, что это сработает, — вмешивается Адриан. — Донесите эту идею до Тони и посмотрите, что он скажет. Мы не можем принять такое решение.

— Пойдем поговорим с ним, почему бы и нет? — предлагает Деннис.

— Ну, если хочешь, — голос Адриана звучит не слишком оптимистично. — Если честно, я на вашей стороне, — говорит он мне. — Я думаю, что такая прозрачность нашей работы может быть полезной, но мы должны обсудить это с Тони.

— Обсудим?

— Конечно.

Мы вчетвером направляемся в кабинет Тони. Он сидит за стеклянной стеной, позади него большая белая доска. Я никогда не видел, чтобы он пользовался ею. Может быть, ему нужно что-то для гордости за свою работу? Людей больше интересуют их эмоции, чем объективные результаты. Мы гораздо более иррациональны, чем думаем[1]. Более того, рациональность теперь приравнивается к хладнокровию, цинизму и даже жестокости. Логическое мышление ценится куда меньше, чем эмоции, чувства и даже неконтролируемое поведение. И похоже, это не новая тенденция[2].

Нам не нравится иметь дело с сильными и логичными противниками, они нас пугают. Намного удобнее иметь дело с тем, кто уязвим, мягок, слаб и нуждается в помощи. Такой человек более управляем и контролируем, чем тот, кто знает, что делать, и способен мыслить хладнокровно. И главное — в этом есть доля ревности: мы начинаем ревновать, когда видим кого-то в хорошем настроении, действующего хладнокровно, свободного от эмоций. Мы не хотим доверять ему наши деньги.

Следовательно, чтобы достичь большего и подняться выше по карьерной лестнице, нужно притворяться, что нами движут спон-

**Чтобы быть успешным, нужно выглядеть эмоциональным и слабым, потому что сильные и рациональные люди пугают и не вызывают доверия.**

---

1 ★ *Thaler R. H. et al.* Nudge: Improving Decisions About Health, Wealth and Happiness. Yale University Press, 2008: «Ложное предположение состоит в том, что почти все люди практически все время делают выбор, который наилучшим образом отвечает их интересам (или по крайней мере лучше, чем выбор, сделанный кем-то другим). Мы утверждаем, что это предположение совершенно ложно».

2 *Lerner J. S. et al.* Emotion and Decision Making. Annual Review of Psychology, Volume 66, 2015: «В настоящее время многие исследователи в области психологии предполагают, что эмоции являются доминирующим фактором при принятии наиболее значимых жизненных решений. Принимая конкретные решения, мы пытаемся избежать отрицательных эмоций (например, вины, страха, сожаления) и усилить позитивные чувства (например, гордости, счастья, любви), даже когда не осознаем этого».

танные и трудно контролируемые эмоции. Это вызовет симпатию у людей, что немедленно повлечет за собой доверие и желание помочь. Другими словами, настала эра шизофреников и антидепрессантов.

— Тони, у нас есть идея, — храбро начинает Деннис. Ему нечего бояться, поскольку Тони не является его непосредственным начальником. — У тебя есть минутка?

— Конечно, что там у вас? — похоже, Тони в хорошем настроении.

— Ну, это не идея, — осторожничает Адриан. — Мы просто обсудили кое-что, и нам бы хотелось услышать твое мнение.

— Конечно, садитесь.

Мы рассаживаемся.

— Рассказывайте.

Тони — хороший человек. Компания платит ему именно за то, что он хороший человек. Не за программное обеспечение, которое он пишет нашими руками.

— Как насчет того, чтобы измерять нашу производительность с помощью показателей и вознаграждать тех, у кого самые высокие значения? — нетерпеливо объясняет Деннис.

— Ну... — Тони смотрит на Адриана. — Какие показатели?

— В этом-то и проблема, что мы толком не знаем какие, — добавляет Бао.

— Чувак, прекрати! — Деннис сердится. — Мы только что обсуждали их. Мы знаем показатели! Мы можем подсчитать количество баллов, которые каждый программист получает, закрывая тикеты. Чем больше баллов я заработаю к концу месяца, тем больше денег получу!

Все улыбаются.

— Что не так? Я вроде правильно все объяснил? — спрашивает меня Деннис.

Я киваю. Тони выглядит озадаченным. Под его руководством работает более 50 программистов, он должен быть очень осторожен, верно?

Тони спрашивает меня:

— Твоя идея?

Похоже, у меня проблемы.

— Ну да, я прочитал об этом в книге о менеджменте, — осторожно отвечаю я. — Там пишется, что важно измерять производительность, чтобы мотивировать людей. Это несложно, во всяком случае, мы можем попробовать, почему бы и нет?

— А ты что думаешь? — Тони поворачивается к Адриану.

Как толковый руководитель, он собирает все мнения, прежде чем принять какое-либо решение или даже высказать свою точку зрения.

— Честно говоря, я не думаю, что нам это подходит, — отвечает Адриан.

Деннис бросает на него удивленный взгляд.

— Может быть, в некоторых других компаниях, где рабочий процесс организован более дискретно, это могло бы сработать. У нас так много неопределенностей.

— Но ты сказал, что тебе понравилось! — Деннис реагирует немедленно.

Впрочем, он ведь прав. Адриан поддерживал нашу идею всего лишь за несколько минут до этого. По всей видимости, Адриан, как хороший командный игрок, почувствовал отношение Тони к этой концепции и сразу же подыграл ему.

— Я этого не говорил, — улыбается Адриан, чувствуя себя в безопасности за спиной Тони, — я сказал, что мы должны обсудить это с Тони.

— Подождите, народ, успокойтесь, — Тони поднимает обе руки. — Давайте будем профессионалами. Ты что думаешь? — спрашивает он у Бао.

**При слабом управлении умение быть заодно с боссом — ключевой профессиональный навык.**

— Мне очень не нравится такая идея, и я уже говорил об этом — по той же причине, которую упомянул Адриан: наша работа недостаточно дискретна. У программистов слишком много других задач, кроме закрытия тикетов. Мы не можем просто превратить их в тикетных обезьян, это их обидит и ничем делу не поможет.

— Меня не обидит! — восклицает Деннис. — О чем ты говоришь? Я программист и в любом случае каждый день закрываю эти тикеты! Почему это может меня обидеть?

— Погоди, — холодно произносит Тони. — Так все же, почему тебе это нравится? — Тони спрашивает меня, и я понимаю, что по-хорошему мне следовало бы поступить так же, как и Адриан, — сдаться.

Бао и Тони смотрят на меня, Адриан вертит телефон в руке.

Сдамся? Нет. Попробую вступить в схватку, авось выгорит. Хотя я знаю, что это глупое решение.

— Я считаю, что технические специалисты предпочитают работать так, чтобы правила были понятны. Особенно когда они знают, для чего работают и как измеряются их результаты. Профессиональные программисты любят ясность, прозрачность, дисциплину и все такое. Отсутствие четких показателей производительности, которые есть у большинства компаний, включая нашу, — это большое разочарование для опытных программистов. В принципе, такая ситуация будет на руку неопытным программистам, так как их невозможно толком мониторить. Кроме того, это на руку лентяям.

— У нас таких нет, — перебивает Бао.

— Тс-с-с... — Тони поднимает руку. — Продолжай.

— Любой показатель будет полезен при условии, что он универсален и прозрачен для всех. Конечно, не нужно считать строки кода, это было бы глупо, но мы можем подсчитывать баллы, которые назначаются за каждую задачу, поставленную руководством, или Машей, или кем-либо, кто создает тикеты, и программисты соглашаются с ними, когда начинают работать над тикетом. К концу месяца или недели мы увидим, кто чего достиг.

— А что потом? — перебивает меня Бао.

— Мы награждаем победителей и убиваем проигравших, — шучу я, но никто не смеется.

— А как насчет остальных дел, которыми занимаются сотрудники? Например, совещания? — спрашивает Тони.

**Убеждать в чем-то высшее руководство — опасное занятие, которого следует избегать любой ценой.**

— Их будет меньше, если они действительно не нужны для достижения именно основных целей, которыми являются баллы за тикеты. Либо сотрудники могут попросить организатора совещания начислить им баллы за участие.

— Эти баллы станут новой местной валютой, — шутит Тони, и все смеются. — Давайте вместо этого использовать биткойны!

— Давайте лучше создадим собственную криптовалюту, — предлагает Бао, и я понимаю, что разговор окончен — Тони не поддержит нашу идею.

— А ты можешь? — Тони, похоже, заинтересовала идея криптовалюты.

— Конечно, просто понадобится несколько мощных серверов.

— А давай, я полностью поддерживаю, сегодня же получу на это бюджет! — Тони снова шутит, и мы все снова смеемся.

— Но почему бы нам хотя бы не попробовать, Тони? — Деннис вернулся к обсуждаемому вопросу.

— Ну это, конечно, интересная идея, — начинает Тони, и мы все понимаем, к чему он клонит. — Ты видел, как это делается в других компаниях? — спрашивает он у меня. — Я имею в виду, почему в Google или Facebook такого нет?

На этот вопрос мне нечего ответить.

— Я не знаю. Может, они тупые? — я говорю и понимаю, что это ошибка.

— Да, конечно, Google тупой, — Бао немедленно использует подвернувшуюся возможность.

— Может быть, их не волнует производительность и эффективность. Или их программисты довольствуются уже тем, что работают в Google и ездят на разноцветных велосипедах? — пытаюсь держаться я, но уже нет смысла — я проиграл.

— Так а наши программисты несчастны, что ли? — изображает удивление Тони. — Я не понимаю.

— Да я же говорю, что мы были бы намного счастливее, если бы точно знали, насколько мы хороши! — восклицает Деннис.

— Деннис, успокойся, — Тони смотрит на него.

— Ну прости, я слишком взволнован. Мне лучше вернуться к кодированию, — Деннис встает и уходит.

Несколько секунд мы все сидим тихо. Тони говорит Адриану:

— Поговори с ним, пожалуйста. Я не понимаю, чего он так расстроился. Давай разрулим эту проблему, нам не нужны проблемы в команде. Хорошо?

Адриан кивает. Тони говорит мне:

— Мы подумаем над этой идеей. Но давай не будем раскачивать лодку. Видишь, что только что случилось с Деннисом? Пожалуйста, держи такие идеи при себе, если это возможно. Справишься?

Я киваю. Я проиграл. Мы встаем и уходим.

# Эксперты и обмен знаниями

Беспокоит ли меня отношение Тони? Да, очень. Он запросто может меня уволить. Человек он хороший, но спонтанный. Как и все хорошие люди. Он просто хочет сгладить острые углы в рабочем процессе. Он не может экспериментировать с новыми идеями. Он отвечает за большую группу людей, включая свою жену и детей. Наверняка у него слишком большая зарплата, чтобы он мог позволить себе рисковать. Поэтому, что бы мы ему ни предлагали, ответ будет: «Как это делает Google?» Он может копировать только те вещи, которые уже были приняты рынком и доказали, что они приемлемы[1].

И неважно, насколько они эффективны, важно, чтобы они не были рискованными. Если что-то гениально, но «может не сработать» — это не для Тони. Это трусость? Похоже. Осторожность? Да, тоже верно. В этом вся сущность топ-менеджмента: лучше ничего не делать, чем рисковать. А ведь эта компания даже не очень большая. Я могу только представить, насколько сложно было бы продвинуть что-то новое в крупной корпорации. Вот почему я не хочу тратить на них свое время.

— Не переживай, мы продолжим этот разговор, — встречает меня Адриан возле кофемашины. — Я работаю с Тони уже более четырех лет. Он хороший человек, но, знаешь, такие идеи требуют времени.

**Парадокс в том, что мы одновременно ненавидим и сам контроль, и вызванный его отсутствием хаос.**

— Да все в порядке, всегда важно обсудить, что нас беспокоит, независимо от результата, — я подыгрываю, притворяясь командным игроком, который заботится не только о произво-

---

[1] *Jones R.* What CEOs Are Afraid Of. Harvard Business Review, February 2015: «Наибольший страх [генерального директора] — оказаться некомпетентным, он также известен как синдром самозванца. Этот страх снижает уверенность и подрывает отношения с другими руководителями».

дительности или качестве, но и о комфортной рабочей обстановке. Люди любят это.

— Ты действительно думаешь, что нашим разработчикам понравится работать с тикетами?

Мы стоим на кухне, держа в руках бумажные стаканчики с темной жидкостью, которую американцы называют кофе.

— Я имею в виду, что люди любят поболтать, им нравится офисная атмосфера, им приятно быть частью команды, они здесь не просто для того, чтобы закрывать тикеты и идти домой, — улыбается он, и я понимаю, что он говорит прежде всего о себе.

— Конечно, — мой тон очень дружелюбен, — я тоже. А кто нет? Мы ведь не роботы. Но все же основная причина, почему мы здесь, наша работа, верно?

— Угу, — он пьет кофе.

Я пытаюсь понять, действительно ли ему интересно узнать, что я думаю, или он собирается гнуть свою линию, и продолжаю:

— Я считаю, в первую очередь должна быть какая-то структура, а потом — развлечения. Посмотри на Денниса. Он один из лучших программистов и очень лояльный, независимо от того, что он там может ляпнуть о компании или о том, что он несчастен. Он допоздна задерживается в офисе и много работает. Но он не удовлетворен положением дел и очень расстроен. Почему так? — риторически спрашиваю я. — Потому что он в вечном стрессе. Стресс возникает из-за его неспособности видеть границы работы и ответственности. Он просто не отдыхает. Конечно, он идет домой, спит, играет в свои любимые стрелялки, у него есть выходные и все такое. Но он всегда чувствует ответственность за что-то большое, что даже не может контролировать. Он не перестает быть ответственным, когда его рабочий день заканчивается. Даже когда его задача или тикет закрыты. Его работа и результаты не определены и не дискретны — вот что беспокоит его больше всего.

