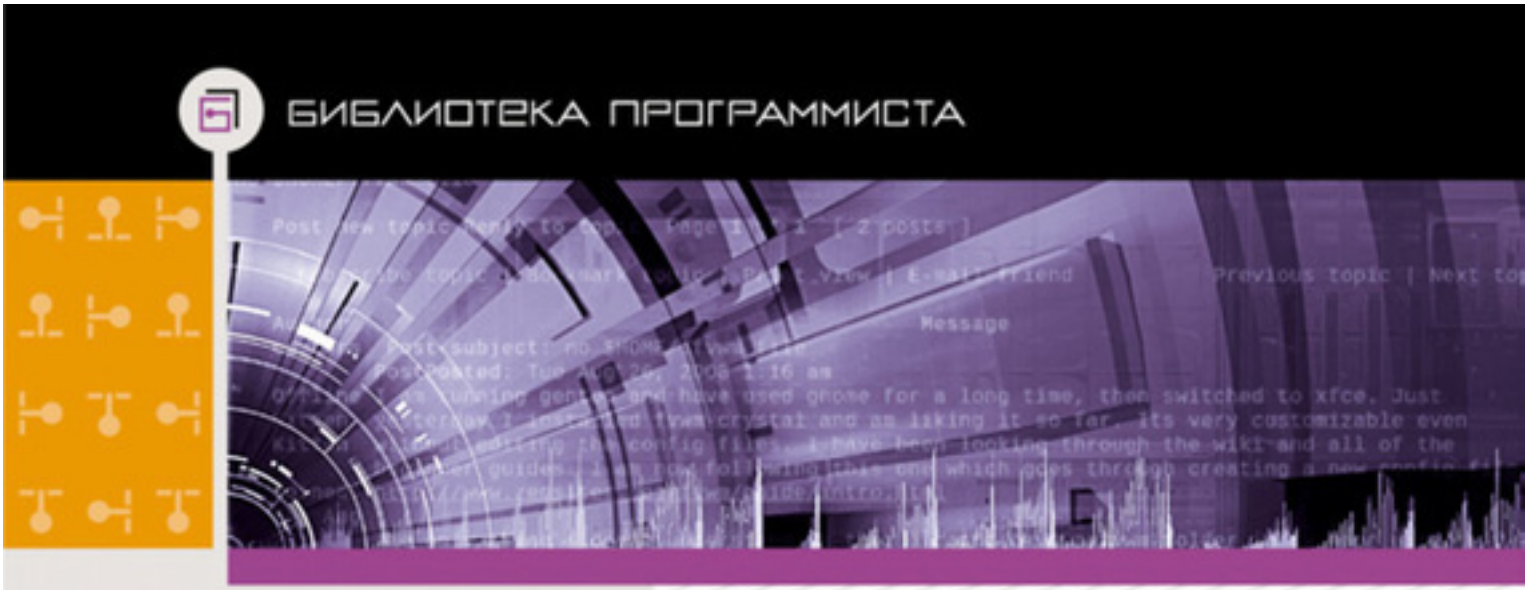


Роберт Мартин

Идеальный программист

Как стать профессионалом разработки ПО

Библиотека программиста (Питер)

Роберт Мартин

Идеальный программист. Как стать профессионалом разработки ПО

«Питер»
2011

УДК 004.415
ББК 32.973.2-018-02

Мартин Р. С.

Идеальный программист. Как стать профессионалом разработки ПО / Р. С. Мартин — «Питер», 2011 — (Библиотека программиста (Питер))

ISBN 978-5-459-01044-2

Всех программистов, которые добиваются успеха в мире разработки ПО, отличает один общий признак: они больше всего заботятся о качестве создаваемого программного обеспечения. Это – основа для них. Потому что они являются профессионалами своего дела.В этой книге легендарный эксперт Роберт Мартин (более известный в сообществе как «Дядюшка Боб»), автор бестселлера «Чистый код», рассказывает о том, что значит «быть профессиональным программистом», описывая методы, инструменты и подходы для разработки «идеального ПО». Книга насыщена практическими советами в отношении всех аспектов программирования: от оценки проекта и написания кода до рефакторинга и тестирования. Эта книга – больше, чем описание методов, она о профессиональном подходе к процессу разработки.

УДК 004.415
ББК 32.973.2-018-02

ISBN 978-5-459-01044-2

© Мартин Р. С., 2011
© Питер, 2011

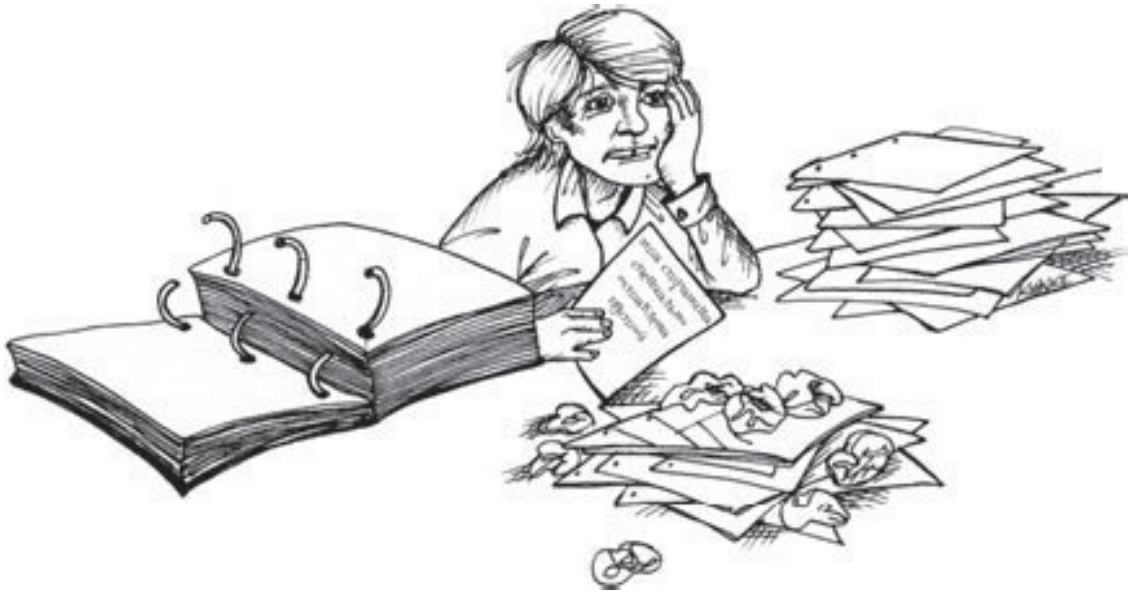
Содержание

Роберт Мартин
Идеальный программист. Как стать профессионалом разработки ПО

Robert C. Martin
The Clean Coder:
A Code of Conduct for Professional Programmers

© Prentice Hall, Inc., 2011
© Перевод на русский язык, издание на русском языке ООО Издательство «Питер», 2012

Обязательное вступление (Не пропускайте, оно вам понадобится!)



Почему вы выбрали эту книгу? Наверное, потому что вы – программист, и вас интересует понятие профессионализма. И правильно! Профессионализм – то, чего так отчаянно не хватает в нашей профессии.

Я тоже программист. Я занимался программированием 42[1] года и за это время повидал многое. Меня увольняли. Меня превозносили до небес. Я побывал руководителем группы, начальником, рядовым работником и даже исполнительным директором. Я работал с выдающимися программистами, и я работал со слизняками.[2] Я занимался разработкой как самых передовых встроенных программных/аппаратных систем, так и корпоративных систем начисления зарплаты. Я программировал на COBOL, FORTRAN, BAL, PDP-8, PDP-11, C, C++, Java, Ruby, Smalltalk и на многих других языках. Я работал с бездарными халявщиками, и я работал с высококвалифицированными профессионалами. Именно последней классификации посвящена эта книга.

На ее страницах я попытаюсь определить, что же это такое – «быть профессиональным программистом». Я опишу те атрибуты и признаки, которые, на мой взгляд, присущи настоящим профессионалам.

Откуда я знаю, что это за атрибуты и признаки? Потому что я познал все это на собственном горьком опыте. Видите ли, когда я поступил на свое первое место работы на должность программиста, никому бы не пришло в голову назвать меня профессионалом.

Это было в 1969 году. Мне тогда было 17 лет. Мой отец убедил местную фирму под названием ASC нанять меня программистом на неполный рабочий день. (Да, мой отец это умеет. Однажды он на моих глазах встал на пути разгоняющейся машины, поднял руку и приказал: «Стоять!» Машина остановилась. Моему папе вообще трудно отказать.) Меня приняли на работу, посадили в комнату, где хранились все руководства к компьютерам IBM, и заста-

[1] Без паники!

[2] Технический термин неизвестного происхождения.

вили записывать в них описания обновлений за несколько лет. Именно тогда я впервые увидел фразу: «Страница намеренно оставлена пустой».

Через пару дней обновления руководств мой начальник предложил мне написать простую программу на Easycoder.[3] Его просьба вызвала у меня бурный энтузиазм, ведь до этого я еще не написал ни одной программы для настоящего компьютера. Впрочем, я бегло просмотрел несколько книг по Autocoder и примерно представлял, с чего следует начать.

Моя программа должна была прочитать записи с магнитной ленты и изменить идентификаторы этих записей. Значения новых идентификаторов начинались с 1 и увеличивались на 1 для каждой последующей записи. Записи с обновленными идентификаторами должны были записываться на новую ленту.

Начальник показал мне полку, на которой лежало множество стопок красных и синих перфокарт. Представьте, что вы купили 50 колод игральных карт – 25 красных и 25 синих, а потом положили эти колоды друг на друга. Так выглядели эти стопки. В них чередовались карты красного и синего цвета; каждая «колода», состоявшая примерно из 200 карт, содержала исходный код библиотеки подпрограмм. Программисты просто снимали верхнюю «колоду» со стопки (убедившись, что они взяли только красные или только синие карты) и клали ее в конец своей стопки перфокарт.

Моя программа была написана на программных формулярах – больших прямоугольных листах бумаги, разделенных на 25 строк и 80 столбцов. Каждая строка соответствовала одной карте. Программа записывалась на формуляре прописными буквами. В последних 6 столбцах каждой строки записывался ее номер. Номера обычно увеличивались с приращением 10, чтобы позднее в стопку можно было вставить новые карты.

Формуляры передавались операторам подготовки данных. В компании работало несколько десятков женщин, которые брали формуляры из большого ящика и «набивали» их на клавишных перфораторах. Эти машины были очень похожи на пишущие машинки, но они не печатали вводимые знаки на бумаге, а кодировали их, пробивая отверстия в перфокартах.