— Хм... — Адриан продолжает слушать и пьет.

— Ты знаешь, что есть стресс и дистресс. Врачи говорят, что стресс полезен для нас, потому что он мобилизует наши внутренние

ресурсы и дает нам энергию. Однако важно убедиться, что стресс быстро уходит и мы можем расслабиться. Если этого не происходит и стресс остается с нами дольше, он превращается в дистресс, в итоге мы получаем рак[1].

— Да, слышал об этом, — в его голосе сквозит скепсис.

— Я тоже не уверен в причинах рака, но дистресс вреден, могу сказать это абсолютно точно. И у Денниса явно дистресс. Он напряжен, когда приходят новые задачи, он их выполняет, исправляет проблемы, но ответственность никуда не уходит. Он остается ответственным за проект даже после того, как его работа закончена. Просто потому, что его ответственность не только в тикетах, а везде. Он не знает, что нужно сделать, чтобы остановиться и передохнуть. Если бы он был не таким профессионалом, ну, как некоторые другие в нашей команде, — мы оба улыбаемся, — он бы просто проигнорировал большую часть этой ответственности, ему было бы все равно. Он бы просто работал с девяти до пяти и выключал телефон, выйдя из офиса. Но он не такой. Он лоялен, у него душа болит за проект. Это и вызывает беспокойство и разочарование. Нам нужно помочь ему оставаться эффективным и позитивно настроенным, устанавливая границы для его задач и его ответственности.

— Ага... — Адриан зевает с закрытым ртом.

— Мы должны официально позволить ему отдохнуть, когда задание выполнено. Мы должны сказать, что он не несет ответственности ни за что, кроме проблем, описанных в его тикетах. Когда он закрывает их, он свободен. И больше не виноват!

— Виноват? — улыбается он.

---

[1] *Moreno-Smith M.* Impact of Stress on Cancer Metastasis. Future Oncology, Volume 6, Number 12, 2010: «Влияние психосоциальных факторов на развитие и прогрессирование рака — давняя гипотеза. Фактически эпидемиологические и клинические исследования за последние 30 лет представили убедительные доказательства связи между хроническим стрессом, депрессией, социальной изоляцией и прогрессированием рака. При этом существует лишь ограниченное количество доказательств роли этих поведенческих факторов в возникновении онкологического заболевания».

— Похоже, ответственность и вина близкие эмоции. Мы не можем допустить, чтобы Деннис чувствовал вину в течение длительного времени. Да не только он, но и любой другой. Вместо того чтобы решить ряд проблем и стать невиновным (это именно то, что нам нужно), он просто выгорает и теряет надежду. Он чувствует себя как заключенный, который выйдет на свободу только после того, как отсидит весь срок, а не когда закончит работу.

**Отсутствие четко определенной индивидуальной ответственности нервирует тех, кто искренне беспокоится о проекте.**

— Да уж... — он снова зевает.

Пришло время завершить этот разговор.

— Я уверен, что мы что-нибудь придумаем, — улыбаюсь я. — Давай вернемся к работе. Я поговорю с Деннисом, не волнуйся.

Я говорю то, что он, как мне кажется, хотел услышать, когда начал этот разговор.

— Спасибо, поехали.

Мы возвращаемся на свои рабочие места.

Маша снова здесь — разговаривает с Деннисом.

— Я же объясняла тебе на прошлой неделе, разве ты не помнишь? — говорит она немного нервно.

— Да, действительно, — соглашается Деннис. — Подожди, я не помню, можем пройтись по этому еще раз?

— Деннис, ты меня убиваешь! Смотри, когда заказ закрыт и клиент хочет получить частичное возмещение, мы не должны возвращать комиссию, которую уже заплатили банку. Однако если комиссия составляет менее доллара, мы возвращаем ее. Помнишь?

— О да, эта комиссия доконает меня!

Маша смотрит на меня:

— Твой друг постоянно забывает о том, что я прошу его сделать, и совершенно не хочет вникнуть в то, как работает компания. Он просто программирует, но не помнит о наших бизнес-правилах.

— Да ладно тебе, — Деннис встает, вытягивая спину. — Ты несправедлива.

— Что случилось? — спрашиваю я.

— Вот скажи, — начинает Маша, — что ты думаешь? Должен ли хороший программист вникать в бизнес-правила или обязан просто реализовывать то, что я ему говорю?

Она явно ждет моей поддержки.

— Не совсем, — отвечаю я.

— Что? — удивляется она.

— Видишь? — восклицает Деннис. — Я выиграл!

— Чувак, подожди, дай мне объяснить. Я считаю, что никто в проекте не должен обладать недоступными для остальных сведениями из документации. Иными словами, если ты знаешь что-то, что Деннис не может прочитать в файлах проекта, это твоя вина, а не его. На самом деле это даже не твоя вина, а вина проекта.

**Совместные проекты подвержены большему хаосу, потому что соблазн игнорировать ведение документации становится сильнее.**

— Снова-здорово, — вздыхает она. — Мы не можем все записать, это просто невозможно. Требования и бизнес-правила меняются так часто, что намного быстрее и проще объяснить их лично, быстро переговорив. Если бы мы жили в разных странах и работали удаленно, я бы, наверное, поняла, что документация лучше, но мы же в одном офисе!

— Быстрее, говоришь? И посмотри, что происходит сейчас. Тебе нужно объяснять все еще раз этому ленивому парню, — я смотрю на Денниса. — И это занимает много времени. Если бы ты просто написала правила один раз — тебе даже не пришлось бы подходить к нам, ты бы отправила ему по почте ссылку на документ, вот и все.

— Слушай, — отвечает она, — забудь о документации, мы пробовали это в прошлом году, и не один раз. Но не сработало. Никто не пишет инструкцию и не читает ее, легче все объяснить на словах. Думаю, потому компания и арендует этот офис — в первую очередь для того, чтобы мы общались лично. Если бы мы работали только с документацией, нам бы не понадобился офис. Мы бы просто работали из дома.

— А что в этом плохого? — спрашиваю я[1].

— Не начинай. У всех компаний есть офисы, посмотри на большие фирмы[2, 3]. Так просто удобнее[4]. Люди не роботы. Они не любят возиться с документами, они предпочитают работать с другими людьми.

Я пожимаю плечами и улыбаюсь.

— Может быть, они не знают, как работать с документами, поэтому предпочитают работать с людьми?

— Я знаю, как работать с документами, у меня степень магистра в области разработки программного обеспечения, — обиделась она.

— Да он не имел в виду тебя лично, Маша, перестань, — Деннис делает попытку успокоить ее.

---

1 *Groskopf C.* For Programmers, the Ultimate Office Perk is Avoiding the Office Entirely. Quartz, April 12, 2017: «Количество программистов, которые работают полный рабочий день из дома, в течение последнего десятилетия увеличивается со средней скоростью 11,5 % в год».

2 *Kasriel S.* IBM's Remote Work Reversal is a Losing Battle Against The New Normal. Fast Company, May 2017: «До недавнего времени IBM была одним из первых и крупнейших сторонников удаленной работы. Но сейчас это уже не так. В марте компания начала предлагать тысячам сотрудников либо выходить в офисы, либо искать другую работу».

3 *Truong A.* Reddit Gives Remote Employees Until End of Year to Relocate to San Francisco, Fast Company, October 2014: «Выяснилось, что наши команды (в каждом офисе) и удаленные работники в целом проделали хорошую работу, но разделение не позволило нам эффективно координировать наши действия на уровне всей компании. Несмотря на все усилия, потребовавшие быстрых действий, глубокого понимания и эффективной координации между людьми в нескольких офисах, дела не идут так, как необходимо нам и нашим пользователям».

4 *Swisher K.* Physically Together: Here's the Internal Yahoo No-Work-From-Home Memo for Remote Workers and Maybe More. AllThingsD, 22 February 2013: «Если мы хотим стать лучшим местом для работы, нам важно личное общение и сотрудничество, поэтому мы должны работать бок о бок. Вот почему так важно, чтобы мы все присутствовали в наших офисах. Некоторые из лучших решений и идей рождаются из разговоров в курилке или буфете, встреч с новыми людьми и спонтанных совещаний в командах. Скорость и качество часто приносятся в жертву, когда мы работаем из дома. Мы должны быть одним Yahoo!, и это начинается с того, что мы находимся вместе».

— Слушай, не обижайся, — продолжаю я, — разве не очевидно, что чем лучше организована наша документация, тем быстрее и проще программисты смогут реализовать твои идеи? Тебе будет намного удобнее работать, если вместо многократных объяснений ты запишешь все один раз. Я не имею в виду длинные многостраничные документы — их никто не любит ни писать, ни читать. У проектной документации есть разные формы. Честно говоря, я считаю, что лучшая форма — это тикеты. Они похожи на настоящие чаты, но мы всегда можем вернуться к истории, посмотреть, кто что сказал, и избежать повторения вопросов и ответов.

— Но у нас уже есть тикеты, — возражает Маша.

— Да, они у нас есть, но там очень мало информации. Большая часть сведений хранится в наших головах. Представьте, что произойдет, если компания, не дай Бог, потеряет тебя или его, — указываю на Денниса. — Это будет катастрофа, так ведь? Потому что вы слишком много знаете, и эти знания нигде не задокументированы[1].

— Это называется фактором автобуса[2], — изрекает Деннис.

— Да, как-то так. Но дело не только в этом. Проблема в том, как распространять знания в команде. Аккумуляторы знаний замедляют общую работу.

**Умная команда делает все, чтобы избавиться от экспертов. Они создают заторы в потоке знаний.**

---

[1] *Rigby P. C. et al.* Quantifying and Mitigating Turnover-Induced Knowledge Loss: Case Studies of Chrome and a Project at Avaya. Proceedings of the 38th International Conference on Software Engineering (ICSE'16), 2016: «Люди, которые создают программное обеспечение, часто не делятся своими знаниями о внутренней работе системы, и это затрудняет поддержку продукта другими. Поэтому важно оценить риск того, что разработчики покинут проект».

[2] *Monperrus M.* Principles of Antifragile Software. The First International Conference on the Art, Science and Engineering of Programming, Brussels, Belgium, 2017: «В разработке программного обеспечения "фактор автобуса" показывает, насколько люди важны для проекта. Если ключевого разработчика сбил автобус (или ему на голову упал кирпич), может ли это обрушить весь проект? С точки зрения надежности такое последствие означает, что любой отказ в работе системы может из незначительной и легкоисправимой проблемы превратиться в катастрофу».

На первый взгляд кажется, что их наличие — это здорово. Однако это не так. Чтобы стать круче и собрать больше знаний, эксперты неизбежно должны блокировать доступ к информации для всех остальных. Не глобально, конечно, но в некоторой степени. В противном случае, если информация находится в свободном доступе и проста для понимания, экспертов просто не будет[1]. Мы все будем экспертами в той или иной степени. Но лучше, когда знания в равной степени распределены среди всех членов команды[2], что, скорее всего, означает, что никто ничего не знает и всё есть в документации, — улыбаюсь я. — Таким образом, отвечая на твой первоначальный вопрос, Маша, я считаю, что программисты должны намеренно самоустраниться от бизнес-знаний, чтобы не превращаться в экспертов.

---

[1] *Sedano T.* Sustainable Software Development through Overlapping Pair Rotation. Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, Ciudad Real, Spain, 2016: «Традиционная мудрость говорит о том, что отток членов команды негативно скажется на успехе проекта и что подробная документация необходима для смягчения этой ситуации. К сожалению, документация быстро устаревает, что снижает эффективность такого подхода. В ходе основательного теоретического исследования мы наблюдали, что проекты были успешными, несмотря на срывы сроков и незначительное количество документации. Это поставило следующий вопрос: "Как команды эффективно разрабатывают программное обеспечение, преодолевая трудности в работе?". ...Это достигается путем формирования положительного отношения к распаду команды, поощрения обмена знаниями и преемственности, а также заботы о высоком качестве кода».

[2] *Coplien J. O.* Organizational Patterns of Agile Software Development. Prentice Hall, 2004: «Определим количество незаменимых сотрудников (англ. truck number) как количество людей в организации, которые являются уникальными экспертами в критических областях. Вы не хотите, чтобы незаменимых людей было больше, потому что это означает, что велика вероятность того, что потеря любого члена команды будет означать потерю критического опыта. Слишком высок риск. И все же нельзя сделать так, чтобы незаменимых сотрудников было очень мало (и невозможно вообще без них обойтись). Держите количество незаменимых человек низким; сохраните небольшое количество ключевых экспертов с уникальными знаниями. Создайте культуру обмена знаниями (особенно это касается знаний, которые можно кодифицировать или передать иным образом)».

Маша взмахнула руками:

— Это странно!

— Почему? — Деннис приходит на помощь. — Мне кажется, звучит вполне логично. Смотри, если я слишком много узнаю о компании, то и мне, и тебе работать будет легче, но фирма от этого только пострадает. Потому что никто не сможет понять, что мы делаем. Верно, чувак?

— Да. И еще один момент об экспертах. Программисты оказывают себе медвежью услугу, когда становятся экспертами в предметной области. Они привязываются к одной сфере и к одному программному решению и попросту выталкивают себя с рынка. Кроме того, они перестают расти как профессионалы — перестают изучать фреймворки, языки и библиотеки, потому что те им больше не нужны. И вот однажды, когда программисты попытаются найти новую работу, они вдруг с удивлением увидят, что рынок готов предложить им зарплату втрое меньше, чем они получали раньше. Гарантированная работа сослужила им плохую службу.