На следующий день операторы вернули мою программу по внутренней почте. Маленькая стопка перфокарт была завернута в формуляры и перетянута резинкой. Я поискал на перфокартах ошибки набора. Вроде все нормально. Я положил библиотечную колоду в конец своей стопки программ и отнес ее наверх операторам.

Компьютеры были установлены в машинном зале за закрытыми дверями, в зале с регулируемым микроклиматом и фальшполом (для прокладки кабелей). Я постучал в дверь, суровый оператор забрал у меня колоду и положил ее в другой ящик. Моя программа будет запущена, когда до нее дойдет очередь.

На следующий день я получил свою колоду обратно. Она была завернута в листинг и перетянута резинкой. (Да, в те дни мы использовали *очень много* резинок!)

Я открыл листинг и увидел, что программа не прошла компиляцию. Сообщения об ошибках в листинге оказались слишком сложными для моего понимания, поэтому я отнес их своему начальнику. Он просмотрел листинг, что-то пробормотал про себя, сделал несколько пометок, взял колоду и приказал следовать за ним. Он прошел в операторскую, сел за свободный перфоратор, исправил все карты с ошибками и добавил еще пару карт. Он на скорую руку объяснил суть происходящего, но все промелькнуло в одно мгновение.

Он отнес новую колоду в машинный зал, постучал в дверь, сказал какие-то «волшебные слова» оператору, а затем прошел в компьютерный зал. Оператор установил магнитные ленты на накопители и загрузил колоду, пока мы наблюдали. Завертелись ленты, затарахтел принтер – и на этом все кончилось. Программа заработала.

---

[3] Ассемблер для компьютера Honeywell H200, аналог Autocoder для компьютера IBM 1401.

На следующий день начальник поблагодарил меня за помощь, и моя работа на этом завершилась. Очевидно, фирма ASC посчитала, что ей некогда нянчиться с 17-летними новичками.

Впрочем, моя связь с ASC на этом не завершилась. Через несколько месяцев я получил постоянную работу в вечернюю смену в ASC на обслуживании принтеров. Эти принтеры печатали всякую ерунду с образов, хранившихся на ленте. Я должен был своевременно заправлять принтеры бумагой, ставить ленты с образами, извлекать замятую бумагу и вообще следить за тем, чтобы машины нормально работали.

Все это происходило в 1970 году. Я не мог себе позволить учебу в колледже, да она меня, признаться, не особенно привлекала. Война во Вьетнаме еще не закончилась, и в студенческих городках было неспокойно. Я продолжал штудировать книги по COBOL, Fortran, PL/1, PDP-8 и ассемблеру для IBM 360. Я намеревался обойтись без учебы и как можно быстрее заняться реальным программированием.

Через год я достиг этой цели – меня повысили до штатного программиста в ASC. Я с двумя друзьями Ричардом и Тимом, которым тоже было по 19 лет, трудились вместе с тремя другими программистами над бухгалтерской системой реального времени для фирмы, занимающейся грузовыми перевозками. Мы работали на Varian 620i – простых мини-компьютерах, по архитектуре сходных с PDP-8, не считая того, что у них были 16-разрядные слова и два регистра. Программирование велось на ассемблере.

Мы написали каждую строку кода в этой системе. Да, без преувеличения каждую. Мы написали операционную систему, обработчики прерываний, драйверы ввода/вывода, файловую систему для дисков, систему подгрузки оверлеев и даже компоновщик с динамической переадресацией – не говоря уже о коде приложения. Мы написали все это за 8 месяцев, работая по 70–80 часов в неделю для соблюдения немыслимо жестких сроков. Тогда я получал $7200 в год.

Система была закончена в срок. А потом мы уволились.

Все произошло внезапно, и расставание не было дружеским. Дело в том, что после всей работы и успешной сдачи системы компания дала нам прибавку всего в 2 %. Мы почувствовали себя обманутыми. Некоторые из нас нашли работу в другом месте и попросту подали заявление.

К сожалению, я избрал другой, далеко не лучший путь. Мы с приятелем вломились в кабинет директора и уволились вместе с изрядным скандалом. Это доставило нам эмоциональное удовлетворение – примерно на день.

На следующий день я осознал, что у меня нет работы. Мне было 19 лет, я был безработным без диплома. Я прошел собеседования на нескольких вакансиях из области программирования, но они прошли неудачно. Следующие четыре месяца я проработал в мастерской по ремонту газонокосилок, принадлежащей моему сводному брату. К сожалению, ремонтника из меня не вышло, и в конце концов мне пришлось уйти. Я впал в депрессию.

Я засиживался до трех часов ночи, поедая пиццу и смотря старые фильмы ужасов по черно-белому телевизору моих родителей. Постепенно кошмары стали просачиваться с экрана в мою жизнь. Я валялся в постели до часа дня, потому что не хотел видеть очередной унылый день. Я поступил на курсы математического анализа в региональном колледже и провалил экзамен. Моя жизнь летела под откос.

Моя мать поговорила со мной и объяснила, что так жить нельзя и что я был идиотом, когда уволился, не найдя себе новую работу – да еще со скандалом и на пару с приятелем. Она сказала, что увольняться, не имея новой работы, вообще нельзя, и делать это следует спокойно, трезво и в одиночку. Она сказала, что мне следует позвонить старому начальнику и попроситься на старое место – по ее выражению, «проглотить обиду».

Девятнадцатилетние парни не склонны признавать свои ошибки, и я не был исключением. Тем не менее обстоятельства взяли верх над гордостью. В конечном итоге я позвонил

своему начальнику. И ведь сработало! Он охотно принял меня на $6800 в год, а я охотно принял его предложение.

Следующие полтора года я работал на старом месте, обращая внимание на каждую мелочь и стараясь стать как можно более ценным работником. Моей наградой стали повышения и прибавки. Все шло хорошо. Когда я ушел из этой компании, мы остались в хороших отношениях, а мне уже предложили лучшую работу.

Наверное, вы подумали, что я усвоил полученный урок и стал профессионалом? Ничего подобного. Это был лишь первый из многих уроков, которые мне еще предстояло усвоить. В дальнейшем меня уволили с одной работы за сорванный по беспечности график и чуть не уволили с другой за случайное разглашение конфиденциальной информации. Я брался за рискованные проекты и заваливал их, не обращаясь за помощью, которая, как я знал, была мне необходима. Я рьяно защищал свои технические решения, даже если они противоречили потребностям заказчиков. Я принял на работу совершенно неквалифицированного человека, который стал тяжким бременем для моего нанимателя. И что хуже всего, из-за моих организационных ошибок уволили двух других людей.

Так что относитесь к этой книге как к каталогу моих заблуждений, исповеди в моих прегрешениях и сборнику советов, которые помогут вам избежать моих ошибок.

От издательства

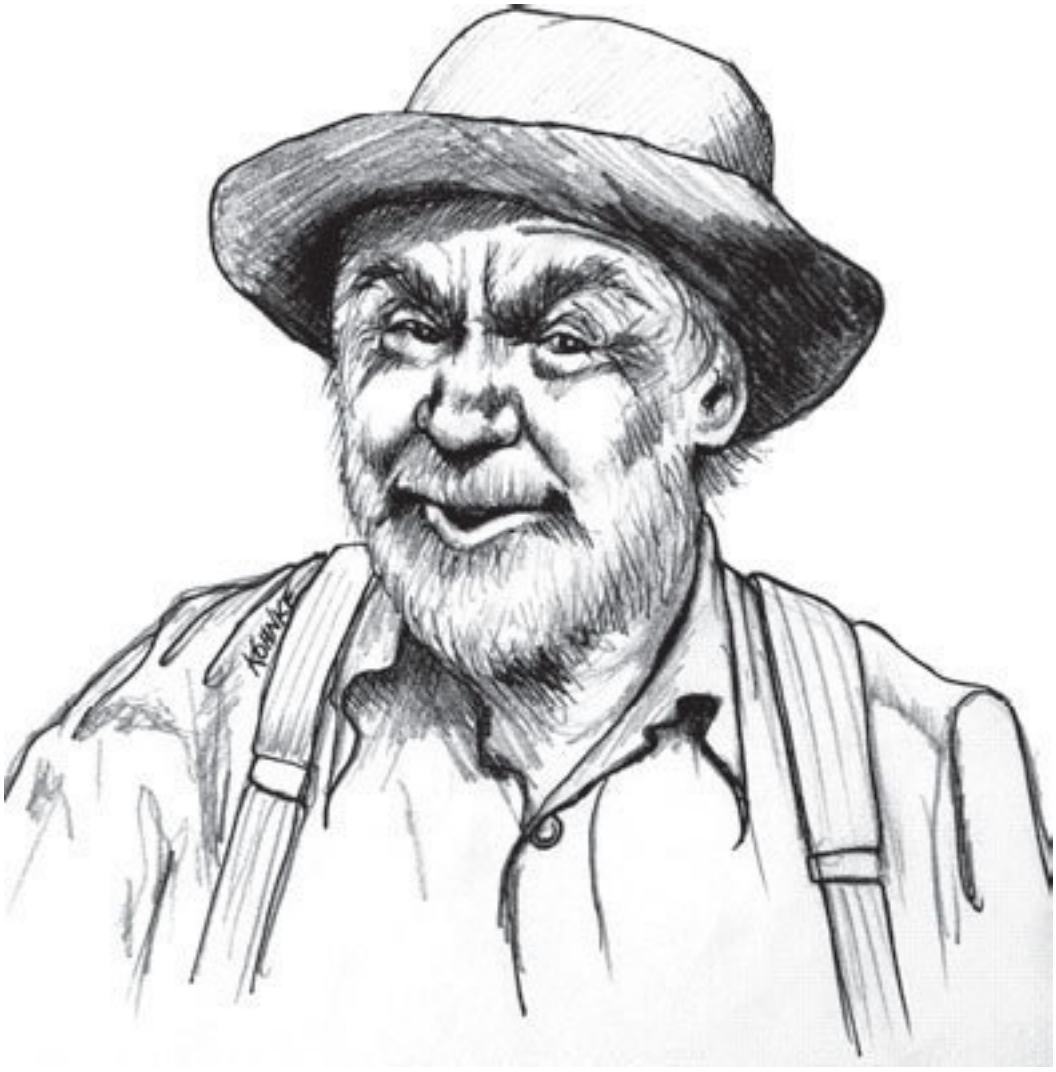
Ваши замечания, предложения и вопросы отправляйте по адресу электронной почты comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

Подробную информацию о наших книгах вы найдете на веб-сайте издательства http://www.piter.com.