— Наши разработчики не хотят менять работу, — говорит Маша.

— Это то, во что руководство любит верить. Реальность совершенно иная. Толковые и активные программисты меняют работу примерно раз в год[1]. Если они остаются дольше, это, вероятно, симптом проблемы[2], известной под названием «Я эксперт!». Компании начинают платить им больше из-за боязни

**Если вы становитесь экспертом в одном конкретном проекте, это большой риск для карьеры, которого умный программист должен избегать.**

---

[1] StackOverflow Developer Survey. https://goo.gl/CtCSyQ, 2017: «35 % респондентов подтвердили, что сменили работу в течение последнего года».

[2] *Giang V.* You Should Plan on Switching Jobs Every Three Years for the Rest of Your Life. Fast Company, January 2016: «Раньше смена работы каждые пару лет не украшала резюме. Это как бы говорило рекрутерам, что вы не способны удержаться на одном месте, ладить с коллегами или что вы неблагонадежны и не можете выполнять свои обязательства. Такое видение очень скоро изменится, особенно когда на рабочем месте — миллениалы с ожиданиями постоянного обучения, развития и продвижения по службе. ...Говорят, что работникам, которые остаются в компании более двух лет, платят на 50 % меньше».

потерять эти хранилища знаний[1]. Это ловушка для бизнеса и для программиста. Они оба в итоге проигрывают.

— То есть ты считаешь, что все разработчики должны быть тупыми обезьянами, которых вообще не должен волновать бизнес, а только свой код? — спрашивает Маша.

— Ты просто обидела нас обоих, — смеется Деннис. — Кодирование само по себе так же интересно, как и ваш бизнес. Во всяком случае мне, честно говоря, это намного интереснее.

— Правда? — Маша выглядит удивленной.

— Да! Я был бы абсолютно счастлив, если бы ты держала меня подальше от своих бизнес-концепций и идей и просто говорила, что именно должно быть реализовано.

— То есть вам действительно не хочется знать, зачем нам нужно то, что мы просим вас реализовать? — она и правда удивлена.

— Ни капельки, — улыбаюсь я. — Хочешь узнать, как мы реализуем эти функции, какие фреймворки мы используем, какие методы оптимизации баз данных, какие языки программирования и все такое?

— Да ну тебя, оставь меня в покое, — улыбается она.

— Вот видишь! — Деннис встает. — Мне тоже не нужны ваши подробности. Мне они так же неинтересны. Я просто хочу видеть тебя счастливой, когда я реализую то, о чем ты просишь, вот и все. Хочу, чтобы Маша была счастлива!

— О, это так мило, — смеется она.

---

[1] *Cromity J.* Silo Persistence: It's not the Technology, it's the Culture! New Review of Information Networking, Volume 16, Number 2, 2011: «Внутри любой организации, учреждения или компании может быть хранилище знаний — человек, отдел, приложение, база данных или сеть, к которой только один или несколько человек могут получить доступ».

# Статический анализ

Деннис вне себя от злости. Не знаю почему, но это не имеет значения. Чувак все время какой-то гиперэмоциональный. Я не могу представить, как он может быть отцом двоих детей, если сам ведет себя как ребенок.

— Хочешь, покажу тебе этот тупой код? — спрашивает он.

Я обхожу стол и сажусь рядом с ним. Код действительно ужасен, впрочем, как и весь остальной код, который я видел здесь. Я не могу понять, что его беспокоит именно сейчас. В любом случае я должен сделать вид, что понял проблему:

— Да, я понимаю.

— Нет, не понимаешь! Я правильно отформатировал этот кусок месяц назад. Кто-то испортил мое форматирование, и я снова исправил его на прошлой неделе. А сейчас код опять выглядит кошмарно. Что же это такое?

— И кто это сделал?

— Гит говорит, что Том.

— Давай найдем его и убьем!

— Чувак, я не шучу, это раздражает! Я продолжаю делать одну и ту же работу снова и снова. Думаю, нам нужно сесть и договориться о правилах форматирования. В противном случае мы будем тратить слишком много времени. Давай соберем совещание.

**Отсутствие единообразия в стиле кодирования вызывает еще большее разочарование, чем функциональные дефекты.**

— Ты уже говорил с Томом?

— Нет, а какой в этом смысл? Кто-то другой еще не раз сделает то же самое. Мы должны договориться все вместе — вся команда. Или ты так не считаешь?

— Ну, вообще-то нет.

— Почему нет? — он закрывает ноутбук, скрещивает руки на груди и смотрит на меня.

— Нам нужен статический анализатор и предполетный сборщик.

Предположение, что программисты могут и должны писать код правильно только потому, что они хорошие программисты, весьма популярно[1]. Я слышал шутку, что мы не появляемся в офисе в нижнем белье не потому, что есть письменное правило по этому поводу, а потому, что сами понимаем, как не должны себя вести[2]. То же самое верно для стиля кодирования и нашего отношения к качеству. Мы не пишем плохой код только потому, что нам этого не хочется. Может быть, эта метафора подходит?

Он нетерпеливо перебивает, даже не слушая, что я говорю:

— Нет, чувак! Нам нужны хорошие программисты, которые понимают, как писать код правильно.

— Хорошие программисты?

— Да, хорошие. Ты так не думаешь? Ты же всегда за качество, разве ты не понимаешь, как важен внешний вид кода?

— Понимаю, конечно. Но дело не в том, хорошие программисты или плохие. Они все хорошие. Если мы поговорим с Томом сейчас, он скажет, что плохой программист — это ты и что ему пришлось переформатировать код после тебя и переписать некоторые его части. Он будет винить тебя. А ты будешь винить его. Все это очень субъективно. Я вообще категорически против этих определений «хорошо vs плохо», когда мы говорим о людях. Мы все можем быть и хорошими, и плохими — все зависит от обстоятельств. Задача — убедиться, что программисты делают то, что нужно проекту. Не потому, что они такие хорошие, а потому, что у них нет другого

---

[1] *Glass R. L.* Frequently Forgotten Fundamental Facts About Software Engineering. IEEE Software, Volume 18, Number 3, 2001: «Согласно исследованию индивидуальных различий, хорошие программисты в 30 раз лучше посредственных».

[2] GeePaw Hill рассказал об этом в эпизоде подкаста Shift-M № 27. https://goo.gl/ip1gt6.

выбора. Макиавелли сказал, что люди делают добро только в том случае, если их заставить[1]. Глупо надеяться, что все будут делать то, что мы считаем хорошим: писать высококачественный, правильно отформатированный Java-код. Они напишут то, что сами считают хорошим, если мы не заставим их делать иначе.

— Заставить? Тот итальянец, которого ты цитируешь, жил сотни лет тому назад[2], когда мир был совершенно другим. Мы больше никого не заставляем, чувак. Таковы сегодняшние реалии.

— Если тебе так сказали, это еще не означает, что все действительно так.

— Мне так сказали?

— Я имею в виду, что ты веришь, будто никто больше никого не заставляет, но на самом деле мы все те же люди, которые жили во времена Макиавелли или даже Сунь-цзы. Мы по-прежнему жадные, эгоистичные, ленивые. Когда мы нужны бизнесу, когда нам приходится работать для него, он должен изобрести инструменты, чтобы заставить нас. Собственно, он должен изобрести и кнут, и пряник. И я не имею в виду, что в этом есть что-то плохое. Это в основном правила и дисциплина, позволяющие нам оставаться сфокусированными и синхронизированными. Я просто говорю, что каждый делает все в лучшем виде, насколько может, в настоящий момент, по предположению Канта[3], и в командной работе это не поможет, — объясняю я и думаю о том, насколько эта концепция далека от философии современных программистов-самоучек, саморегулируемых команд и беззубого управления.

**Нет хороших или плохих программистов, есть слабое или сильное управление.**

---

[1] *Machiavelli N.* Discorsi sopra la prima deca di Tito Livio. 1517, книга 1, глава 3: «Люди никогда не делают добро, если в этом нет необходимости».

[2] Никколо Макиавелли жил в XV–XVI веке и умер 21 июня 1527 года, в возрасте 58 лет.

[3] *Kant I.* Grundlegung zur Metaphysik der Sitten. 1785: «Поступай согласно максиме, которая в то же время может иметь силу всеобщего закона» (категорический императив).

И тут он спрашивает:

— Ты слышал о теории XY?[1]

— Да, но я не очень-то верю в часть Y[2]. Конечно, есть люди, которые все время стараются делать что-то хорошее, но это только потому, что им действительно не нужен никакой внешний мотиватор. Они родились и выросли в очень нравственной и дисциплинированной среде и очень четко знают, что даже попытка сделать что-то плохое другим людям приведет к суровому наказанию[3].

**Наше врожденное желание делать добро (как мы его понимаем) — это результат укрощения и обучения многих поколений ранее.**

Они выучили это, когда им было три года[4]. Теперь им 20, и мы называем их ответственными. На самом деле они просто отлично

---

[1] *McGregor D.* The Human Side of Enterprise. Adventure in Thought and Action, Proceedings of the Fifth Anniversary Convocation of the School of Industrial Management, MIT, 1957: «Философия управления через руководство и контроль (независимо от того, жесткий он или мягкий) недостаточно мотивирует, потому что потребности человека, на которые опирается этот подход, сегодня считаются слабыми мотиваторами поведения».

[2] См. сноску 5 на с. 126: «Философия теории Y была основана на предположении о добровольном компромиссе между потребностями индивидуумов и целями корпораций, которое было отвергнуто и искажено теорией X. Это предположение было наивным, поскольку не учитывало глубоко укоренившиеся конфликты интересов в организациях, которые концентрируются вокруг социального разделения собственности и контроля».

[3] *Neusner J. et al.* Golden Rule: The Ethics of Reciprocity in World Religions. Continuum, 2009: «Золотое правило "Поступай с другими так, как хотел бы, чтобы другие поступили с тобой" отражено в большинстве религий и стало частью человеческой мудрости».

[4] *Kohn A.* Unconditional Parenting: Moving from Rewards and Punishments to Love and Reason. Atria Books, 2006: «Бихевиористы предполагают, что все, что мы делаем, можно объяснить с точки зрения того, приносит ли это какое-то вознаграждение (предложенное либо происходящее естественным образом). Внешние силы, например то, что кто-то ранее был вознагражден (или наказан) за действия, объясняют наше поведение. А то, как мы действуем, — это показатель того, кто мы есть».

выдрессированы — точно так же, как немецкие овчарки, которые не трогают кусок мяса на земле. Мы считаем их умными, а в реальности они когда-то пару раз получили разряд тока, когда пытались съесть что-то с пола, и инстинкт довольно хорошо развился. Они остаются теми же собаками, готовыми кушать все, что им хочется, но благодаря дрессировке выглядят и ведут себя дисциплинированно и «хорошо» — так, как мы это понимаем.

— Тогда давайте используем электрошок и для программистов, — предлагает он.

— Ты шутишь, конечно, но в этой шутке есть доля правды. Нам нужен какой-то электрошок, который заставит нас писать код нужного качества. Без этого мы будем как те собаки — энергичные, но недисциплинированные.

Создание работающего программного обеспечения — лишь небольшая часть всей работы[1]. Намного сложнее создать программный продукт в том виде, в котором его смогут использовать клиенты. Тем не менее мы продолжаем обучать программистов в основном алгоритмам, языкам, булевой логике и фреймворкам, практически не обращая внимания на все остальное, что превращает составляющие программного обеспечения в рабочий продукт, включая поддержку, обслуживаемость, документацию, требования, планирование, оценку времени и т. д. Кодирование — лишь малая часть нашей работы. Похоже, Деннис — типичный пример кодера, который заботится только о своих Java-инструкциях и думает, что все остальное не столь важно и может быть сделано кем-то не таким умным.

— Статический анализ, — начинаю объяснять я, — это процесс проверки качества кода без его выполнения. Должен быть инструмент, который просматривает твои Java-классы и находит, где они выглядят некорректно. Возможно, они прекрасно работают, но статический анализатор это не волнует. А вот на то, как они выглядят, он будет обращать внимание. Проблемы с форматированием будут немедленно обнаружены, как и многое другое, в зависимости от качества анализатора.

---

[1] См. сноску на с. 159.

— А ты их использовал?

— Конечно, в своих проектах с открытым исходным кодом.

— Я так и думал — ты используешь их в проектах с открытым исходным кодом, где программисты не очень профессиональны. Ты не считаешь, что если в команде работают умные старшие разработчики, то эти инструменты не требуются?

— На самом деле даже наоборот, — парирую я, — у тебя неоправданно невысокое мнение о мире разработки с открытым исходным кодом. Статические анализаторы не только проверяют форматирование и расположение скобок, но и дают множество рекомендаций по улучшению качества. Возьми FindBugs[1], достаточно мощный инструмент, проверяющий различные аспекты качества, такие как безопасность, например. Честно признаться, я многое узнал о Java, просто прислушиваясь к его рекомендациям. Статические анализаторы подобны коллективному разуму программистов, которые их создали и внесли в них свой вклад[2]. Представь, что у вас есть очень опытный технический редактор кода, и он целый день сидит рядом с тобой, указывая на места в твоем коде, которые можно улучшить. Тебе это понравится?

**Сильный статический анализатор — лучший учитель программирования.**

---

[1] *Ayewah N. et al.* Using Static Analysis to Find Bugs. IEEE Software, Volume 25, Number 5, 2008: «FindBugs теперь распознает более 300 ошибок программирования и подозрительных идиом кодирования, которые можно обнаружить с помощью простых методов анализа. В FindBugs также предусмотрены некоторые более сложные методы анализа, разработанные для того, чтобы помочь эффективно идентифицировать конкретные проблемы, такие как разыменование нулевых указателей, которые нуждаются в проверке и случаются достаточно часто для того, чтобы технологии обнаружения таких проблем развивались».