1
Профессионализм

Смейся, Кертин, старина. Над нами сыграли отличную шутку – Господь Бог, природа или судьба, как тебе больше нравится. Но кто бы это ни был, у него наверняка есть чувство юмора! Ха!
Ховард, «Сокровище Сьерра-Мадре»



Итак, вы хотите стать профессиональным разработчиком? Ходить с гордо поднятой головой и объявить всему миру: «Я *профессионал*!» Хотите, чтобы люди смотрели на вас с уважением, а матери указывали на вас и говорили своим детям, что они должны вырасти такими же. Вы хотите всего этого, верно?

Оборотная сторона профессионализма

Термин «профессионализм» имеет много смысловых оттенков. Конечно, профессионализм – это своего рода почетный знак и повод для гордости, но также он является признаком ответственности. Понятно, что эти стороны профессионализма неразрывно связаны между собой: нельзя гордиться тем, за что вы не несете никакой ответственности.

Быть непрофессионалом намного проще. Непрофессионалы не несут ответственности за выполняемую работу – они оставляют ответственность своим работодателям. Если непрофессионал совершает ошибку, то мусор за ним прибирает работодатель. Но если ошибка совершается профессионалом, то устранять последствия приходится *ему самому*.

А если в ваш модуль закрадется ошибка, которая обойдется вашей компании в $10 000? Непрофессионал пожмет плечами, скажет: «Всякое бывает», и продолжит писать следующий модуль. Профессионал должен выписать своей компании чек на $10 000![4]

Да, когда речь идет о ваших личных деньгах, все выглядит немного иначе, верно? Но это ощущение присутствует у профессионалов постоянно. Более того, в нем заключается сущность профессионализма. Потому что профессионализм – это ответственное отношение к делу.

[4] Если, конечно, он правильно понимает профессиональную ответственность.

Ответственность

Вы прочитали введение, правда? Если не прочитали – вернитесь и прочитайте сейчас; оно задает контекст для всего остального материала.

Чтобы понять, почему так важно брать на себя ответственность, я на собственном опыте пережил последствия отказа от нее.

В 1979 году я работал на компанию Teradyne. Я был «ответственным инженером» за разработку программы, управляющей мини- и микрокомпьютерной системой для измерения качества телефонных линий. Центральный мини-компьютер подключался по выделенным или коммутируемым линиям на скорости 300 бод к десяткам периферийных микрокомпьютеров, управлявших измерительным оборудованием. Код был написан на ассемблере.

Нашими пользователями были администраторы по обслуживанию, работавшие в крупных телефонных компаниях. Каждый из них отвечал за 100 000 и более телефонных линий. Моя система помогала администраторам находить и исправлять неполадки и проблемы в телефонных линиях еще до того, как они будут замечены клиентами. Таким образом сокращалась частота жалоб клиентов – показатель, который измерялся комиссиями по предприятиям коммунального обслуживания и использовался для регулировки тарифов. Короче говоря, эти системы были невероятно важными.

Каждую ночь эти системы проводили «ночную проверку»: центральный мини-компьютер приказывал каждому из периферийных микрокомпьютеров протестировать все телефонные линии, находящиеся под его контролем. Каждое утро центральный компьютер получал список сбойных линий с характеристиками дефектов. По данным отчета администраторы по обслуживанию строили графики работы ремонтников, чтобы сбои исправлялись до поступления жалоб от клиентов.

Время от времени я рассылал нескольким десяткам заказчиков новую версию своей системы. «Рассылал» – самое правильное слово: я записывал программу на ленты и отправлял эти ленты своим клиентам. Клиенты загружали ленты, а затем перезапускали свои системы.

Очередная версия исправляла ряд незначительных дефектов и содержала новую функцию, которую требовали наши клиенты. Мы пообещали реализовать эту новую функцию к определенной дате. Я едва успел записать ленты в ночную смену, чтобы клиенты получили их к обещанной дате.

Через два дня мне позволил Том, наш менеджер эксплуатационного отдела. По его словам, несколько клиентов пожаловались на то, что «ночная проверка» не завершилась, и они не получили отчетов. У меня душа ушла в пятки – ведь чтобы вовремя выдать готовую версию программы, я не стал тестировать новый код. Я протестировал основную функциональность системы, но на тестирование проверки линий потребовались бы много часов, а я должен был выдать программы. Ни одна из исправленных ошибок не содержалась в коде проверки, поэтому я чувствовал себя в безопасности.

Потеря ночного отчета была серьезным делом. Она означала, что у ремонтников было меньше работы, а позднее им придется отрабатывать упущенное. Также некоторые клиенты могли заметить дефект и пожаловаться. Потери данных за целую ночь было достаточно, чтобы менеджер по обслуживанию позвонил Тому и устроил ему разнос.

Я включил тестовую систему, загрузил новую программу и запустил проверку. Программа проработала несколько часов, а затем аварийно завершилась. Код не работал. Если бы я протестировал его до поставки, то данные не были бы потеряны, а менеджеры по обслуживанию не терзали бы Тома.

Я позвонил Тому и сообщил, что мне удалось воспроизвести проблему. Оказалось, что многие другие клиенты уже обращались к нему с той же проблемой. Затем он спросил, когда

я смогу исправить ошибку. Я ответил, что пока не знаю, но работаю над ней, а пока клиенты могут вернуться к старой версии программы. Том рассердился – возврат стал бы двойным ударом для клиентов: они теряют данные за целую ночь и не могут использовать обещанную функцию.

Ошибку было трудно найти, а тестирование занимало несколько часов. Первое исправление не сработало. Второе – тоже. Мне понадобилось несколько попыток (а следовательно, дней), чтобы разобраться в происходящем. Все это время Том звонил мне через каждые несколько часов и спрашивал, когда будет исправлена ошибка. Он также передавал мне все, что ему говорили клиенты и как неудобно было предлагать им поставить старые ленты.

В конце концов я нашел дефект, отправил новые ленты, и все вошло в норму. Том (который не был моим начальником) остыл, и весь эпизод остался в прошлом. Но когда все было кончено, мой начальник пришел ко мне и сказал: «Это не должно повториться». Я согласился.

Поразмыслив, я понял, что отправка программы без тестирования кода проверки была безответственным поступком. Я пренебрег тестированием для того, чтобы сказать, что программа была отправлена вовремя. При этом я думал только о своей репутации, а не о клиенте и не о работодателе. А нужно было поступить ответственно: сообщить Тому, что тестирование не завершено и что я не готов сдать программу в назначенный срок. Было бы неприятно, Том расстроился бы, но клиенты не потеряли бы свои данные, и обошлось бы без звонков рассерженных клиентов.

Первое правило: не навреди

Итак, какие же принципы присущи ответственному поведению? Руководствоваться клятвой Гиппократа немного нескромно, но разве можно найти лучший источник? И в конце концов, это только логично – одаренный профессионал должен в первую очередь думать о том, чтобы его способности применялись только в добрых целях?

Какой вред может причинить разработчик? С чисто программной точки зрения он может навредить функциональности и структуре продукта. Давайте разберемся, как избежать именно такого ущерба.

Не навреди функциональности

Естественно, мы хотим, чтобы наши программы работали. Многие из нас стали программистами после того, как им удалось заставить работать свою первую программу, и это чувство хочется испытать снова. Но в работоспособности наших программ заинтересованы не только мы. Наши клиенты и работодатели хотят того же. Не забывайте – они платят нам за создание программ, которые делают именно то, что им нужно.

Функциональность программ страдает от ошибок. Следовательно, одним из признаков профессионализма должно быть написание программ с минимальным количеством ошибок.

«Но постойте! Ведь это нереально. Программный код слишком сложен, чтобы его можно было написать без ошибок».

Конечно, вы правы. Программный код слишком сложен, и ошибки будут всегда. К сожалению, это не избавляет вас от ответственности. Человеческое тело слишком сложно, чтобы изучить его от начала и до конца, но врачи все равно клянутся не причинять вреда. И если уж *они* не пытаются уйти от ответственности, то с какой стати это может быть позволено нам?

«Хотите сказать, что мы должны писать совершенный код?»

Нет, я хочу сказать, что вы должны отвечать за свое несовершенство. Тот факт, что в вашем коде заведомо будут присутствовать ошибки, не означает, что вы не несете за них ответственность. Написать идеальную программу практически невозможно, но за все недочеты несете ответственность именно вы, и никто другой.

Это верный признак настоящего профессионала – умение отвечать за свои ошибки, появление которых практически неизбежно. Итак, мой начинающий профессионал, прежде всего научитесь извиняться. Извинения необходимы, но недостаточны. Нельзя просто совершать одни и те же ошибки снова и снова. По мере вашего профессионального становления частота ошибок в вашем коде должна асимптотически стремиться к нулю. Она никогда не достигнет нуля, но вы ответственны за то, чтобы она была как можно ближе к нулю.

Контроль качества не должен ничего обнаружить

Когда вы передаете окончательную версию продукта в службу контроля качества, вы должны рассчитывать на то, что контроль не выявит никаких проблем. Было бы в высшей степени непрофессионально передавать на контроль качества заведомо дефектный код. А какой код является заведомо дефектным? Любой, в качестве которого вы не *уверены*!

Некоторые «специалисты» используют службу контроля качества для выявления ошибок. Они рассчитывают на то, что контроль качества обнаружит ошибки и вернет их список разработчикам. Некоторые компании даже выплачивают премии службе контроля качества за выявленные ошибки. Чем больше ошибок – тем больше премия.

Дело даже не в том, что это в высшей степени дорогостоящая практика, которая наносит ущерб компании и продукту. И не в том, что такое поведение срывает сроки и подрывает доверие к организации дела в группе разработки. И даже не в том, что это простое проявление лени и безответственности. Передавать на контроль качества код, работоспособность которого вы не можете гарантировать, непрофессионально. Такое поведение нарушает правило «не навреди».

Найдет ли служба контроля качества ошибки? Возможно, так что приготовьтесь извиняться, – а потом подумайте, почему эти ошибки ускользнули от вашего внимания, и сделайте что-нибудь для того, чтобы это не повторилось.

Когда служба контроля качества (или еще хуже – *пользователь*) обнаруживает ошибку, это должно вас удивить, огорчить и настроить на то, чтобы предотвратить повторение подобных событий в будущем.

Вы должны быть уверены в том, что ваш код работает

Как узнать, работает ли ваш код? Легко. Протестируйте его. Потом протестируйте еще раз. Протестируйте слева направо, потом справа налево. А теперь еще и сверху вниз!

Возможно, вас беспокоит, что столь тщательное тестирование кода отнимает слишком много времени. В конце концов, у вас есть графики и сроки, которые нужно соблюдать. Если тратить все время на тестирование, то когда писать код? Все верно! Поэтому тестирование следует автоматизировать. Напишите модульные тесты, которые можно выполнить в любой момент, и запускайте их как можно чаще.

Какая часть кода должна тестироваться этими автоматизированными модульными тестами? Мне действительно нужно отвечать на этот вопрос? Весь код! Весь. Без исключения.

Скажете, я предлагаю 100 % тестовое покрытие кода? Ничего подобного. Я не *предлагаю*, а *требую*. Каждая написанная вами строка кода должна быть протестирована. Точка.

Может, это нереалистично? Почему? Вы пишете код, потому что ожидаете, что он будет выполняться. Если вы ожидаете, что код будет выполняться, то вы должны знать, что он работает. А *знать* это можно только в одном случае – по результатам тестирования.

Я являюсь основным автором и исполнителем проекта с открытым кодом FitNesse. На момент написания книги размер FitNesse достиг 60K строк, 26 из которых содержатся в 2000+ модульных тестах. По данным Emma, покрытие этих 2000 тестов составляет около 90 % кода. Почему не выше? Потому что Emma видит не все выполняемые строки! По моей оценке, степень покрытия намного выше. Составляет ли она 100 %? Нет, 100 % – асимптотический предел.

Но ведь некоторые части кода трудно тестировать? Да, но только потому, что этот код был так *спроектирован*. Значит, код нужно проектировать с расчетом на *простоту* тестирования. И для этого лучше всего написать тесты сначала – до того кода, который должен их пройти.

Этот принцип используется в методологии разработки через тестирование (TDD, Test Driven Development), которая будет более подробно описана в одной из следующих глав.

Автоматизированный контроль качества

Вся процедура контроля качества FitNesse заключается в выполнении модульных и приемных тестов. Если тесты проходят успешно, я выдаю продукт. При этом процедура контроля качества занимает около трех минут, и я могу выполнить ее в любой момент.

Конечно, из-за ошибки в FitNesse никто не умрет и никто не потеряет миллионы долларов. С другой стороны, у FitNesse много тысяч пользователей, а список дефектов очень невелик.

Безусловно, некоторые системы настолько критичны, что короткого автоматизированного теста недостаточно для определения их готовности к развертыванию. С другой стороны, вам как разработчику необходим относительно быстрый и надежный механизм проверки того, что написанный код работает и не мешает работе остальных частей системы. Итак, автоматизированные тесты по меньшей мере должны сообщить вам, что система с большой вероятностью пройдет контроль качества.

Не навреди структуре

Настоящий профессионал знает, что добавление функциональности в ущерб структуре – последнее дело. Структура кода обеспечивает его гибкость. Нарушая структуру, вы разрушаете будущее кода.

Все программные проекты базируются на фундаментальном предположении о простоте изменений. Если вы нарушаете это предположение, создавая негибкие структуры, то вы тем самым подрываете экономическую модель, заложенную в основу всей отрасли.

Внесение изменений не должно приводить к непомерным затратам.

К сожалению, слишком многие проекты вязнут в болоте плохой структуры. Задачи, которые когда-то решались за считанные дни, начинают занимать недели, а потом и месяцы. Руководство в отчаянных попытках наверстать потерянный темп нанимает дополнительных разработчиков для ускорения работы. Но эти разработчики только ухудшают ситуацию, углубляя структурные повреждения и создавая новые препятствия.

О принципах и паттернах проектирования, способствующих созданию гибких, удобных в сопровождении структур, написано много книг.[5]

Профессиональные разработчики держат эти правила в памяти и стараются строить свои программные архитектуры по ним. Однако существует один нюанс, о котором часто забывают: *если вы хотите, чтобы ваш код был гибким, его необходимо проверять на гибкость*!

Как убедиться в том, что в ваш продукт легко вносятся изменения? Только одним способом – попытаться внести в него изменения! И если сделать это оказывается сложнее, чем предполагалось, то вы перерабатываете структуру кода, чтобы следующие изменения вносились проще.

Когда следует вносить такие изменения? *Всегда*! Каждый раз при работе с модулем следует понемногу совершенствовать его структуру. Каждое чтение кода должно приводить к доработке структуры.

Эта идеология иногда называется *безжалостным рефакторингом*. Я называю этот принцип «правилом бойскаута»: *всегда оставляйте модуль чище, чем до вашего прихода*. Всегда совершайте добрые дела в коде, когда вам представится такая возможность.

Такой подход полностью противоречит отношению некоторых людей к программному коду. Они считают, что серии частых изменений в рабочем коде опасны. Нет! Опасно оставлять код в статическом, неизменном состоянии. Если не проверять код на гибкость, то когда потребуется внести изменения, он может оказаться излишне жестким.

Почему многие разработчики боятся вносить частые изменения в свой код? Да потому что они боятся его «сломать»! А почему они этого боятся? Потому что у них нет тестов.

Мы снова возвращаемся к тестам. Если у вас имеется автоматизированный тестовый пакет, покрывающий почти 100 % кода, и если этот пакет можно быстро выполнить в любой момент времени, то вы попросту не будете бояться изменять код. А как доказать, что вы не боитесь изменять код? Изменяйте его почаще.

---

[5] Robert C. Martin, *Principles, Patterns, and Practices of Agile Software Development*, Upper Saddle River, NJ: Prentice Hall, 2002.

Профессиональные разработчики настолько уверены в своем коде и тестах, что они с легкостью вносят случайные, спонтанные изменения. Они могут ни с того ни с сего переименовать класс. Заметив слишком длинный метод во время чтения модуля, они по ходу дела разбивают его на несколько меньших методов. Они преобразуют команду switch в полиморфную конструкцию или сворачивают иерархию наследования в линейную цепочку. Короче говоря, они относятся к коду так же, как скульптор относится к глине – они постоянно разминают его и придают новую форму.

Трудовая этика

За свою карьеру отвечаете вы сами. Ваш работодатель не обязан заботиться о вашей востребованности на рынке труда. Он не обязан обучать вас, отправлять вас на конференции или покупать книги. Всем этим должны заниматься *вы сами*. Горе тому разработчику, который доверит свою карьеру своему работодателю!

Некоторые работодатели согласны покупать вам книги, отправлять вас на семинары и конференции. Прекрасно, они оказывают вам услугу. Но никогда не думайте, что они обязаны это делать! Если ваш работодатель не делает этого за вас, подумайте, как сделать это своими силами.

Ваш работодатель также не обязан выделять вам время для учебы. Некоторые выделяют время на повышение квалификации – или даже требуют, чтобы вы это делали. Но и в этом случае они оказывают вам услугу, и вы должны быть им благодарны. Не рассчитывайте на это как на нечто само собой разумеющееся.

Вы обязаны своему работодателю некоторым количеством времени и усилий. Для примера возьмем стандартную для США 40-часовую рабочую неделю. Эти 40 часов должны быть проведены за решением проблем *вашего работодателя*, а не *ваших* личных проблем.

Запланируйте 60 рабочих часов в неделю. Первые 40 вы работаете на своего работодателя, а остальные 20 на себя. В эти 20 часов вы читаете книги, практикуетесь, учитесь и иным образом развиваете свою карьеру.

Наверняка вы подумали: «А как же моя семья? Моя личная жизнь? Я должен пожертвовать всем ради своего работодателя?»

Я не говорю обо всем вашем личном времени. Я говорю о 20 дополнительных часах в неделю. Если вы будете использовать обеденный перерыв для чтения и прослушивания подкастов и еще 90 минут в день на изучение нового языка – это решит все проблемы.

Давайте немного посчитаем. В неделе 168 часов. 40 достается вашему работодателю, еще 20 – вашей карьере. Остается 108. 56 тратится на сон, на все остальное остается 52. Возможно, вы не хотите брать на себя подобные обязательства. И это вполне нормально, но тогда не считайте себя профессионалом. Профессионалы не жалеют времени на совершенствование в своей профессии.

Возможно, вы считаете, что работа должна оставаться на рабочем месте и ее не следует брать домой. Согласен! В эти 20 часов вы должны работать не на своего работодателя, а на свою карьеру.

Иногда эти два направления совпадают. Иногда работа, выполняемая для работодателя, оказывается исключительно полезной для вашей карьеры. В таком случае потратить на нее некоторые из этих 20 часов будет вполне разумно. Но помните: эти 20 часов предназначены для вас. Они используются для того, чтобы повысить вашу профессиональную ценность.

Может показаться, что мой путь ведет к «перегоранию» на работе. Напротив, он помогает избежать этой печальной участи. Вероятно, вы стали разработчиком из-за своего энтузиазма к программированию, а ваше желание стать профессионалом обусловлено этим энтузиазмом. За эти 20 часов вы будете заниматься тем, что подкрепит ваш энтузиазм. Эти 20 часов должны быть *интересными*!

Знай свою область

Вы знаете, что такое диаграмма Насси—Шнейдермана? Если не знаете – почему? А чем отличаются конечные автоматы Мили и Мура? Должны знать. Сможете написать процедуру быстрой сортировки, не обращаясь к описанию алгоритма? Выполнить функциональную

декомпозицию диаграммы информационного потока? Что означает термин «бесхозные данные»? Для чего нужны «таблицы Парнаса»?

За последние 50 лет в нашей области появилось множество новых идей, дисциплин, методов, инструментов и терминов. Сколько из них вы знаете? Каждый, кто хочет стать профессионалом, обязан знать заметную часть и постоянно увеличивать размер этой части.

Почему необходимо знать все это? Разве наша область не прогрессирует так быстро, что старые идеи теряют актуальность? Первая часть вопроса вполне очевидна: безусловно, в нашей области происходит стремительный прогресс. Но интересно заметить, что этот прогресс во многих отношениях имеет периферийную природу. Действительно, нам уже не приходится по 24 часа дожидаться завершения компиляции. И действительно, мы пишем системы, размер которых измеряется гигабайтами. Правда и то, что мы работаем в глобальной сети, предоставляющей мгновенный доступ к информации. Но с другой стороны, мы пишем те же команды if и while, что и 50 лет назад. Многое изменилось. Многое осталось неизменным.

Вторая часть вопроса так же очевидно неверна. Лишь очень немногие идеи последних 50 лет потеряли актуальность. Некоторые ушли на второй план, это правда. Концепция каскадной разработки, скажем, явно перестала пользоваться популярностью. Однако это не означает, что мы не должны знать, что это за концепция, каковы ее сильные и слабые стороны.

В целом подавляющее большинство с трудом завоеванных идей последних 50 лет ничуть не утратило своей ценности. А может, эти идеи стали еще более ценными. Вспомните проклятие Сантаяны: «Не помнящие прошлого обречены на его повторение».

Далее приводится *минимальный* список тем, в которых должен разбираться каждый разработчик.