[2] *Chess B.* Secure Programming with Static Analysis, Pearson Education, 2007: «Статический анализ позволяет находить ошибки на ранних стадиях разработки, даже до первого запуска программы. Раннее обнаружение ошибки снижает стоимость ее исправления, а быстрая обратная связь может помочь в работе программистов: у них есть возможность исправлять ошибки, о которых они раньше и не подозревали. Сценарии атаки и сведения о конструкциях кода, используемые инструментом статического анализа, помогают передаче информации».

Он берет жвачку.

— Думаю, да, — отвечает Деннис через несколько секунд. — Но разве их нельзя автоматизировать? Вместо того чтобы указывать на неправильный код, инструмент мог бы просто автоматически его исправлять. Вот как в случае с функцией автоматического форматирования в IDEA[1] — я просто нажимаю кнопку, и весь класс форматируется правильно. Разве мы не можем автоматизировать все остальное таким же образом?

— Думаю, вполне можем, особенно в отношении простейших проблем форматирования, но я все еще считаю, что это не совсем правильный подход, даже для пробелов и скобок. Во всем этом есть очень важный образовательный фактор. Если ты отформатируешь код автоматически, ты никогда ничего не узнаешь и будешь снова и снова повторять одни и те же ошибки. Кодирование заставляет не только компьютер делать то, что ты хочешь, но и других людей понимать твое намерение, твою идею, твой образ мышления. Форматирование кода, то, как ты называешь свои переменные, методы и классы, — это язык, на котором ты общаешься со своими коллегами-программистами. Я считаю, что каждому программисту важно чувствовать язык и получать удовольствие от его использования[2], а не только автоматически форматировать и идти дальше.

— Знаешь, что я думаю? — риторически спрашивает он. — Я думаю, что большинство из них начнут жаловаться на этот ана-

---

[1] Деннис имеет в виду IntelliJ IDEA, IDE для Java от JetBrains.

[2] *Sadowski C.* Lessons from Building Static Analysis Tools at Google. Communications of the ACM, Volume 61, Issue 4, 2018: «Счастье разработчика — это основной показатель. И литература, и наш опыт говорят о том, что попытки интегрировать статический анализ в организацию по разработке программного обеспечения зачастую проваливаются. В Google, например, обычно не требуют, чтобы инженеры использовали инструменты статического анализа. Инженеры, работающие над статическим анализом, должны продемонстрировать его пользу с помощью достоверных данных. Чтобы проект статического анализа был успешным, разработчики должны чувствовать, что им это выгодно и интересно».

лиз, потому что каждый предпочитает использовать свой собственный стиль. Посмотри на Тома. Как ты сказал, он переформатировал мой код не потому, что хотел насолить, а потому, что посчитал, что так код выглядит лучше. Теперь ты дашь ему статический анализатор, и он удивится и расстроится. Ему придется позабыть о своем собственном стиле и начать использовать твой стиль. Видишь проблему, чувак?

— Наш стиль, — я на секунду делаю паузу, — это не мой или твой стиль, это стиль, принятый в отрасли, стиль, который мы берем из библиотек с открытым исходным кодом. В этом его прелесть — он никому не принадлежит. Если кто-то будет жаловаться на него, я всегда могу сказать, что это не я его изобрел. Это то, что рекомендует индустрия. А вот если мы, скажем, соберемся вместе и создадим свое руководство по корпоративному стилю, как это делают во многих других компаниях, — у каждого программиста появится веская причина жаловаться, потому что, независимо от того, насколько толковыми будут рекомендации по этому стилю, у него возникнут проблемы. Если мы будем против него (я говорю о новом программисте), то можем проиграть. Если против него будет рынок — у него не будет никаких шансов.

> **Статические анализаторы насыщают проект мудростью, накопленной и используемой целой отраслью.**

Деннис задумчиво смотрит в окно.

— В принципе, логично... Но я все же уверен, что все начнут жаловаться на него, чувак. Чтобы очистить код и написать его безупречно, как того требуют эти инструменты, нужно потратить уйму времени. Мы не роботы, мы совершаем ошибки, и не такая уж это и проблема. Код же работает, в конце концов. Ты уверен, что нам действительно нужен такой строгий контроль?

— Ну с чего ты взял, что он строгий? Эти инструменты гибко настраиваются: они могут быть как очень строгими, так и вполне либеральными. Конечно, чем строже, тем лучше. И ты абсолютно прав: разработка будет замедляться для каждой конкретной задачи

и каждого программиста, потому что теперь уже будет недостаточно просто написать рабочий код; понадобится привести его в порядок, прежде чем он будет принят проектом. Сколько времени это займет, зависит от навыков каждого человека, но в целом дело будет двигаться медленнее. Однако весь проект быстрее продвинется вперед, потому что в ветке master станет меньше беспорядка. Посмотри на свою ситуацию: ты недоволен изменениями, которые внес Том, и время, которое ты сейчас тратишь на их исправление, — это потерянное время. И ты в этом отнюдь не одинок. Все члены команды выполняют какие-то действия по два раза, а то и больше. Или вообще не выполняют, но все равно работают с грязной кодовой базой. Строгие правила сделают нашу жизнь проще. И знаешь что? Чем строже правила, тем лучше. Я считаю, что нужно регулярно прилагать усилия, чтобы ужесточить правила. И чем старше проект, тем строже должны быть его правила.

— Интересно... Я уверен, что нам сейчас окажут сопротивление, и большинство из них, — указывает он через плечо, — будут против этих правил.

**Проект должен постоянно инвестировать в свои статические анализаторы, чтобы сделать свои правила более строгими.**

— Конечно, будут.

— Хорошо, тогда давай попробуем! — смеется он.

— Почему бы и нет, будут проблемы и похуже, чем сопротивление.

— Какие?

— Дело в том, что мы не можем прямо сейчас задействовать статические анализаторы в полной мере. Потому что они выявят так много проблем, что мы никогда не сможем их решить.

— Ну, мы же исправим их потихоньку, одну за другой.

— В том-то все и дело — мы не можем исправлять их одну за другой, мы должны либо исправить все сразу, либо забыть об этом. Потому что статический анализатор не должен использоваться в качестве информативного инструмента. Это привратник, который отклоняет все, что недостаточно хорошо сделано, защищая нашу ветвь master.

— Отклоняет?

Слово «отклонить», видимо, наиболее раздражающее понятие философии качества, и большинство программистов его не любят. Они предпочли бы, чтобы контроль качества был комфортен для них. Но это невозможно.

— Вот как это должно работать[1], — отвечаю я. — Мы устанавливаем инструмент и запускаем его. Он сообщает обо всех проблемах, которые находит. Мы проходим через весь код и решаем все проблемы. Инструмент говорит, что код чистый. Затем мы всем запрещаем совершать какие-либо действия непосредственно в ветке master. Вместо этого всем, включая тебя и меня, придется работать в своих ветках. Допустим, ты исправляешь ошибку и делаешь это в своей ветке. Когда ветка готова, ты отдаешь ее предполетному сборщику. Это робот, скрипт. Он берет твою ветку, сливает ее в master и спрашивает у статического анализатора, все ли чисто. Если твой код нарушает какие-либо правила форматирования, робот говорит, что твоя ветка не подходит, и она не попадает в master. Если нарушений нет, твоя ветка будет добавлена, а код появится в master. Это словно электрошокер для тебя. Ты всегда будешь знать, что, если нарушишь форматирование, твой код не примут. Конечно, правила будут одинаковыми для всех. Роботу все равно, откуда исходит код — от тебя, Тома, меня или Тони. Перед ним все равны.

— Хм, вообще здравая идея. Давай попробуем.

— Я же говорю тебе, мы не можем просто взять и сделать. Это легко выполнимо тогда, когда проект только начинается, а кодовая база пуста. Тогда мы просто настраиваем анализатор и робот и начинаем кодировать. В нашей ситуации мы не можем настроить робота, так как он будет отклонять изменения абсолютно каждого, потому что анализатор сообщит о куче проблем. И мы не можем исправить их все, потому что их слишком много.

**Статический анализатор — привратник перед репозиторием кода, отклоняющий те изменения, которые нарушают какое-либо правило.**

[1] *Bugayenko Y.* How to Prevent SVN Conflicts in Distributed Agile PHP Projects. php|Architect, August 2010.

— Стало быть, это не прокатит?

— Похоже на то. Но и просто договориться с Томом тоже не решение проблемы. Ты с ним условишься об определенных методах кодирования, а завтра кто-то еще раз нарушит твой код. Все эти «руководства по стилю кодирования» — просто слова, если хочешь знать мое мнение. Я не верю в соглашения и конвенции, если они не выполняются и не автоматизируются беспристрастными роботами. Люди хорошо умеют врать, а вот роботы на это пока не способны.

— Очень скоро уже будут способны, — смеется он. — Слушай, мне все-таки очень нравится твоя идея, давай подумаем, как мы можем ее реализовать. Как насчет того, чтобы начать с небольшого набора правил? Вместо того чтобы задействовать те инструменты, о которых ты упоминал, в полную силу, можем ли мы настроить их так, чтобы они сообщали только о нескольких типах проблем? Например, можем ли мы просто убедиться, что используем четыре пробела везде вместо табуляций? Вот веришь или нет, но в некоторых частях нашего кода, довольно старых, где-то все еще есть табуляции. Я толком и не знаю, кто их добавил, однако регулярно приходится исправлять их.

— Технически можно настроить всего несколько правил. Возможно, ты прав. Можем попробовать. И нам нужно создать предполетный бот, который будет объединять ветки. Или мы можем взять какой-нибудь уже существующий. Я подумаю о том, какой вариант для бота мы могли бы выбрать. А ты пока попробуй выяснить, как можно настроить какой-нибудь анализатор, чтобы определить только один тип проблемы — табуляции.

— Понял, сделаю.

В таких сценариях статический анализ используется редко. В большинстве проектов, которые я видел, это просто набор красивых графиков и отчетов, которые показывают, сколько проблем в коде, и иногда выдают ряд предложений о том, как их можно исправить. Крайне редко команды разработчиков программного обеспечения превращают статические анализаторы в привратников, которые мо-

гут отклонять их код. Всегда важнее быстрее выполнить поставку, а не обеспечить качество.

Скорость поставки — наиболее важный показатель программиста. Но этот показатель имеет смысл только тогда, когда уже существуют довольно строгие правила, которые затрудняют их нарушение.

Это похоже на спортивные соревнования. Возьмем, например, бокс. Для каждого боксера цель очевидна — сделать как можно больше эффективных ударов. Ну, точнее, нанести их больше, чем противник, за ограниченное время. Выглядит довольно легко: бейте побольше — и выиграете. Однако существуют очень строгие правила, которые не позволяют боксу превращаться в уличную драку. Без правил бокс больше не был бы спортом и никто не захотел бы участвовать в поединке, поскольку это очень опасно. Вместо того чтобы становиться великими спортсменами, боксеры были бы жестокими и агрессивными обезьянами. Цель «сделать как можно больше ударов» выполнима тогда, когда есть строгие правила. Иначе это просто испортит игру.

**Программисты должны быть заинтересованы в том, чтобы вносить изменения как можно быстрее, но проект должен отклонять их максимально активно (насколько это возможно).**

То же самое относится и к разработчикам программного обеспечения. Они должны быть мотивированы поставить свой код как можно быстрее. Для них не должно быть никаких других целей. В идеале нужно платить за каждую поставку, и чем быстрее они происходят, тем больше денег должны получать работники. Однако этот подход будет эффективным только в том случае, если сохраняется высокое качество проекта. Один из лучших методов поддержания качества, который должен применяться к каждому новому фрагменту кода, — статический анализ.

Если правила анализа очень строги, а программистам платят за то, чтобы они выполнялись как можно быстрее, качество проекта будет высочайшим. Всегда должен быть конфликт интересов между мотивацией программистов («быстро разбогатеть») и мотивацией

проекта («держать кодовую базу в чистоте»). Если одна из этих мотиваций недостаточно сильна, возникнут серьезные проблемы.

Если программисты инициативны и стремятся больше заработать, а в проекте не предусмотрены статический анализ, модульные тесты, проверка кода и автоматические конвейеры поставки, то база кода довольно быстро будет испорчена. А с другой стороны, если проект полностью защищен и отклоняет все, что ставит под угрозу его качество, но программисты не заинтересованы в продвижении своего кода или просто слишком ленивы, то разработка в конце концов застопорится.

# Скорость и качество

Джиоти о чем-то беседует с Адрианом. У нее всегда немного грустный и разочарованный тон. Я могу ее понять. Быть тестировщиком в наше время унизительно[1], поскольку компании не совсем понимают истинную ценность тестирования. Инженерам-тестировщикам они платят меньше, чем программистам[2], и нанимают в основном новичков или даже стажеров. Тестирование выглядит как побочный эффект программирования[3].

А должно быть наоборот, ведь работающее программное обеспечение — это только третья часть готового к использованию продукта, программирование в два раза менее важно, чем тестирование. Чтобы превратить работающую часть кода в продукт, его необходимо тщательно протестировать и сообщить об ошибках. Без тестирования ошибки будут обнаружены пользователями, что

---

1 *Weyuker E. J. et al.* Clearing a Career Path for Software Testers. IEEE Software, Volume 17, Number 2, 2000: «Притом что тестирование программного обеспечения крайне важно для успеха продукта, в большинстве организаций оно не пользуется большим уважением. ...Поскольку тестирование программного обеспечения иногда рассматривается как бесполезная работа, наиболее опытные тестировщики, как правило, стремятся вырасти до программистов, аналитиков или системных архитекторов».