• Паттерны проектирования. Вы должны быть способны описать все 24 паттерна из книги «Банды Четырех» и иметь практическое представление о многих паттернах из книг «Pattern-Oriented Software Architecture».

• Принципы проектирования. Вы должны знать принципы SOLID и хорошо разбираться в принципах компонентного проектирования.

• Методы. Вы должны понимать суть методологий XP, Scrum, экономной[6] разработки (Lean), Kanban, каскадной разработки, структурного анализа и структурного проектирования.

• Дисциплины. Практикуйтесь в практическом применении разработки через тестирование (TDD), объектно-ориентированного проектирования, структурного программирования, непрерывной интеграции и парного программирования.

• Артефакты. Вы должны уметь работать с UML, DFD, структурными диаграммами, сетями Петри, диаграммами переходов, блок-схемами и таблицами решений.

Непрерывное обучение

Неистовый темп изменений в нашей отрасли означает, что разработчики должны постоянно изучать большой объем материала только для того, чтобы оставаться в курсе дела. Горе проектировщикам, которые перестают программировать – они быстро оказываются не у дел. Горе программистам, которые перестают изучать новые языки – им придется смотреть, как отрасль проходит мимо них. Горе разработчикам, которые не изучают новые дисциплины и методологии – их ожидает упадок на фоне процветания коллег.

Пойдете ли вы к врачу, который не знает, что сейчас происходит в медицине и не читает медицинские журналы? Обратитесь ли вы к консультанту по налогам, который не следит за налоговым законодательством и прецедентами? Так зачем работодателю нанимать разработчика, который не стремится быть в курсе дел?

---

[6] Также называемой «бережливой». – *Примеч. перев.*

Читайте книги, статьи, блоги, твиты. Посещайте конференции и собрания пользовательских групп. Участвуйте в работе исследовательских групп. Изучайте то, что лежит за пределами вашей привычной зоны. Если вы программист. NET – изучайте Java. Если вы программируете на Java – изучайте Ruby. Если вы программируете на C – изучайте Lisp. А если вам захочется серьезно поработать мозгами, изучайте Prolog и Forth!

Тренировка

Профессионалы тренируются. Настоящие профессионалы прилежно работают над тем, чтобы их навыки были постоянно отточены и готовы к применению. Недостаточно выполнять свою повседневную работу и называть ее тренировкой. Повседневная работа – это исполнение обязанностей, а не тренировка. Тренировка начинается тогда, когда вы целенаправленно применяете свои навыки за пределами своих рабочих обязанностей с единственной целью совершенствования этих навыков.

Что может означать тренировка для разработчика? На первый взгляд сама концепция выглядит абсурдно. Но давайте ненадолго задержимся и подумаем. Как музыканты совершенствуют свое мастерство? Не на концертах, а во время занятий. Как они это делают? Среди прочего, у них имеются специальные упражнения, гаммы и этюды. Музыканты повторяют их снова и снова, чтобы тренировать свои пальцы и ум и чтобы поддерживать свое мастерство на должном уровне.

Как тренируются разработчики? В этой книге целая глава посвящена разным методам тренировки, поэтому я не стану углубляться в подробности сейчас. Например, я часто применяю метод повторения простых упражнений вроде «игры в боулинг» или «разложения на простые множители». Я называю эти упражнения *ката*. Существует много разных ката, из которых можно выбрать то, что лучше подойдет вам.

Ката обычно имеет вид простой задачи по программированию – например, написать функцию, которая раскладывает целое число на простые множители. Целью выполнения ката является не поиск решения; вы уже знаете, как решается задача. Ката тренируют ваши пальцы и ваш мозг.

Я ежедневно выполняю одну-две ката, часто в процессе погружения в работу. Я пишу их на Java, Ruby, Clojure или на каком-нибудь другом языке, который я хочу поддерживать в рабочем состоянии. Я использую ката для тренировки конкретных навыков (например, приучая пальцы к использованию клавиш ускоренного доступа) или приемов рефакторинга.

Относитесь к ката как к 10-минутной разминке по утрам и 10-минутной релаксации по вечерам.

Совместная работа

Второй лучший способ чему-то научиться – совместная работа с другими людьми. Профессиональные разработчики намеренно стараются вместе программировать, вместе тренироваться, вместе проектировать и планировать. При этом они много узнают друг от друга, выполняют свою работу быстрее и с меньшим количеством ошибок.

Это не означает, что вы должны проводить 100 % своего рабочего времени за совместной работой. Одиночная работа тоже очень важна. Как бы я ни любил программировать в паре, мне абсолютно необходимо время от времени поработать одному.

Наставничество

Самый лучший способ чему-то научиться – учить других. Факты запоминаются быстрее всего тогда, когда вы должны их сообщить другим людям, за которых вы отвечаете. Таким образом, основную пользу от преподавания получает прежде всего преподаватель.

Аналогичным образом лучший способ ввести новых людей в организацию – посидеть с ними и объяснить, что и как работает. Профессионалы берут на себя персональную ответственность за обучение новичков. Они не бросают новичков, предоставляя им самим решать свои проблемы.

Знание предметной области

Каждый профессионал обязан понимать предметную область программируемых им решений. Если вы пишете бухгалтерскую систему, вы должны разбираться в бухгалтерии. Если вы пишете приложение для туристической фирмы, вы должны разбираться в туризме. Быть экспертом не обязательно, но к изучению темы необходимо относиться ответственно. Начиная проект в новой для себя области, прочитайте одну-две книги по теме. Проведите собеседование с клиентом и пользователями об основах предметной области. Поговорите с экспертами, постарайтесь понять их принципы и ценности.

Худшая разновидность непрофессионализма – просто программировать по спецификации, не понимая того, почему эта спецификация подходит для решения своей задачи. Вы должны обладать достаточными познаниями в предметной области для того, чтобы распознать и исправить возможные ошибки в спецификации.

Понимание интересов работодателя/заказчика

Проблемы вашего работодателя – это *ваши* проблемы. Вы должны понимать их и постараться найти лучшие решения. В ходе разработки системы представьте себя на месте своего работодателя и убедитесь в том, что разрабатываемые вами возможности действительно соответствуют его потребностям.

Разработчику легко представить себя на месте другого разработчика. Но когда речь идет об отношении к работодателю, многие попадают в ловушку представлений «мы и они». Профессионалы всеми силами стремятся избежать этого.

Скромность

Программирование – творческая деятельность. Во время написания кода мы творим нечто из ничего. Мы решительно наводим порядок в хаосе. Мы уверенно определяем поведение машины, которая в случае ошибки могла бы причинить невообразимый ущерб. Таким образом, программирование является актом исключительно амбициозным.

Профессионалам не свойственно ложное смирение. Профессионал знает свою работу и гордится ей. Профессионал уверен в своих способностях и сознательно идет на риск, основываясь на этой уверенности. Профессионал не может быть робким.

Однако профессионал также знает, что в отдельных случаях он потерпит неудачу, его оценки риска окажутся неверными, а его способности – недостаточными; он посмотрит в зеркало и увидит, что оттуда ему улыбается самонадеянный болван.

Итак, оказавшись мишенью для насмешки, профессионал смеется первым. Он сам никогда не высмеивает других, но принимает заслуженные насмешки и легко отмахивается от неза-

служенных. Он не издевается над коллегами, допустившими ошибку, потому что знает – следующим может быть он сам.

Профессионал понимает свою гордыню, а также то, что судьба рано или поздно заметит и найдет его слабые места. И когда ее выстрел попадет в цель, остается только следовать совету: смейтесь.

2
Как сказать «нет»

Делай. Или не делай. Не надо пытаться.
Йода

В начале 1970-х годов мы с двумя 19-летними друзьями работали над бухгалтерской системой реального времени для профсоюза грузоперевозчиков в Чикаго для компании ASC. Если у вас в памяти всплывают такие имена, как Джимми Хоффа,[7] – так и должно быть. В 1971 году профсоюз грузоперевозчиков был серьезной организацией.

Наша система должна была пойти в эксплуатацию к определенной дате. На эту дату были поставлены большие деньги. Наша группа работала по 60, 70 и 80 часов в неделю, чтобы уложиться в срок.

За неделю до даты запуска система наконец-то была собрана. В ней было множество ошибок и нерешенных проблем, и мы лихорадочно разбирались с ними по списку. Нам не хватало времени на то, чтобы есть и спать, не говоря уже о том, чтобы думать. Нашим начальником в ASC был Фрэнк, отставной полковник ВВС. Это был один из тех громогласных напористых руководителей, которые предлагают выбрать: либо ты *немедленно* прыгаешь с парашютом с высоты 3000 м либо делаешь то же самое, но уже без парашюта. Мы, 19-летние парни, его терпеть не могли.

Фрэнк сказал, что все должно быть сделано к установленной дате. И говорить больше не о чем. Когда придет срок, наша система должна быть готова. Точка. И никаких «но».

Мой босс Билл был симпатичным парнем. Он проработал с Фрэнком немало лет и хорошо понимал, что возможно, а что нет. Он сказал нам, что система должна заработать к установленной дате, что бы ни произошло.

И система заработала к установленной дате. И это кончилось оглушительным провалом.

Штаб-квартира профсоюза грузоперевозчиков связывалась с нашим компьютером, находящимся за 30 миль к северу, через дюжину полудуплексных терминалов со скоростью 300 бод. Каждый терминал зависал примерно через каждые полчаса. Мы сталкивались с этой про-

[7] Американский профсоюзный лидер, исчезнувший при загадочных обстоятельствах. – *Примеч. пер.*

блемой и прежде, но у нас не было времени на моделирование трафика, который операторы ввода данных создадут для нашей системы.

Ситуация усугублялась тем, что отрывные листки печатались на телетайпах ASR35, которые тоже подключались к нашей системе по 110-бодным телефонным линиям – и тоже зависали на середине печати.

Проблема решалась перезагрузкой. Итак, все операторы, терминалы которых еще работали, должны были завершить свою текущую работу. Когда вся работа прекращалась, они звонили нам, и мы перезагружали компьютер. Операторам зависших терминалов приходилось делать все заново. И это происходило чаще одного раза в час.

Через полдня руководитель отделения профсоюза приказал нам отключить систему и не включать ее, пока она не заработает. В итоге они потеряли половину рабочего дня, а все данные пришлось вводить заново в старой системе.

Вопли и рев Фрэнка разносились по всему зданию. Так продолжалось долго, очень долго. Затем Билл и наш системный аналитик Джалиль зашли к нам и спросили, сколько времени понадобится на обеспечение стабильной работы системы. Я сказал: «Четыре недели».

На их лицах отразился ужас, затем решимость. «Нет, – сказали они, – система должна заработать к пятнице».