2 По данным PayScale, на 26 апреля 2018 года средняя зарплата программиста в США составляет 60 785 долларов США. Средняя зарплата тестировщика программного обеспечения — 55 453 доллара (на 10 % ниже).

3 *Glass R. L. et al.* Software Testing and Industry Needs. IEEE Software, Volume 23, Number 4, 2006: «Работа тестировщиком часто выглядит как утешительный приз для тех, кого не считают подходящими для работы инженером-программистом, и разрыв в заработной плате между разработчиками и тестировщиками остается значительным. Тестировщики часто жалуются на недостаток уважения, доверия и влияния. Карьера развивается скорее случайным образом, чем по запланированной схеме. Большинство тестировщиков — самоучки, многие никогда не читали книг на эту тему. Не существует универсальной терминологии и базы данных для тестирования. Обучение в университете посредственное. Основные программы сертификации тестировщиков могут поставить под угрозу будущее тестирования — в них больше внимания уделяется второстепенным понятиям».

серьезно повлияет на степень их удовлетворенности. Тестирование и тестировщики продвигают продукт, делая его готовым к продаже. Программисты только помогают им исправлять ошибки. Это можно воспринимать как шутку, и в этом утверждении есть доля юмора, но также есть доля правды.

Несмотря на очевидную важность тестирования программного обеспечения, тестировщики в большинстве команд разработчиков считаются сотрудниками второго сорта.

Тестирование — более сложный процесс, чем написание кода[1]. Тестировщики «ломают» продукт, находя его дефекты. Программисты же создают его с этими дефектами. Ответственность программистов намного ниже, чем тестировщиков. Если тестировщик пропустит что-то важное, самолет упадет. Если программист сделает ошибку, то при наличии ряда тестировщиков эту ошибку можно будет выловить.

Профессиональный отдел тестирования — это не группа обезьян, нажимающих один и тот же набор кнопок снова и снова целый день. Напротив, это команда инженеров, способных находить уязвимости, автоматизировать процесс поиска[2], создавать удобные для прочтения и нахождения отчеты[3] и сообщать о проблемах так, чтобы их могли понять программисты. Инженеры по тестирова-

---

1 *Patton R.* Software Testing. Sams Publishing, 2000: «На первый взгляд может показаться, что работа тестировщика программного обеспечения проще, чем работа программиста. Ломать код и находить ошибки, безусловно, должно быть проще, чем писать код. Но, как ни странно, это не так».

2 *Dustin E. et al.* Automated Software Testing: Introduction, Management and Performance. Addison-Wesley Professional, 1999: «Значительная часть работы по тестированию, необходимой для проекта, теперь должна проводиться с помощью инструментов автоматического тестирования. Ручное тестирование трудоемкое и подвержено ошибкам, не поддерживает проверку качества высокого уровня, которая возможна благодаря автоматизированным инструментам».

3 *Bettenburg N. et al.* What Makes a Good Bug Report? Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2008: «Хорошо написанные отчеты об ошибках, вероятно, привлекут больше внимания разработчиков, чем плохо написанные».

нию — тестировщики — должны зарабатывать даже больше, чем программисты. Разве не печально то, что в индустрии в большинстве случаев ищут нетребовательных к оплате труда джуниоров?

Они разговаривают довольно эмоционально, и я пытаюсь понять, о чем речь.

— Это не ее вина, — твердит Бао Адриану. — Мы сами должны лучше тестировать код. Хорошие программисты обязаны нести ответственность за код, который пишут.

— Да, я согласна, — подтверждает она.

— Я знаю, — кивает Адриан, — но ошибки неизбежны.

— Вы могли бы быть аккуратнее, ребята, — грустно говорит она.

— Что случилось? — я встреваю в разговор.

— Мы обсчитали более 200 клиентов из-за ошибки, которую вы сделали вчера, — объясняет Бао. — Теперь мы должны вернуть кучу денег.

— Мы уже откатили изменения? — спрашиваю я у Адриана.

— Да, полчаса назад.

— Так получается, что это наша вина? — спрашиваю я у Бао.

— Нет, это моя вина, — саркастически улыбается он. — Что это за вопрос такой? Народ, вы должны лучше тестировать код и серьезнее и ответственнее относиться к тому, что пишете. Мы работаем с деньгами, помните об этом. Не вините Джиоти, она только помогает вам улучшить продукт, но главная ответственность лежит на вас.

**Профессиональная команда всегда знает, кто виноват в допущенных ошибках, и не боится открыто признать это.**

Он доволен, что мы сделали ошибку, и мало беспокоится о вреде, который мы нанесли клиентам. Это понятно: каждый раз, когда мы проигрываем, в глазах Тони он растет.

— Я не думаю, что это правда, — мой черед улыбаться.

— Что не правда?

— Я не думаю, что ответственность программиста заключается в создании кода без ошибок. Это очень серьезное и фундаментальное

заблуждение. Даже не заблуждение, а подход. Такой подход приведет только к тому, что ухудшит качество и покажет пользователю больше ошибок.

— Опять ваши философские беседы? — Бао нервничает. — У меня есть дела поважнее. Народ, вы меня разочаровываете, — бросает он, уходя.

— Он всегда такой, не волнуйся, — шепчет Адриан.

Я спокойно продолжаю:

— Я знаю, не переживай. Слушай, я сказал это на полном серьезе. Ожидать, что программисты создадут код, в котором не будет ошибок, — глупая затея[1]. Это не работа программиста — отлавливать ошибки и исправлять их. Программист создает новый код, исследует новые возможности для оптимизации, рефакторинга, делает код быстрее, элегантнее и все такое. Программист не должен беспокоиться об ошибках, которые клиенты могут увидеть. Программист должен беспокоиться только о скорости поставки. Чем быстрее закрываются тикеты, тем профессиональнее программист. Вот и все. То, что произошло вчера, было провалом нашего конвейера поставки[2].

**Ожидать от программистов написания кода без ошибок — ужасная и очень распространенная ошибка.**

— Моя ошибка? — с грустью смотрит на меня Джиоти.

— Я не хочу показывать на кого-то пальцем или обижать кого-либо, это была наша общая неудача, а не какого-то отдельного человека... Кстати, кто допустил ошибку?

— Это был Чунг, — вздыхает Адриан. — Он еще недостаточно опытный. Я зря дал ему эту задачу.

— Нет! То есть да, твоя ошибка есть, но не в том, чтобы дать ему задание, а в том, что не удалось правильно настроить конвейер поставки.

---

[1] ❦ *Munson J. C.* Software Engineering Measurement, CRC Press, 2003: «Хороший программист напишет безупречный код, а плохой — код со множеством ошибок».

[2] ★ *Humble J. et al.* Continuous Delivery: Reliable Software Releases through Build, Test and Deployment Automation, Addison-Wesley Professional, 2010.

— Что такое конвейер поставки? — спрашивает Джиоти.

— Это вся цепочка процедур, которые мы выполняем между моментом, когда программист создал на своем компьютере новый фрагмент кода, и моментом, когда наши клиенты видят новую функциональность в сети. Это называется конвейером поставки. Он включает несколько этапов: объединение веток, тестирование кода, развертывание на промежуточном сервере, ручное тестирование, развертывание на производственном сервере и некоторые другие. Цель этого конвейера — максимально изолировать программистов от заказчиков и создать туннель препятствий, который отфильтрует проблемный код. Конвейер должен работать как решето, через которое смогут пройти только хорошие фрагменты кода. Все остальное будет возвращаться на доработку[1]. Когда клиент жалуется на ошибку (или многие клиенты жалуются, как случилось сегодня) — это проблема с решетом, а не с программистами.

Джиоти кивает и спрашивает:

— А кто несет ответственность за решето?

Я указываю на Адриана:

— Прежде всего он.

Адриан улыбается. Он, очевидно, знает, о чем я говорю, но прекрасно понимает, что его босс Тони не имеет ни малейшего представления о конвейере поставки и просто хочет, чтобы программное обеспечение «работало». Именно поэтому Адриан воспринимает все как интересную беседу без каких-либо реальных последствий.

— Ну, я не говорю, что Адриан — единственный человек, который должен разработать конвейер. Мы все должны участвовать. На самом деле у нас уже есть некоторые элементы, такие как

---

[1] *Nygard M. T.* Release It!: Design and Deploy Production-Ready Software. 2nd Edition, Pragmatic Bookshelf, 2018: «Мы называем это конвейером сборки, но это больше похоже на воронку сборки. На каждом этапе конвейера сборки выполняется поиск причин отклонить сборку. Тесты провалились? Отклоняем. Линт жалуется? Отклоняем. Сборка не проходит интеграционные тесты на промежуточном сервере? Отклоняем. Готовый архив нелепо выглядит? Отклоняем».

автоматизированная сборка, некоторые модульные тесты и ваш отдел. В будущем их станет намного больше, будут еще статический анализ, интеграционные тесты, возможно, некоторые автоматизированные A/B-тесты, тесты производительности и нагрузки, даже сине-зеленое развертывание[1]. Как видишь, ваш отдел — тоже очень важный элемент конвейера. Ошибку, с которой столкнулись клиенты сегодня утром, должны были обнаружить ваши тестировщики. Обрати внимание — программисты не являются частью конвейера поставки. Они просто создают вход для этого. То, что происходит после того, как они заканчивают создание и изменение кода, — это и есть конвейер. Так что если мы проанализируем этот конкретный инцидент, получается, что мы не должны обвинять Чунга в совершении ошибки, мы не должны обвинять Адриана в том, что он дал Чунгу задачу, с которой тот не справился, но мы можем обвинить Адриана в том, что он не сделал конвейер достаточно мощным, и Джиоти в том, что не была достаточно сильным звеном в этом конвейере.

— Получается, что это моя личная вина? — снова спрашивает она.

— Не принимай на свой счет, лично никто не виноват, — подключается Адриан.

— Ну, я бы сказал, что личная, — перебиваю я, — но скорее в профессиональном плане. Это не твоя вина как личности. Ты хороший человек, тебе небезразлично, чем мы здесь занимаемся, ты стремишься узнавать что-то новое. Никто не говорит, что это лично твоя вина. Я бы сказал, что это вина нашего отдела тестирования. Сейчас ты возглавляешь его, так что это недостаток твоего управления. Но теперь ты знаешь, где есть проблема, и сможешь ее исправить.

— Как я могу это исправить?

— С ходу не скажу, но мы можем обсудить это потом. Проблема очевидна, верно? — я смотрю на нее, и она кивает. — Наше тестиро-

**Слабый конвейер поставки всегда виноват во всех проблемах, с которыми сталкиваются клиенты.**

[1] *Fowler M.* BlueGreenDeployment. 2010, https://goo.gl/GQsavF.

вание слабовато. Это позволяет ошибкам все же иногда проскальзывать и доставлять неудобства клиентам. Мы должны найти способ улучшить процесс.

— Может быть, лучше прекратить делать ошибки? — спрашивает Джиоти, и мне интересно, действительно ли она не слушала или она все же в чем-то права. Может быть, это все-таки мы виновны в ошибках, которые делаем?

Я уже сталкивался с подобным подходом прежде и не уверен, что с этим действительно можно что-либо сделать. Руководители привыкли думать, что «отсутствие ошибок» означает высокое качество[1]. А «высокое качество» — это когда пользователь не видит никаких ошибок. Есть несколько способов добиться этого. Наиболее очевидный (и при этом худший) — попросить нас допускать меньше ошибок[2]. Что можно сделать, чтобы заставить нас писать программное обеспечение с меньшим количеством ошибок? Нас могут обучать, нас могут просить или даже умолять, нам могут угрожать, нас могут даже заменить другими программистами, которые должны быть «лучше».

Все это может сработать. Количество ошибок может снизиться, особенно если нас хорошенько запугать. Однако скорость поставки серьезно пострадает. Мы будем больше заботиться о качестве, а не о скорости, и этот процесс станет абсолютно неуправляемым. У нас всегда будет отмазка, что качество еще слишком низкое, чтобы выпустить продукт.

Руководители будут полагаться на нас в области качества и не станут вкладывать достаточно средств в конвейер поставки. Они будут считать, что наша работа — создавать качественный код,

---

1 *Maguire S.* Writing Solid Code. Greyden Press, 2013: «Учитывая растущую сложность программного обеспечения и связанное с этим увеличение количества ошибок, программистам необходимо создавать код без ошибок намного раньше — на этапе разработки, перед тем как код будет передан тестировщикам».

2 *Panko R. R.* Spreadsheets on Trial: A Survey of Research on Spreadsheet Risks. Proceedings of the Twenty-Ninth Hawaii International Conference on System Sciences, 1996: «Мы не делаем ошибки постоянно, но они все равно иногда проскакивают, даже если мы стараемся быть аккуратными».

а проверять наши результаты и отфильтровывать ошибки вовсе не обязательно.

Правильным будет другое решение: мы не должны нести ответственность за качество в целом. Забота о клиентах и ошибках кода — не наше дело. Наша работа — создавать код. Работа руководителя состоит в том, чтобы правильно организовать систему обеспечения и контроля качества, отклоняющую неработающие элементы и принимающую только работающие.

> Самая важная черта хорошего программиста — способность быстро вносить изменения, проходя все проверки качества.

Единственный действительный показатель, который должен быть у программиста, — это скорость поставки. Чем больше тикетов я закрываю за день, тем я более крутой программист. Конечно, закрытие тикета означает прохождение всех возможных проверок качества. Моя работа не в том, чтобы писать что-то высокого качества, а в том, чтобы писать код, способный пройти эти проверки. Многие ли руководители в состоянии противостоять реальности и применять этот подход?