Я сказал: «Послушайте, система едва заработала на прошлой неделе. Нам нужно разобраться с множеством проблем. На это понадобится четыре недели».

Но Билл и Джалиль были непреклонны: «Нет, это должно быть в пятницу. Можно хотя бы попробовать?»

Пятница была выбрана удачно – нагрузка в конце недели была намного ниже. Нам удалось найти много дефектов и исправить их до наступления понедельника. Но даже после этого вся система едва стояла, словно карточный домик. Проблемы с зависаниями по-прежнему происходили один-два раза в день. Также были и другие проблемы. Через несколько недель система была доведена до состояния, в котором жалобы прекратились, и вроде бы стала возможна нормальная жизнь.

И тогда, как я рассказывал во введении, мы все уволились. И фирма столкнулась с настоящим кризисом: ей пришлось нанимать новых программистов, чтобы разобраться с потоком жалоб от клиентов.

Кто виноват в этом фиаско? Разумеется, проблема отчасти обусловлена стилем руководства Фрэнка. Его тактика запугивания мешала ему услышать правду. Понятно, что Билл и Джалиль должны были противостоять давлению Фрэнка намного активнее. И само собой, наш руководитель группы не должен был соглашаться на требование выдать продукт к пятнице. Да и мне следовало продолжать говорить «нет» вместо того, чтобы повторять за руководителем группы.

Профессионалы говорят правду облеченным властью. У них достаточно смелости, чтобы сказать «нет» своим начальникам.

Как сказать «нет» начальнику? Ведь это ваш *начальник*! Разве вы не обязаны делать то, что говорит начальник?

Нет! Говорите «нет», если вы профессионал.

Рабам запрещается говорить «нет». Наемные работники неохотно говорят «нет». Но профессионалу *положено* говорить «нет». Более того, хорошим руководителям очень нужны люди, у которых хватает смелости сказать «нет». Только так можно действительно чего-то добиться.

Антагонистические роли

Одному из рецензентов книги эта глава очень не понравилась. Он сказал, что из-за нее он едва не отложил книгу. Ему доводилось создавать группы, в которых не было антагонистических отношений; группы работали вместе в гармонии и без конфронтации. Я рад за этого рецензента, но не уверен в том, что его группы действительно были избавлены от конфронтации настолько, насколько ему кажется. А если так – не уверен, что они были в полной мере эффективными. По собственному опыту скажу, что трудные решения лучше принимать на основании конфронтации антагонистических ролей.

Руководители – люди, которые должны исполнять свои обязанности, и большинство руководителей знает, как выполнять свою работу на должном уровне. Часть этой работы заключается в том, чтобы как можно более жестко преследовать и защищать свои цели.

Программисты – тоже люди, которые должны исполнять свои обязанности, и большинство из них знает, как выполнять свою работу на должном уровне. И если они относятся к числу настоящих профессионалов, они будут как можно более жестко преследовать и защищать *свои* цели.

Когда ваш руководитель говорит вам, что страница входа в систему должна быть готова к завтрашнему дню, он преследует и защищает одну из своих целей. Он выполняет свою работу. Если вы хорошо знаете, что сделать страницу к завтрашнему дню невозможно, то отвечая: «Хорошо, я попытаюсь», вы не выполняете свою работу. Выполнить ее в этот момент можно только одним способом: сказать: «Нет, это невозможно».

Но разве вы не должны выполнять распоряжения начальства? Нет, ваш начальник рассчитывает на то, что вы будете защищать свои цели так же жестко, как он защищает свои. Таким образом вы вдвоем приходите к *оптимальному результату*.

Оптимальным результатом является цель, общая для вас и вашего руководителя. Фокус в том, чтобы найти эту цель, а для этого обычно необходимы переговоры.

Переговоры могут быть приятными.

> **Майк**: «Пола, страница входа в систему мне нужна к завтрашнему дню».
> **Пола**: «Ого! Уже завтра? Хорошо я попробую».
> **Майк**: «Отлично, спасибо!»

Приятный разговор, никакой конфронтации. Обе стороны расстались с улыбками. Очень мило.

Но при этом обе стороны вели себя непрофессионально. Пола отлично знает, что работа над страницей займет больше одного дня, поэтому она просто врет. Возможно, она не считает свои слова враньем. Возможно, она действительно хочет попробовать и надеется, что ей каким-нибудь чудом удастся исполнить свое обещание. Но в конечном итоге она все равно врет.

С другой стороны, Майк принял ее «Я попробую» за «Да». И это просто глупо: он должен знать, что Пола попытается избежать конфронтации, поэтому он должен был проявить настойчивость и спросить: «Мне кажется или ты сомневаешься? Уверена, что ты сможешь сделать страницу к завтрашнему дню?»

Или еще одна приятная беседа.

> **Майк**: «Пола, страница входа в систему мне нужна к завтрашнему дню».
> **Пола**: «Прости, Майк, но мне понадобится больше времени».
> **Майк**: «И как ты думаешь, когда она будет готова?»
> **Пола**: «Как насчет двух недель – нормально?»
> **Майк**: (записывает что-то в ежедневнике) «Хорошо, спасибо».

Приятный, но ужасно неэффективный и совершенно непрофессиональный разговор. Обе стороны потерпели неудачу в своем поиске оптимального результата. Вместо того чтобы спрашивать, устроят Майка две недели или нет, Пола должна была высказаться утвердительно: «У меня это займет две недели, Майк».

С другой стороны, Майк принял предложенный срок без вопросов, словно его личные цели не имеют никакого значения. Интересно, не собирается ли он просто сообщить своему начальнику, что демонстрацию программы заказчику придется отложить из-за Полы? Такое пассивно-агрессивное поведение морально предосудительно.

В обоих случаях ни одна из сторон не преследовала общей цели. Ни одна из сторон не пыталась найти оптимальный результат. Давайте посмотрим, как это делается.

Майк: «Пола, страница входа в систему мне нужна к завтрашнему дню».

Пола: «Нет, Майк, здесь работы на две недели».

Майк: «Две недели? По оценкам проектировщиков, работа должна была занять три дня, а прошло уже пять!»

Пола: «Проектировщики ошибались, Майк. Они выдали свою оценку до того, как служба маркетинга сформулировала окончательные требования. У меня осталось работы еще на 10 дней. Ты не видел мои обновленные оценки в вики?»

Майк: (с суровым видом и недовольным голосом) «Это недопустимо, Пола. Завтра я буду представлять клиентам демо-версию, и я должен им показать, что страница входа работает».

Пола: «Какая часть страницы входа должна работать к завтрашнему дню?»

Майк: «Мне нужна страница входа! Я должен иметь возможность войти в систему».

Пола: «Майк, я могу сделать макет страницы входа, который позволит войти в систему. Сейчас простейший вариант уже работает. Макет не проверяет имя пользователя и пароль и не отправляет забытый пароль по электронной почте. У верхнего края нет баннера с фирменным логотипом, не работает кнопка справки и всплывающая подсказка. Страница не сохраняет cookie, чтобы запомнить данные для следующего входа, и не устанавливает ограничений доступа. Но войти в систему вы сможете. Подойдет?»

Майк: «Значит, вход будет работать?»

Пола: «Да, вход будет работать».

Майк: «Отлично, Пола, ты меня спасла!» (отходит с довольным видом)

Стороны пришли к оптимальному результату. Для этого они сказали «нет», а потом выработали взаимоприемлемое решение. Они действовали как профессионалы. В разговоре присутствовал элемент конфронтации и в нем было несколько неудобных моментов, но это неизбежно, когда два человека настойчиво преследуют несовпадающие цели.

Как насчет «почему»?

Возможно, вы думаете, что Поле следовало более подробно объяснить, *почему* работа над страницей заняла намного больше времени. По собственному опыту могу сказать, что *причины* намного менее важны, чем *факт*. А факт заключается в том, что на страницу понадобится две недели. Почему две недели – это уже второстепенно.

Впрочем, объяснение может помочь Майку понять (а следовательно, и принять) этот факт. И если у Майка имеется техническая квалификация и темперамент для понимания, такие объяснения могут быть полезными. С другой стороны, Майк может не согласиться с

выводами. Возможно, он решит, что Пола делает все неправильно. Он может сказать, что ей не нужен такой объем тестирования или рецензирования или что этап 12 можно исключить из описания. Изобилие подробностей может обернуться мелочным регламентированием.

Высокие ставки

Говорить «нет» важнее всего тогда, когда ставки высоки. Чем выше ставки, тем больше ценность сказанного «нет».

Казалось бы, утверждение очевидное. Если риск настолько велик, что от успеха зависит выживание компании, вы должны без малейших колебаний предоставить руководству самую точную информацию. А это часто означает «нет».

> **Дон** (начальник по управлению разработкой): «Итак, по нашим текущим прогнозам проект «Золотой Гусь» будет завершен через 12 недель от сегодняшнего дня, с погрешностью плюс/минус 5 недель».
>
> **Чарльз** (исполнительный директор): (четверть минуты сидит молча, постепенно багровея) «То есть ты хочешь сказать, что мы опаздываем на 17 недель?»
>
> **Дон**: «Да, это возможно».
>
> **Чарльз**: (встает, Дон встает на секунду позже) «Черт возьми, Дон! Все должно было быть готово три недели назад! Заказчик из „Галитрона" звонит каждый день и спрашивает, где его чертова система. И я должен сказать, что им придется подождать еще четыре месяца? Предложи что-нибудь получше».
>
> **Дон**: «Чак, я тебе говорил три месяца назад после реорганизации, что нам понадобится еще четыре месяца. Я хочу сказать, ты сократил мой штат на 20 %! Ты тогда сообщил „Галитрону", что мы задержим сдачу продукта?»
>
> **Чарльз**: «Ты отлично знаешь, что не сообщил. Мы не можем себе позволить потерять этот заказ, Дон. (Чарльз делает паузу, бледнеет.) Без „Галитрона" нам крышка. Ты это знаешь, верно? А теперь после этой задержки я боюсь… Что я скажу совету директоров? (Снова садится в кресло, пытаясь сохранить самообладание.) Дон, ты должен что-то придумать».
>
> **Дон**: «Я ничего не могу сделать, Чак. Мы это уже обсуждали. „Галитрон" не собирается сокращать требования и не соглашается на промежуточные версии. Они хотят, чтобы установка проводилась только один раз и на этом все заканчивалось. Я просто не смогу сделать это быстрее. Ничего не выйдет».
>
> **Чарльз**: «Черт побери. Даже если я скажу, что от этого зависит твоя работа?»
>
> **Дон**: «Если меня уволить, оценка от этого не изменится, Чарльз».
>
> **Чарльз**: «Все, разговор закончен. Возвращайся к своей группе и следи за тем, чтобы проект двигался. А мне нужно сделать несколько очень неприятных звонков».