Очевидно, что у каждой компании и каждого проекта есть свои уникальные стандарты качества. В одних проектах критична только скорость, так как их цель — создать прототип, привлечь инвестиционный капитал и переписать весь продукт с нуля. Так почему я должен производить что-то высококачественное, если у такого проекта нет никаких проверок в конвейере поставки? Не потому, что это плохо, а потому, что это их бизнес-кейс. Я бы просто выпустил такой проект как можно быстрее.

У другого проекта может быть очень строгая система контроля качества с конвейером поставки, который начнет отклонять практически все, что не удовлетворяет всем возможным проверкам качества, и который будет практически нереально обмануть. Такой проект может быть в сфере здравоохранения, где любая ошибка может стоить кому-то жизни. В таком проекте я бы старался проверить все как можно быстрее, сделав все возможные ошибки, а когда код был бы отклонен, мне потребовалось бы заново дораба-

тывать, полировать, улучшать его и т. д. Другими словами, должен быть постоянный конфликт между скоростью и качеством. Мы должны быть мотивированы, чтобы выпустить проект быстрее, в то время как компания должна быть мотивирована бизнес-ситуацией. Истина обычно где-то посередине.

**Баланс между скоростью и качеством должен определяться бизнес-кейсом, а не программистами.**

Когда у нас слишком много критических ошибок в производстве — это четкий сигнал о том, что пришло время укрепить систему контроля качества. Если конвейер слишком слабый, он пропускает много нерабочих элементов. Мы не должны обвинять программистов или менять их — это не поможет. Нам нужно добавить в конвейер новые фильтры.

Единственная причина заменить программиста — его медлительность. Все остальные причины — просто оправдания, используемые руководством, когда оно не может признать собственные ошибки. Как в случае с этим утренним инцидентом: ни Адриан, ни Джиоти не были готовы признать, что это их вина. Они хотели обвинить Чунга, ведь так проще. Разве не очевидно, что на самом деле виновниками были именно они?

# Конвейер поставки

Адриан смотрит на меня и улыбается:

— А в принципе, Джиоти права. Нам следовало бы внимательнее писать код. Смотри, вот в отношении конкретно этой ошибки все было очевидно. Давай покажу тебе код, — открывает он ноутбук.

Джиоти смотрит на меня.

— Да не надо, я позже посмотрю. Кажется, ты упускаешь суть, — я стараюсь сохранять спокойствие. — Мы не должны обвинять Чунга, это не возымеет нужного эффекта. Более того, это опасно. Это будет неправильный месседж для всех нас. Все будут просто бояться, и при этом настоящая проблема никуда не денется.

— Что? — он закрывает ноутбук.

— Наш слабый конвейер поставки! — я стараюсь подчеркнуть слово «слабый». — Обвиняя и запугивая программистов, мы только замедляем развитие[1]. Люди будут бояться, не имея на самом деле никаких инструментов для избавления от стресса. Просто попросить всех «писать код аккуратнее» не очень поможет, потому что программисты никогда не знают, где возникнет следующая проблема, что ее вызовет и во сколько это обойдется. Это как ходить по минному полю: никогда не знаешь, какая часть кода взорвется. Мы просто перекладываем ответственность за качество на плечи программистов и говорим им: «Там есть мины, идите осторожно!» Представьте, что они чувствуют.

**Перекладывание ответственности за ошибки на программистов только вызывает стресс и замедляет разработку.**

— Ты преувеличиваешь, — улыбается Адриан.

— Ну, может быть, совсем чуть-чуть. Нам нужны продуктивные и эффективные сотрудники, которые могут писать большое количе-

---

[1] *Evans J.* Fear Makes You a Worse Programmer. https://goo.gl/MeLGxN, 2014: «Если вы боитесь вносить изменения в код, то не сможете что-либо усовершенствовать или навести в коде порядок. Возможно, вы даже не сможете развернуть код, который уже написали и протестировали, потому что он слишком пугает вас. Вы просто предпочитаете "не трогать работающую систему", даже если она далека от совершенства».

ство кода и не устают от этого. Принимая двусмысленные правила игры, мы убиваем их мотивацию и вызываем у них дистресс[1]. Такие программисты не будут писать совершенный код, они просто уйдут.

Джиоти спрашивает:

— Что такое дистресс?

Я отвечаю:

— Существует теория[2], что есть два типа стресса: эустресс и дистресс. Первый, хоть и вызывает дискомфорт, все же ведет к личностному росту, а второй имеет негативные последствия. Эустресс появляется и исчезает, может длиться некоторое время. Он держит нас в тонусе[3]. Однако постоянный стресс, от которого нельзя избавиться, просто адаптируясь к ситуации, — это дистресс, который в итоге приводит к беспокойству, изнеможению и депрессии[4].

**Чтобы избежать дистресса, нужно индивидуально и четко определить ответственность.**

— Интересно. Я не знала, что стресс может быть положительным.

---

1 *Stranks J.* Stress at Work: Management and Prevention. Elsevier, 2005: «Позитивный стресс — один из результатов грамотного управления и зрелого лидерства, когда все работают сообща и их усилия ценятся и поддерживаются. Это улучшает самочувствие и может быть использовано для повышения производительности и достижения лучших результатов. А вот отрицательный стресс, или дистресс, возникающий, например, из-за необходимости соблюдения установленных сроков или передачи ответственности, обычно приводит к ухудшению здоровья, и работодатели должны учитывать это в рамках стратегии управления стрессом».

2 *Hans S.* Confusion and Controversy in the Stress Field. Journal of Human Stress, Volume 1, Number 2, 1975.

3 *Nelson D. L. et al.* Health Psychology and Work Stress: A More Positive Approach. Handbook of Occupational Health Psychology, Volume 2, 2003: «Сотрудники, работающие в состоянии эустресса, вовлечены (это означает, что они работают с энтузиазмом и удовлетворены выполняемой работой). Они могут сохранять хорошее расположение духа, даже когда сталкиваются с сильными стрессорами».

4 *Quick J. C.* Preventive Stress Management in Organizations. 2nd Edition, American Psychological Association, 2013: «Реакция на стресс при правильном управлении приводит к эустрессу и повышению производительности. Однако есть и обратная сторона медали — дистресс, который развивается, когда неправильно управляют реакцией на стресс или когда она происходит не так, как надо».

— Может. Без стресса не было бы ни прогресса, ни роста, ни мотивации[1]. Нам определенно нужен стресс, чтобы чего-то достичь[2]. Однако, как только цель достигнута, стресс должен исчезнуть. Когда утром я вижу новую задачу или ошибку, то чувствую стресс: что-то не работает, для кого-то я должен создать что-то новое, я не уверен в результате, боюсь нарушить уже работающий код и т. д. Я начинаю работать над задачей, чтобы выйти из зоны стресса. Теперь представь, что я завершаю дело, мой код поступает в производство, но я не знаю, закончилась ли эта история или на следующее утро Бао появится в офисе с ухмылкой и обвинениями в том, что несколько сотен клиентов потеряли из-за меня номера своих кредитных карт. Видишь? — я делаю короткую паузу. — Выполнив задание, я не выйду из зоны стресса. Я останусь в таком состоянии и ночью, находясь дома, и утром по дороге в офис. Я буквально буду жить в стрессе. Мой стресс перерастет в дистресс, который может привести к серьезным проблемам со здоровьем, например к депрессии[3].

— Согласна! — восклицает она.

— Конечно, — соглашаюсь я, — эта проблема актуальна для всех нас. Мы все в дистрессе, и чем большую групповую ответственность мы берем на себя, тем менее продуктивными и более подавленными становимся. Быть ответственным за все — значит не быть ответственным ни за что и в то же время постоянно быть готовым к тому, что тебя будут во всем винить[4]. Это позиция раба, уж извините меня

---

[1] ★ *Latham G. P.* Work motivation: History, Theory, Research and Practice. Sage Publications, 2007.

[2] *Hargrove M. B. et al.* Generating Eustress by Challenging Employees: Helping People Savor Their Work. Organizational Dynamics, Volume 42, 2013: «Мы предлагаем руководителям иногда создавать здоровый стресс для своих сотрудников».

[3] *Hansen C. H. et al.* Screening Medical Patients for Distress and Depression: Does Measurement in the Clinic Prior to the Consultation Overestimate Distress Measured at Home? Psychological Medicine, Volume 43, Number 10, 2013.

[4] *Michie S.* Causes and Management of Stress at Work. Occupational and Environmental Medicine, Volume 59, Number 1, 2002: «Ситуации, которые могут вызывать стресс, как правило, непредсказуемые или неконтролируемые, неопределенные, двусмысленные или незнакомые либо связаны с вовлеченностью в конфликт, потерей или неудачей. ...Если стресс сохраняется, происходят изменения в нейроэндокринной, сердечно-сосудистой, вегетативной и иммунной системах, что приводит к ухудшению психического и физического здоровья».

за такие слова. А вот правильный подход: каждый несет строгую ответственность именно за свои задачи и сферы контроля. Меня не должно беспокоить то, что происходит после завершения моей задачи. Мой код слит в master? Все, я свое дело сделал. Это единственно возможное и правильное определение выполненной работы: если вы приняли мой код, не обвиняйте меня в каких-либо ошибках впоследствии, — я улыбаюсь и думаю о том, как бы это восприняли те, кто привык к прокоммунистическим постулатам о коллективной ответственности. Ведь что-то подобное прогнозировал Маркс в обществе будущего?

Адриан саркастически замечает:

— По-моему, это что-то сродни крайнему индивидуализму[1].

— Да, так и есть. Конечно, команда не сможет эффективно работать и даже просто существовать, если не будет своеобразного клея между эгоистичными и жадными индивидуумами[2].

— Да ладно, мы не эгоистичные и не жадные, — улыбается Джиоти.

— И очень жаль, — серьезно отвечаю я. — Мы привыкли работать из-за стресса, а не из-за жадности или интереса. И я не имею в виду только нашу команду. Я имею в виду всю отрасль. Посмотрите на ситуацию с Чунгом. Несколько минут назад вы пытались обвинить

---

1 *Triandis H. C.* Individualism-Collectivism and Personality. Journal of Personality, Volume 69, Number 6, 2001: «В индивидуалистических обществах люди автономны и независимы от своих групп, отдают приоритет личным целям, а не целям своих групп, их поведение основывается в первую очередь на своем отношении, а не на нормах их групп и теория обмена адекватно прогнозирует их социальное поведение. ...Чем сложнее культура, тем более индивидуалистичной она может быть».

2 *Triandis H. C. et al.* Allocentric Versus Idiocentric Tendencies: Convergent and Discriminant Validation. Journal of Research in Personality, Volume 19, Issue 4, 1985: «Те, кто аллоцентричен (то есть концентрируют свое внимание и действия на других людях, а не на себе), с большей вероятностью будут подчеркивать ценность сотрудничества, равенства и честности, а те, кто идиоцентричен (то есть стремится сосредоточиться на собственных целях и потребностях), подчеркивают ценность комфортной жизни, конкуренции, удовольствия и общественного признания. Те, кто был аллоцентричен, сообщили, что получили больше социальной поддержки и в лучшем качестве; те, кто был идиоцентричен, были более мотивированными к достижениям, более отчужденными, апатичными и сообщили о чувстве одиночестве».

бедного чувака в том, что из-за него мы потеряли утром деньги. Вся вина легла бы на его хрупкие плечи. Для него это был бы жуткий стресс на последующие несколько недель. И он стал бы очень легко управляемым, делал бы все, что мы ему скажем. Вообще, для любого босса это самый простой способ управлять своими людьми и контролировать их: подвергнуть их стрессу и сделать виноватыми. Они будут постоянно пытаться выбраться из зоны стресса и никогда не добьются успеха. Такая стретегия будет эффективна длительное время, пока работники не выгорят и не уйдут. Затем они отдохнут несколько месяцев и найдут нового босса, который тоже начнет вызывать в них чувство вины. В такой модели ключевой фактор успеха, — я изображаю обеими руками кавычки, — это групповая ответственность. Просто сделай нас ответственными за общий результат проекта — и мы будем в постоянном дистрессе.

— А как иначе ты это себе представляешь? Чтобы никто не нес ни за что никакой ответственности? Что это за команда? Я не понимаю, — говорит Джиоти, и я чувствую: она просто хотела бы, чтобы я признал, что эта утренняя ошибка была нашей неудачей.

— Я не говорил, что никто ни за что не будет нести ответственность, не пойми меня неправильно. Я хочу, чтобы мы отвечали за наши индивидуальные задачи и наши личные ошибки. Давай посмотрим на ошибку, которую вчера совершил Чунг. Я объясню, как я хотел бы видеть ситуацию, а ты скажешь, что с ней не так. Сначала Адриан просит Чунга реализовать эту функциональность или исправить ошибку, я не знаю, что это было...

**Приписывание ошибки ее автору не делает его более ответственным — только более запуганным.**

— На самом деле он исправлял ошибку Денниса, — поясняет Адриан.

— Видишь, ты уже называешь ошибки именами сделавших их людей, это абсолютно неправильно! Это то, о чем я и говорю: мы не должны обвинять программистов в их ошибках. Ошибки могут принадлежать им только до тех пор, пока код не будет слит в репозиторий. После этого все ошибки наши! — воскликнул я и подумал, а не слишком ли опасно освобождать программистов от их личной ответственности за ошибки? Не создаст ли это еще больше проблем?

Джиоти улыбается:

— Ну хорошо, продолжай.

И я продолжаю:

— Итак, Адриан просит Чунга исправить нашу ошибку, — я подчеркиваю слово «нашу», и они оба улыбаются. — Чунг создает ветку[1] и начинает в ней работать. Он младший разработчик. Допустим, он понятия не имеет, что правильно и что неправильно в этом репозитории. Таким образом, он неизбежно допускает ошибки. Много ошибок. И некоторые из них очень серьезные. Но ему все равно, потому что он знает, что наш конвейер поставки супермощный и не даст серьезным проблемам проскочить в производство. Через несколько часов или дней он заканчивает свою работу и выполняет все модульные тесты. Некоторые из них красные, потому что он сделал несколько ошибок. Он исправляет ошибки — если может. Если не может, просит помощи у кого-то из коллег. Не знаю, что именно он будет делать. Может быть, он спросит в StackOverflow, может, попробует найти решение проблемы в проектной документации — это не так важно. Важно то, что на данном этапе это его проблема. Все, что идет не так, — это его проблема. Если юнит-тесты красные — это его проблема. Если сборка нечистая — это его проблема. Он может попросить о помощи, но мы ему ничего не должны. Он полностью сам по себе.

— Как-то уж совсем сурово, — комментирует Адриан.

— Ну да, сурово. Для него это очень напряженное время. Он должен исправить ошибку, но толком не знает как. Чтобы выйти из состояния стресса, он должен увидеть, что его ветвь сливается в репозиторий. Что-то подобное чувствуют спортсмены, когда бегут стометровку. Им нужно добраться до финиша, и это очень стрессовое занятие. Но они точно знают, где находится финишная линия, и понимают, что сразу после нее стресс полностью исчезнет.

---

[1] *Bass L.* DevOps: A Software Architect's Perspective. Pearson Education, 2015: «Проблема выполнения всех разработок в одной основной ветке заключается в том, что разработчик одновременно трудится над несколькими различными задачами в одном модуле. Когда одна задача завершена, в модуль не может быть сделан коммит, пока не будут выполнены другие задачи, иначе в конвейер развертывания окажется добавлен неполный и непроверенный код для новой функциональности».

— Любишь ты метафоры, — улыбается Джиоти.

— Они помогают нам лучше понять суть, верно? Итак, вернемся к истории с Чунгом. У него есть ветка, и он проводит в ней все необходимые изменения. Он знает: чтобы принять его ветку, наш конвейер поставки должен убедиться, что изменения не нарушают наши правила и не снижают планку качества. Он знает правила: сборка должна быть чистой, все модульные и интеграционные тесты должны быть зелеными, не должно быть нарушений статического анализа и, конечно, нужно пройти проверку кода. Чунг прилагает все необходимые усилия для прохождения своего кода через эти барьеры. Он знает, что сразу после того, как финишная черта пересечена и ветка попала в хранилище, он свободен. Так что в конце концов это случилось. Мы принимаем его ветвь, объединяем с master — и все, после этого он свободен. Мы даже как-то отмечаем это событие. Можно даже дать ему премию, скажем, 20 долларов.

**Что нас больше всего мотивирует? Отдельные задачи, у каждой из которых есть явное завершение.**

Они оба улыбаются.

— Люди работают не только ради денег, знаешь ли, — говорит Джиоти. — Есть исследования[1], подтверждающие, что связь между зарплатой и удовлетворенностью работой очень слабая. Более того, некоторые утверждают[2], что, когда награда ощутима и предсказуема,

---

[1] *Judge T. A. et al.* The Relationship Between Pay and Job Satisfaction: A Meta-analysis of the Literature. Journal of Vocational Behavior, Volume 77, 2010: «Доход приносит людям лишь слабое удовлетворение, даже если они оценивают его только лишь своей заработной платой».

[2] *Deci E. L.* A Meta-Analytic Review of Experiments Examining the Effects of Extrinsic Rewards on Intrinsic Motivation. Psychological Bulletin, Volume 125, Number 6, 1999: «Тщательный анализ последствий вознаграждения, описанных в 128 исследованиях, приводит к выводу, что материальные вознаграждения имеют тенденцию оказывать отрицательное влияние на внутреннюю мотивацию с указанными нами ограничивающими условиями. Даже когда материальное поощрение предлагается за хорошую работу, оно, как правило, снижает внутреннюю мотивацию и инициативу работника».

внутренняя мотивация уменьшается. Иными словами, в наши дни людям не нравится работать просто за деньги, особенно программистам[1]. И даже если они работают только за деньги, то внешние вознаграждения, как правило, со временем становятся все меньшими мотиваторами[2].

— Да, я тоже слышал об этом, но полагаю, что большинство этих исследований[3] не противоречат старому доброму методу кнута и пряника. Некоторые из них на самом деле подтверждают[4], что именно больше всего мотивирует нас контролировать свои успехи и неудачи. Некоторые ученые оспаривают[5] утверждение, что внешние поощрения наносят ущерб внутренней мотивации. Посмотри на такую стрессовую ситуацию, как спортивные соревнования. Спортсмены обладают большой мотивацией и могут добиться отличных результатов, если у них очень строгие правила игры и если у них есть кнут и пряник.

**Лучший мотиватор — это набор очевидных и справедливых правил с достойным вознаграждением.**

---

[1] *Beecham S. et al.* Motivation in Software Engineering: A Systematic Literature Review. Information and Software Technology, Volume 50, Issues 9–10, 2008: «Хорошая зарплата может быть мотивацией в нестабильной среде и только в начале карьеры инженера, хотя обычно считается, что зарплата и есть тот фактор, который удерживает сотрудника на работе».

[2] *Stipek D. J.* Motivation and Instruction. Handbook of Educational Psychology, Macmillan, 1996.

[3] *Lai E. R.* Motivation: A Literature Review. Person Research's Report, 2011: «Материальные вознаграждения могут быть особенно вредны для внутренней мотивации, равно как и отрицательная обратная связь по результатам работы и положительная обратная связь при ее контролируемом управлении».

[4] *Eccles J. S. et al.* Motivational Beliefs, Values and Goals. Annual Review of Psychology, Volume 53, 2002: «Непонимание причин своих успехов и неудач подрывает мотивацию работать над соответствующими задачами».

[5] *Hidi S. et al.* Motivating the Academically Unmotivated: A Critical Issue for the 21st Century. Review of Educational Research, Volume 70, Number 2, 2000: «Хотя мы признаем положительный эффект личной заинтересованности и внутренней мотивации, мы призываем преподавателей и исследователей учитывать потенциальные дополнительные преимущества от вызванного внешними факторами ситуативного интереса и внешней мотивации».

Если у нас будет что-то похожее для программистов, они будут только благодарны.

— Зачем тогда давать Чунгу 20 долларов? — улыбается она.

— На мой взгляд, лучшим мотиватором будут четкие правила игры, когда никто не сможет обманывать, в сочетании с сильной конкуренцией[1]. Или, как сказали бы психологи, возможность контролировать свои успехи и неудачи. Деньги в качестве награды можно использовать только тогда, когда правила абсолютно прозрачны и недвусмысленны. Цель[2] — не деньги сами по себе, а правила.

— Если это так очевидно, как ты говоришь, почему большинство других разработчиков программного обеспечения не ис-

---

[1] *Deutsch M.* Cooperation and Competition in The Handbook of Conflict Resolution: Theory and Practice. Jossey-Bass, 2006: «Конкуренция может варьироваться от деструктивной до конструктивной: недобросовестная, нерегулируемая конкуренция в деструктивной части; честная, регулируемая конкуренция между ними; конструктивная конкуренция в позитивной части. В конструктивном соревновании выигрывают не только победители, но и проигравшие. Таким образом, в теннисном матче, который принимает форму конструктивного соревнования, победитель предлагает, как проигравший может улучшить свои навыки, дает возможность проигравшему практиковать их, делает матч ценным опытом. В конструктивном соревновании победители следят за тем, чтобы проигравшие стали лучше (или по крайней мере не хуже), чем были до соревнования».

[2] *LeDuc Jr. A. L.* Motivation of Programmers. SIGMIS Database, Volume 11, Number 4, 1980: «Деньги как внешний мотиватор — бесконечно увлекательная тема. Если и есть успешный внешний мотиватор, то, казалось бы, это деньги; такой мотиватор ощутим, поддается количественной оценке и может использоваться для удовлетворения личных потребностей. В течение многих лет в сфере программирования было видно, что деньги выступали сильным мотиватором, подталкивали людей выбирать эту профессию — словно золотая лихорадка. Этот мотиватор перемещал их с работы на работу и поощрял за труд. Но вот все изменилось. В настоящее время студент-дипломник стремится программировать скорее потому, что это весело, или потому, что это умение открывает возможности для трудоустройства, дает уверенность в завтрашнем дне, а не потому, что предлагает богатство. Как считают многие руководители, для большинства людей деньги являются очень плохим внешним мотиватором. Даже если они выступают мотиватором, то со временем теряют свою мотивационную силу. В этом случае они могут быть хороши только для привлечения внимания».

пользуют такой подход? По крайней мере я не видела ни одной компании, где бы применялись такие строгие правила мотивации. Почему?[1]

— Хороший вопрос. Да, строгие и прозрачные правила полезны для целеустремленных и амбициозных программистов, которым нравится, когда им бросают вызов, — это игроки A[2], как их называет Джек Уэлч, и они составляют всего 20 % любой команды (в соответствии со знаменитым принципом Парето). Остальная часть команды состоит из игроков B (70 %), которые работают адекватно, и игроков C (10 %), которые вообще не работают. Добавь к этому еще слабое и некомпетентное руководство — и вот результат: дисциплину не соблюдают. Большинство не хочет, чтобы им бросали вызов, не хочет следовать правилам и требованиям к качеству. Вместо этого они хотят делать как можно меньше, получать деньги и идти домой, чтобы смотреть телевизор. Руководство слишком слабо, чтобы бороться с большинством и продвигать правила. Результат очевиден: программисты управляют своим начальством, а не наоборот. Я смотрю на них и думаю: неужели только я считаю, что это плохо? Может быть, большинство команд и программистов считают идеальной ситуацию, когда ими никто не управляет?

---

[1] *Licker P. S.* The Japanese Approach: A Better Way to Manage Programmers? Communications of the ACM, Volume 26, Number 9, 1983: «Вместо заботы о человеке в целом и его отношениях с сотрудниками руководство предлагает эквивалентную зарплату, узкую специализацию и быстрое устаревание знаний и умений. При целостном подходе руководители будут широко рассматривать рабочую среду, а не фокусироваться на специализированных задачах, таких как создание кода или его отладка. В этом случае программист будет отвечать за создание программ; руководитель — за то, чтобы убедиться, что программист сделал то, что нужно. Но большинство программистов работают не в целостной среде. Одни программисты, которые постоянно совершенствуются, могут предпочесть, чтобы их поменьше контролировали. Другие переходят в системный анализ. Остальные, как предполагает Крафт, становятся безликой массой программистов второго сорта, единственной мотивацией к работе которых будут деньги».

[2] *Welch J. et al.* Jack: Straight from the Gut. Business Plus, 2003: «Работа должна быть увлекательным занятием. Но для многих людей это просто работа».

Джиоти вздыхает:

— Я не на 100 % согласна с тобой, но давай вернемся к истории с Чунгом и конвейером. Мне действительно интересно. Что должно было произойти дальше?

— После добавления своей ветки он получил 20 долларов и может выбирать следующую задачу или ошибку, которую нужно исправить. Это больше не его код, мы никогда ему его не вернем, если вдруг возникнут проблемы, связанные с той частью, которую он только что добавил или изменил, — я смотрю ей прямо в глаза. — Никогда!

— Хорошо, — смеется она, — никогда так никогда!

Похоже, она меня понимает, но я все равно повторяю:

— Мы приняли код, наша серия проверок качества подтверждает, что все в порядке, поэтому у нас больше нет никаких оснований называть его кодом Чунга. Мы с ним закончили. Первый этап конвейера поставки успешно пройден.

— Что дальше? — спрашивает она.

— Есть немало возможных сценариев[1], но я предпочитаю следующий: в какой-то момент ваша команда делает ветку из master и называет ее релизной веткой. Вы развертываете код из этой ветки в промежуточной веб-среде. Он должен быть очень похож на производственный[2]. Затем вы начинаете тестировать и «ломать» его.

---

[1] *Mitchell L.* Git Branching Strategies. net, Issue 267, 2015.

[2] *Everett G. D.* Software Testing: Testing Across the Entire Software Development Life Cycle. John Wiley & Sons, 2007: «Цель среды тестирования — заставить тестируемое приложение продемонстрировать истинное поведение при эксплуатации, проводя наблюдения и измерения вне его производственной среды. Достижение этой цели может быть таким же сложным, как разработка и выполнение самих тестов. ...Следующий тип среды тестирования, который вы можете найти, называется промежуточной средой, миграционной средой или средой развертывания. Разработчики рассматривают эту среду как следующее место, куда их программы отправляются перед тем, как поступят в эксплуатацию. Она представляет собой точку сбора всех написанных программ. Кроме того, это отличная среда для тестирования написанных программ, так как готовые компоненты становятся доступными, а само приложение — написанным».

— «Ломать»? — переспрашивает она.

— Да, мы уже об этом говорили, помнишь? Смысл тестирования — «сломать» тестируемый продукт. Ваша цель — найти как можно больше ошибок и сообщить о них. Чем больше вы найдете, тем лучше. Но вы должны учитывать определенные цифры. Скажем, следует тестировать и «ломать», пока не найдете десять критических дефектов или пока количество обнаруженных дефектов за день не станет меньше пяти. Как-то так. Я не уверен насчет точных цифр, но должны быть измеримые критерии готовности. Нам нужно точно знать, когда вы прекратите тестирование и продукт будет готов к поставке.