Конечно, Чарльз должен был все сказать «Галитрону» еще три месяца назад, когда он впервые услышал новый прогноз. По крайней мере сейчас он поступает правильно, сообщая информацию заказчику (и совету директоров). Но если бы Дон не настоял на своем, эти звонки могли бы быть отложены на еще более поздний срок.

Умение работать в коллективе

Все мы слышали, как важно «уметь работать в коллективе». Это означает, что вы выполняете свои функции настолько хорошо, насколько возможно, и помогаете своим коллегам, если они окажутся в беде. Человек, умеющий работать в коллективе, часто общается с другими, обращает внимание на своих коллег и добросовестно исполняет свои обязанности.

Умение работать в коллективе вовсе не означает, что вы должны со всеми соглашаться. Рассмотрим следующую ситуацию.

Пола: «Майк, у меня свежие прогнозы. Группа согласна с тем, что демо-версия будет готова через восемь недель плюс/минус одна неделя».

Майк: «Пола, мы уже запланировали сдачу демо-версии, это будет через шесть недель».

Пола: «Даже не спросив нашего мнения? Майк, ты не можешь взвалить это на нас».

Майк: «Это уже сделано».

Пола: (вздыхает) «Ладно, я вернусь в группу и посмотрю, что мы сможем выдать через шесть недель, но это будет не вся система. Некоторые функции будут отсутствовать, а загрузка данных будет неполной».

Майк: «Пола, заказчик хочет увидеть полную демо-версию».

Пола: «Этого не будет, Майк».

Майк: «Черт. Ладно, напиши описание того, что вы сможете сделать, и передай мне завтра утром».

Пола: «Хорошо, сделаю».

Майк: «Нельзя ли как-нибудь ускорить работу? Может, работать более эффективно, более творчески?»

Пола: «Куда уж эффективнее, Майк. Мы хорошо знаем задачу, и на ее реализацию нам понадобится восемь или девять недель, но не шесть».

Майк: «Можно работать сверхурочно».

Пола: «От этого все только замедлится. Помнишь, что вышло в последний раз, когда мы выгоняли народ на сверхурочные?»

Майк: «Да, но на этот раз этого не случится».

Пола: «Все будет точно так же, Майк. Поверь мне. Нам нужно восемь или девять недель, а не шесть».

Майк: «Хорошо, напиши план, но постоянно думай о том, как справиться за шесть недель. Наверняка что-нибудь можно придумать».

Пола: «Нет, Майк, нельзя. Я могу написать план на шесть недель, но в нем не будет многих важных функций и данных. Только так, и никак иначе».

Майк: «Договорились, но я уверен, что вы сможете сотворить чудо, если попытаетесь».

(Пола уходит, качая головой.)

Позднее, на собрании у директора…

Дон: «Итак, Майк, заказчик рассчитывает получить демо-версию через шесть недель. И надеется, что все будет работать».

Майк: «Да, все будет готово. Моя группа просиживает за компьютером дни и ночи, и мы справимся. Возможно, придется поработать сверхурочно и проявить творческий подход, но мы сделаем!»

Дон: «Как хорошо, что вы понимаете интересы коллектива!»

Кто же *на самом деле* «понимает интересы коллектива» в этой ситуации? Пола действует в общих интересах, потому что она по мере возможностей сообщает, что сделать можно, а что нельзя. Она жестко защищает свою позицию, несмотря на все уговоры и нытье Майка. Майк действует в интересах коллектива из одного человека – самого Майка. Очевидно, он не думает об интересах Полы, потому что он только что пообещал за нее сделать то, что она (как она сама сказала открытым текстом) сделать не сможет. Он явно и не думал и об интересах Дона (хотя он бы с этим не согласился), потому что солгал ему.

Почему же Майк так поступил? Он хотел, чтобы Дон видел в нем человека, понимающего интересы коллектива, а также верил в свою способность манипулировать Полой для достижения шестинедельного срока. Не нужно считать Майка злым и испорченным; он просто слишком уверен в том, что ему удастся заставить других людей делать то, что он хочет.

Не пытайтесь

Худшее, что может сделать Пола в ответ на манипуляции Майка, – сказать: «Хорошо, я попытаюсь». Не хочу приплетать сюда Йоду, но в данном случае он прав. *Не надо пытаться*.

Вам не нравится эта мысль? Может, вы думаете, что пытаться что-то сделать полезно? Ведь Колумб не открыл бы Америку, если бы не пытался?

Слово «попытаться» имеет много определений. В данном случае я имею в виду значение «приложить дополнительные усилия». Какие дополнительные усилия может приложить Пола, чтобы демо-версия была готова к сроку? Если это возможно, получается, что ее группа ранее работала не в полную силу.

Обещая «попытаться», вы признаетесь в том, что ранее вы сдерживались; что у вас остался дополнительный резерв, которым вы можете воспользоваться. Вы признаетесь в том, что цель может быть достигнута посредством приложения дополнительных усилий; более того, вы фактически обязуетесь применить эти дополнительные усилия для достижения цели. Следовательно, обещая попытаться, вы обязуетесь добиться успеха. Тем самым вы взваливаете на себя тяжелое бремя. Если «попытка» не приведет к желаемому результату, это рассматривается как провал.

У вас есть дополнительный источник энергии, который вы еще не пустили в ход? И если вы задействуете его, сможете ли вы достичь поставленной цели? Или вы просто создаете условия для своего будущего провала?

Обещая попытаться, вы обещаете изменить свои планы. Прежних планов оказалось недостаточно. Обещая попытаться, вы говорите, что у вас есть новый план. Что это за план? Какие изменения вы внесете в свое поведение? Что вы собираетесь сделать иначе сейчас, когда вы «пытаетесь»?

Если у вас нет другого плана, если вы не измените свое поведение, если все будет идти точно так же, как до вашего обещания, то что тогда означает ваше «попытаюсь»?

Если вы не придерживаете часть энергии в резерве, если у вас нет нового плана, если вы не намерены изменять свое поведение и если вы достаточно уверены в исходной оценке, то ваше обещание «попытаться» в корне непорядочно. Вы *врете*. И по всей видимости, вы делаете это для того, чтобы сохранить лицо и избежать конфронтации.

Подход Полы был куда более правильным. Она продолжала напоминать Майку, что исходная оценка была неопределенной. Она всегда говорила «восемь или девять недель». Она подчеркнула существующую неопределенность и ни разу не отступила. Она ни разу не утверждала, что какие-то дополнительные усилия, или новый план, или изменение поведения смогут уменьшить эту неопределенность.

Три недели спустя…

Майк: «Пола, через три недели сдаем демо-версию. Заказчики хотят посмотреть, как работает отправка файлов».

Пола: «Майк, это не входит в согласованный список функций».

Майк: «Я знаю, но они требуют».

Пола: «Хорошо, тогда из демо-версии придется выкинуть единый вход или резервное копирование».

Майк: «Ни в коем случае! Они хотят видеть и эти функции!»

Пола: «Значит, они хотят полностью реализованную функциональность. Ты это хочешь сказать? Я же говорила, что это невозможно».

Майк: «Прости, Пола, но заказчик не уступает. Они хотят увидеть все сразу».

Пола: «Этого не будет, Майк. Просто не будет».

Майк: «Да ладно, Пола, можно хотя бы *попытаться*?»

Пола: «Майк, я могу попытаться летать по воздуху. Могу попытаться превратить свинец в золото. Могу попытаться переплыть Атлантический океан. Как ты думаешь, у меня получится?»

Майк: «Послушай, я же не требую невозможного».

Пола: «Нет, Майк, именно *требуешь*».

(Майк ухмыляется, кивает и отворачивается, собираясь отойти.)

Майк: «Я верю в тебя, Пола; знаю, что ты меня не подведешь».

Пола: (в спину Майку) «Майк, ты меня не слушаешь? Это плохо кончится».

(Майк просто машет рукой, не поворачиваясь.)

Пассивная агрессивность

Пола должна принять интересное решение. Она подозревает, что Майк не рассказывает Дону о ее оценках. Она может просто позволить Майку идти своим путем и в конечном итоге свалиться в пропасть. Ей нужно лишь позаботиться о том, чтобы в переписке были зарегистрированы все копии соответствующих служебных записок. Когда разразится катастрофа, Пола покажет всем, *что* и *когда* она говорила Майку. Это пассивно-агрессивная тактика – просто позволить Майку самому залезть в петлю.

Также Пола может попытаться предотвратить катастрофу, обратившись к Дону напрямую. Безусловно, это рискованно, но в этом и заключается суть понимания интересов коллектива. Когда прямо на вас мчится грузовой поезд и никто, кроме вас, его не видит, у вас есть выбор: либо молча отойти в сторону и посмотреть, как он задавит всех остальных, либо закричать: «Поезд! Немедленно уходите!»

Проходит два дня…

Пола: «Майк, ты сообщил Дону о моих оценках? Он сказал заказчику, что в демо-версии не будет работать функция отправки файлов?»

Майк: «Пола, ты сказала, что сделаешь это для меня».

Пола: «Нет, Майк, я этого не говорила. Я тебе сказала, что это невозможно. Вот копия служебной записки, которую я отправила тебе после нашего разговора».

Майк: «Да, но ты же сказала, что *попытаешься*, верно?»

Пола: «Мы это уже обсуждали, Майк. Помнишь, свинец и золото?»

Майк: (вздыхает) «Послушай, Пола, это очень нужно. Просто нужно. Пожалуйста, сделай все необходимое, но ты просто должна сделать это к сроку».

Пола: «Майк, ты ошибаешься. Я *должна* сделать совсем другое – сообщить Дону, если этого не сделаешь ты».

Майк: «Это нарушение субординации, так нельзя».

Пола: «А я и не хочу, Майк, но ты меня вынуждаешь».

Майк: «Ох, Пола…»

Пола: «Послушай, Майк, мы не успеем реализовать всю функциональность демо-версии вовремя. Усвой это наконец. Перестань уговаривать меня больше работать. Перестать обманывать себя, что я каким-то чудом вытащу кролика из шляпы. Пойми, что ты должен сообщить это Дону, и притом сообщить прямо сегодня».

Майк: (с широко раскрытыми глазами) «Сегодня?»

Пола: «Да, Майк, сегодня. Потому что ты, я и Дон проведем встречу, на которой обсудим функциональность, включаемую в демо-версию. Если эта встреча не состоится, я буду вынуждена сама обратиться к Дону. Вот копия служебной записки, в которой это объясняется».

Майк: «Ты просто прикрываешься!»