— Разве я не могу просто сказать тебе, когда больше нет ошибок? Это мы и делаем сейчас — тестируем, чтобы убедиться, что ошибок нет.

Я вижу, что либо она не поняла все, что я говорил ей прежде, либо, может быть, она меня просто троллит.

— Это неправильный подход! В любом продукте есть ошибки. В любом, Джиоти, в любом! — восклицаю я, а она улыбается.

Возможно, она думает, что я немного не в себе.

— Неправильно говорить или предполагать, что в программном обеспечении нет ошибок. Там есть ошибки, причем много. Вопрос только в том, кто и когда их обнаружит. Задача нашей команды — выявлять и устранять их раньше, чем их увидят наши клиенты. Таким образом, подтверждение того, что программное обеспечение не содержит ошибок, противоречит нашей основной цели. Нам нужны ошибки, а не их отсутствие. Если их нет тогда, когда продукт еще находится в наших руках, это означает одно: они появятся, когда он будет у клиентов. Понимаешь?

— Похоже, да...

Я чувствую, что она говорит это, просто чтобы успокоить меня.

— Хорошо, тогда ваша цель — тестировать, устранять ошибки и сообщать об этом. Ваша основная мотивация — поиск ошибок, ничего более. И важно, что чем больше вы найдете, тем лучше. Я бы даже предложил платить вашей команде за них.

— Двадцать долларов! — встрял Адриан.

— Вы, похоже, очень скептически относитесь к денежным поощрениям, но я говорю вам, что это может сработать, если система управления спроектирована правильно. В любом случае забудь про 20 долларов. Мое мнение таково: Джиоти должна быть мотивирована искать ошибки и их количество должно иметь первостепенное значение. Мы должны стимулировать ее отдел показывать нам большие цифры. В какой-то момент им будет все труднее находить ошибки в новом релизе. Мы все знаем, что ошибки там все еще остаются, но стоимость поиска каждой из них с каждым днем становится выше. В какой-то момент, когда цифры достаточно велики, мы принимаем решение, что релиз готов.

**Продукт готов к выпуску, когда тестировщики не могут больше найти ошибок, несмотря на явную мотивацию их деятельности.**

Хотя я никогда не видел, чтобы руководители команд разработчиков программного обеспечения проповедовали такую концепцию выпуска продукта, они работают в соответствии с ней. Продукт поступает в эксплуатацию, когда руководитель или вся команда (если они не подчеркивают роль руководителя), чувствуют, что пришло подходящее время для этого. Они не собирают метрики, не знают, сколько ошибок было найдено, очень часто они даже не отслеживают их. Но их предчувствия основаны на показателях. Они чувствуют, что продукт готов, когда поток ошибок «уже не столь критичен». Может быть, этого «чувства пресыщения» достаточно и нам не нужны никакие показатели?

Адриан интересуется:

— Погоди, а как насчет исправления ошибок?

— Ах да, я забыл. Итак, обо всех найденных ошибках Джиоти сообщает нам, и в отчете для каждой ошибки указан номер релиза, соответствующий названию ветки Git. Ошибки отправляются программистам, их приоритет наивысший по сравнению со всеми другими тикетами. Ну, вопрос с приоритетом не всегда однозначен. Возможно, некоторые из них могут быть незначительными, может быть, их не нужно исправлять срочно или вообще никогда не нужно

использовать. Это решение по каждой конкретной ошибке должны принимать руководители. Я имею в виду тебя, Адриан, это должно быть твое решение. Работа Джиоти — находить ошибки, наша работа — их исправлять. Каждое исправление должно быть объединено с веткой релиза, а также слито обратно в ветку master.

— Ты имеешь в виду бэкпортирование?[1]

— Ну, в этом случае это форвардинг. Изменения, объединенные в ветке релиза, также должны отображаться в master. Таким образом, вся ветка релиза должна быть объединена с master. Конечно, возникнут некоторые технические сложности, но мы обсудим их позже, когда придет время. Важно то, что ветка релиза не должна отличаться от master, иначе в дальнейшем у нас будут большие проблемы. Это должно быть место, где ошибки исправляются изолированным способом. Смысл этой ветки заключается в стабилизации тестируемого продукта. Если бы мы тестировали в master, то никогда бы не закончили, потому что он всегда находится в очень активной разработке. Мы бы бесконечно занимались тестированием, потому что скорость поиска ошибок никогда не уменьшилась бы. Программисты добавляли бы каждый день новые, а Джиоти обнаруживала бы их. Они исправляют — она открывает новые. Это нескончаемый процесс. Чтобы достичь более или менее стабильной точки, нам нужно изолировать разработчиков и тестировщиков.

— Что, если этот момент никогда не наступит? — спрашивает Джиоти. — Или просто будет занимать слишком много времени?

— Это возможно. Мы можем создать ветку релиза, начать тестировать и «ломать» ее, сообщать об ошибках и ждать, пока показатель достигнет желаемого значения.

**Попытка выпуска может быть прекращена, если при тестировании обнаружено слишком много ошибок.**

[1] *Li Y. et al.* Semantic Slicing of Software Version Histories. IEEE Transactions on Software Engineering, Volume 44, Number 2, 2018: «Иногда разработчикам необходимо перенести определенную функциональность, например функцию или исправление ошибки, из одной ветки в другую. Бэкпортинг — пример такой миграции, когда изменения, внесенные в новую версию программного обеспечения, переносятся в более раннюю, чтобы предоставить обновленную функциональность всем пользователям».

Но этого не происходит, потому что... Я действительно не знаю из-за чего... Хорошо, например, одна из разветвленных нами функциональностей не была полностью реализована в master — мы сделали ветку с очень нестабильным и неполным фрагментом кода. Мы пытаемся его стабилизировать, но не получается. А вот в master уже добавлено немало изменений, и код там куда стабильнее. Что делать? Мы просто отбрасываем ветку релиза, удаляем ее и начинаем с нуля. Можно назвать это «прекращенной попыткой выпуска», и это совершенно нормальная ситуация. Конечно, мы должны стараться такого избегать, но всякое может случиться. Мы не потеряем ничего, кроме времени.

— Но время важно, — улыбается Адриан.

— Конечно, но качество важнее, так ведь? Хотя, конечно, зависит от проекта. В нашем случае, когда клиентская база достаточно велика и репутация на рынке играет огромную роль, качество превыше всего. Разве не так?

— Ты прав, — подтверждает Джиоти. — Что дальше?

— Когда ветка релиза будет готова, мы развернем ее в производство. Вот и все. Здесь нет чудес, всего три шага в конвейере поставки: объединение ветви Чунга в master, стабилизация ветки релиза и ее развертывание в производство. Первый шаг — отбраковка всего плохого кода, второй шаг — попытка «сломать» продукт всеми возможными способами, третий — поставка результата работы клиентам. Вот почему он называется конвейером поставки. Потому что главная цель — поставить наш продукт клиентам.

— Но ты говорил, что на пути будет немало препятствий. Сейчас осталось всего три шага. Где препятствия? — спрашивает Адриан.

— Они будут. Во-первых, мы должны сделать все возможное, чтобы отказаться от ветки Чунга. Мы должны поставить на своем пути инструменты, валидаторы, контроллеры, тестеры, автоматизированные тесты и ручные проверки. Должно быть трудно добавить что-либо, если оно не соответствует высочайшему качеству. Во-вторых, нужно сделать тестирование очень сложным и мощным, попытаться «сломать» продукт всеми возможными способами, проверяя производительность, нагрузку, удобство ис-

пользования и проводя A/B-тестирование[1]. Ваш отдел должен быть самым сильным в компании, — я смотрю на Джиоти. — У вас должны работать программисты, инженеры, дизайнеры и архитекторы, способные создавать автоматизированные препятствия и использовать имеющиеся инструменты. Поскольку вы будете заинтересованы в том, чтобы находить как можно больше ошибок, в ваших интересах не снижать скорость их обнаружения в каждом следующем релизе.

— Но ведь так мы никогда не выпустим продукт! — волнуется Джиоти.

— Выпустим, потому что руководству на самом-то деле и не нужно идеальное качество. Какое-то качество в какой-то момент будет считаться приемлемым, тогда и выпустим.

— Мы не должны раскрывать эту стратегию нашим клиентам, — улыбается она.

— Ты права, им совсем не обязательно знать, что продукт, который они видят, все еще содержит ряд необнаруженных ошибок и часть уже найденных, но еще не исправленных. Но это реальность: либо команда разработчиков управляет ею, либо боится ее[2]. Я голосую за первый вариант. А ты?

---

[1] *Black R. et al.* Foundations of Software Testing ISTQB Certification. Cengage Learning EMEA, 3rd Edition, 2012.

[2] *Boehm B. W. et al.* Software Risk Management. IEEE Software, Volume 14, Number 3, 1997: «Руководители программных проектов охотнее признавали бы свои риски и управляли ими, если бы больше знали об организациях, которые уже решили действовать в этом направлении. Хотя есть свидетельства того, что управление рисками программного обеспечения начинает затрагивать многие компании и правительственные учреждения, опубликованных сведений об этом очень мало. Существует причина, по которой так происходит и, вероятно, будет происходить в дальнейшем: управление программными рисками имеет смысл, но разговоры об этом могут привести к юридическим обязательствам. В случае сбоя программного продукта наличие формального плана по риску, в котором признается возможность такого сбоя, может усложнить и даже поставить под угрозу правовую позицию производителя. Есть основания подозревать, что большинство практикующих специалистов по управлению рисками придерживаются позиции "не спрашивай, не говори" по отношению к публикации историй своих успехов».

— Я не знаю, все это для меня ново и как-то даже немного необычно.

— Давайте тогда пообедаем.

И мы все встаем.

**Качество — продукт конфликта между программистами и тестировщиками.**

Конфликт между тестировщиками и программистами, даже если на бумаге он логичен, в реальной жизни происходит очень редко. Главным образом потому, что они работают рядом друг с другом в одном офисе и над одной частью проекта. Они склонны избегать конфронтации как на межличностном, так и на профессиональном уровне. Тестировщики ощущают дискомфорт, когда находят ошибки, а программисты — когда их делают. В идеале этого не должно происходить. Вместо этого тестировщиков нужно поощрять, когда они находят ошибки, а программистов — когда исправляют их. Ошибки — своего рода топливо программного проекта. Поскольку любое топливо токсично и опасно, ошибки тоже пугают. Может ли команда разработчиков достичь такого высокого уровня мастерства, когда ошибки будут приветствоваться, а их нахождение и исправление будет восприниматься как добрый знак?

# Эпилог

В следующей книге Джиоти разработает новую политику тестирования и представит ее Тони на утверждение; Деннис добавит статический анализ в автоматизированный цикл сборки. Они попытаются начать считать ошибки, поступающие от ручных тестировщиков; в качестве эксперимента будут введены показатели эффективности для программистов. Continuous Integration будет обсуждаться подробно, будут проанализированы ее подводные камни. Кроме того, будет введен независимый технический аудит, и Тони попытается найти для этого удаленного консультанта. Тони также наймет команду аутсорсинга для увеличения скорости разработки.

США, Украина, Россия,
2017–2018

*Егор Бугаенко*

# Наш код. Ремесло, профессия, искусство

*Перевел с английского А. Тумаркин*


| | |
|---|---|
| Заведующая редакцией | *Ю. Сергиенко* |
| Руководитель проекта | *О. Сивченко* |
| Ведущий редактор | *Н. Гринчик* |
| Литературный редактор | *О. Андросик* |
| Корректор | *Е. Павлович* |
| Верстка | *Г. Блинов* |

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: 03.2019. Наименование: книжная продукция. Срок годности: не ограничен.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12 —
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 20.03.19. Формат 70×100/16. Бумага офсетная. Усл. п. л. 18,060. Тираж 1500. Заказ 2397.

Отпечатано в ОАО «Можайский полиграфический комбинат».
143200, г. Можайск, ул. Мира, 93.
www.oaompk.ru, www.оаомпк.рф тел.: (495) 745-84-28, (49638) 20-685

Егор Бугаенко — основатель и CEO компании Zerocracy, разрабатывающей AI-роботов; ООП-фундаменталист, автор серии книг об объектно-ориентированном программировании; провокационный блогер на yegor256.com; пишущий Java-программист. Известен также созданием Cactoos, Takes Framework, JCabi и Rultor. Егор — филантроп, ежегодно жертвующий несколько тысяч долларов номинантам его собственной премии за самый качественный программный проект.

«Быть программистом может быть интересно и весело, но быть разработчиком программного обеспечения — это сущий ад. Компьютеры логичны, люди — нет. Увы, в современной индустрии программного обеспечения не платят за программирование. Платят за разработку программного обеспечения, а это подразумевает выполнение задач в команде — вместе с другими людьми. Команды состоят из своенравных людей, а не из классов и методов Java. Успех программного проекта зависит от умных инженеров, которые зачастую ленивы, невежественны, эгоистичны, раздражительны и попросту несчастны. Успех зависит от людей, которые нередко не умеют общаться, делиться знаниями, руководить и подчиняться, а также следовать указаниям. Он зависит от нашей способности формировать команды и участвовать в их деятельности. А еще от наших социальных навыков — порой в гораздо большей степени, нежели от навыков технических. Драма? Согласен. Эта драма касается каждого из наших собратьев по профессии, поэтому, если стремитесь выжить в такой профессии, читайте эту книгу».

ISBN: 978-5-4461-1144-2

ПИТЕР®

Заказ книг:
тел.: (812) 703-73-74
books@piter.com

WWW.PITER.COM
каталог книг и интернет-магазин

instagram.com/piterbooks
youtube.com/ThePiterBooks
vk.com/piterbooks
facebook.com/piterbooks