Пола: «Майк, я пытаюсь прикрыть нас обоих. Ты представляешь, что произойдет, если заказчик придет сюда, ожидая увидеть полную демо-версию, а мы ее не сможем предъявить?»

Чем кончится история Полы и Майка? Проработайте возможные варианты сами. Суть в том, что Пола вела себя очень профессионально. Она говорила «нет» в правильно выбранные моменты и говорила правильно. Она сказала «нет», когда на нее давили, чтобы она изменила свою оценку. Она сказала «нет» на все попытки манипуляций, лесть и мольбы.

И что самое важное – она сказала «нет» самообману и бездействию Майка. Пола действовала в интересах коллектива. Майку была необходима помощь, и она использовала все, что было в ее силах, чтобы ему помочь.

Цена согласия

Чаще мы предпочитаем говорить «да». Действительно, в здоровом коллективе люди стараются найти путь к согласию. Руководители и разработчики в хорошо управляемых группах ведут переговоры до тех пор, пока не найдут взаимоприемлемый план действий.

Но как мы уже видели, иногда, для того чтобы прийти к «да», нужно не бояться сказать «нет».

Для примера возьмем следующую историю, опубликованную Джоном Бланко в своем блоге.[8] Она воспроизводится здесь с его разрешения. Во время чтения спросите себя, когда и как ему следовало сказать «нет».

ХОРОШИЙ КОД СТАЛ НЕВОЗМОЖНЫМ?

В юношеском возрасте вы решаете стать программистом. В старших классах вы учитесь писать программы по объектно-ориентированным принципам. В колледже вы применяете усвоенные принципы в таких областях, как искусственный интеллект и 3D-графика.

А после вступления в круг профессионалов начинается ваша бесконечная работа по написанию качественного на коммерческом уровне, простого в сопровождении, «идеального» кода, который выдержит испытание временем.

Качество коммерческого уровня? Ха. Это довольно забавно.

Я считаю, что мне везет. Я люблю паттерны проектирования. Мне нравится изучать теорию идеального программирования. Я запросто могу затеять часовую дискуссию по поводу неудачного выбора иерархии наследования моим партнером по XP – что отношения типа «содержит» во многих ситуациях предпочтительнее отношений «является частным случаем». Но в последнее время мне не дает покоя один вопрос…

…Почему в современном программировании стал невозможным хороший код?

ТИПИЧНОЕ ПРЕДЛОЖЕНИЕ

Работая по контракту, я провожу свои дни (и ночи) за разработкой мобильных приложений для своих клиентов. И за те многие годы, когда я этим занимался, я понял, что требования работы на заказчика не позволяют мне писать по-настоящему качественные приложения, которые мне хотелось бы создавать.

Прежде чем я начну, позвольте сказать, что меня нельзя упрекнуть в недостатке старания. Мне нравится тема чистого кода. Я не знаю никого, кто бы стремился к идеальной архитектуре программного продукта так же сильно. Проблема кроется в исполнении, и не по той причине, о которой вы подумали.

Позвольте рассказать вам историю.

В конце прошлого года одна хорошо известная компания провела конкурс на разработку приложения. Фирма – очень крупный оператор

[8] http://raptureinvenice.com/?p=63

розничной торговли; для сохранения конфиденциальности назовем ее «Горилла Маркет». Представители заказчика указали, что им нужно организовать свое присутствие в области приложений iPhone, а приложение должно быть готово к «черной пятнице».[9] В чем проблема? Сегодня уже 1 ноября. На создание приложения остается всего 4 недели. Да, и еще в это время Apple обычно требуется до двух недель на утверждение приложений (старые добрые времена). Выходит, приложение должно быть написано… ЗА ДВЕ НЕДЕЛИ?!?

Да. У нас две недели на создание приложения. И к сожалению, наша заявка победила. (В бизнесе фигура заказчика играет важную роль.) Никуда не денешься.

«Ничего страшного, – говорит Руководитель № 1 из «Горилла Маркета», – приложение простое. Все, что нужно, – показать пользователю несколько продуктов из нашего каталога и дать ему возможность найти адреса магазинов. На нашем сайте это уже сделано. Мы дадим готовую графику. Вероятно, вы сможете использовать – как это называется? – да, жесткое кодирование!»

В разговор вступает Руководитель № 2: «И еще нам понадобятся купоны на скидку, которые пользователь сможет предъявить на кассе. Откровенно говоря, приложение пишется „на выброс”. Давайте сделаем его, а потом для фазы II „с нуля” будет написано другое, больше и лучше».

Так оно и происходит. Несмотря на годы постоянных напоминаний о том, что каждая затребованная заказчиком функция всегда оказывается сложнее, чем кажется из его объяснений, вы соглашаетесь. Вы действительно верите, что на этот раз все будет сделано за две недели. Да! Мы справимся! На этот раз все будет иначе! Несколько графических изображений и обращение к службе для получения адреса магазина. XML! Запросто. Мы справимся. Поехали!

Всего одного дня оказывается достаточно, чтобы я снова вернулся к реальности.

Я: Итак, дайте мне информацию, необходимую для вызова веб-службы с адресами магазинов.

Заказчик: А что такое «веб-служба»?

Я:…….

Именно так все и происходило. Данные об адресах магазинов, выводимые в правом верхнем углу их веб-сайта, предоставлялись вовсе не веб-службой – они генерировались кодом Java. И вдобавок хостинг обеспечивался стратегическим партнером «Горилла Маркета».

В моей ситуации мне удалось добиться от «Горилла Маркета» только текущего списка магазинов в виде файла Excel. Код поиска пришлось писать «с нуля».

Позднее в этот же день последовал второй удар: заказчик хотел, чтобы данные продуктов и купонов могли еженедельно меняться. О жестком кодировании данных можно забыть! Выходит, за две недели придется написать

[9] День массовых распродаж в США; приходится на период с 22 по 29 ноября. – *Примеч. перев.*

не только приложение для iPhone, но и исполнительную часть на PHP, и интегрировать ее с… Что? Контролем качества тоже придется заниматься мне?

Чтобы компенсировать возросший объем работы, нам придется программировать немного быстрее. Забудьте про паттерн «Абстрактная фабрика». Заменяем паттерн «Компоновщик» большим и уродливым циклом f or – некогда!

Хороший код стал невозможным.

ДВЕ НЕДЕЛИ ДО СДАЧИ ПРОЕКТА

Уверяю вас, эти две недели были довольно паршивыми. Первые два дня пропали из-за многочасовых собраний по моему следующему проекту. (Это только подчеркивает, насколько мало времени осталось для работы.) В конечном итоге на работу у меня осталось 8 дней. В первую неделю я проработал 74 часа, а в следующую… Боже… Я даже не помню, это стерлось из моих синапсов. Наверное, к лучшему.

Я провел эти восемь дней за яростным программированием. Я пустил в ход все возможные средства, чтобы справиться со своей работой: копирование/вставку (AKA повторное использование кода), «волшебные числа» (чтобы избежать дублирующихся определений констант с их последующим – о ужас! – повторным вводом) – и НИКАКИХ модульных тестов! (Кому нужны проблемы в такое время, они только отобьют охоту работать!)

Код получился довольно скверным, и у меня не было времени на рефакторинг. Впрочем, при таких сроках он был весьма неплох – ведь код все равно писался «на выброс», верно? Что-то из этого кажется вам знакомым? Подождите, дальше будет еще интереснее.

Накладывая завершающие штрихи (прежде чем переходить к написанию серверного кода), я начал поглядывать на кодовую базу и думать, что все, возможно, не так уж плохо. Ведь приложение работает, в конце концов. Я выжил!

«Боб у нас работает совсем недавно, он был очень занят и не мог позвонить раньше. А теперь он говорит, что пользователи должны вводить адреса своей электронной почты для получения купонов. Он еще не видел приложения, но думает, что это отличная идея! Кроме того, нам понадобится система построения отчетов для получения введенных адресов с сервера. И если уж речь зашла о купонах, они должны иметь ограниченный срок действия, а срок действия мы должны задавать сами. Да, и еще…»

А теперь вернемся на шаг назад. Что мы знаем о хорошем коде? Хороший код должен быть расширяемым. Простым в сопровождении. Он должен легко модифицироваться. Он должен читаться, как проза. Так вот, мой код не был хорошим.

И еще одно. Если вы хотите повысить свою квалификацию как разработчика, всегда помните: заказчик постоянно увеличивает объем работы. Он всегда хочет добавить в приложение новые возможности. Он всегда хочет вносить изменения – НА ПОЗДНЕЙ СТАДИИ.

Вот простая формула успеха:

(количество руководителей)[10]
+ 2 * количество новых руководителей
+ количество детей у Боба
= ДНЕЙ, ДОБАВЛЯЕМЫХ В ПОСЛЕДНЮЮ МИНУТУ

Руководители – такие же люди, как мы. Они должны обеспечивать свои семьи (если Сатана разрешил им завести семью). Они хотят, чтобы приложение было успешным (время повышения!). Проблема в том, что все они хотят претендовать на свою долю успеха в проекте. После того как все будет сказано и сделано, они хотят указать на некоторую функцию или архитектурное решение, которое они бы могли назвать своей личной заслугой.

Но вернемся к нашему проекту. Мы добавили еще пару дней и реализовали ввод адресов электронной почты. А потом я упал в обморок от усталости.

ЗАКАЗЧИК МЕНЬШЕ БЕСПОКОИТСЯ О ПРОЕКТЕ, ЧЕМ ВЫ САМИ

Клиенты, несмотря на все их заявления, несмотря на очевидную срочность, никогда не беспокоятся о нарушении графика сильнее, чем вы. В день завершения работы над приложением я разослал сообщение с финальной сборкой всем ключевым участникам. Руководителям (гррр!), менеджерам и т. д. «ГОТОВО! ВОТ ВАМ ВЕРСИЯ 1.0! СЛАВА БОГУ!» Я нажал кнопку «Отправить», откинулся в кресле и с довольной ухмылкой начал представлять себе, как заказчики несут меня на руках, а на 42-й улице проходит парад, где меня венчают лаврами «Величайшего Разработчика Всех Времен». По крайней мере, мое лицо должно быть на их рекламе, верно?

Как ни странно, фирма-заказчик не торопилась меня хвалить. Я вообще не знал, что они думают по этому поводу. Я не получил никакой реакции. Ни единого сообщения. Похоже, руководство «Горилла Маркет» решило, что этот этап уже пройден, и перешло к следующему проекту.

Думаете, я вру? Убедитесь сами. Я подал заявку в Apple с незаполненным описанием приложения. Я запросил описание у «Горилла Маркета», но мне никто не ответил, а ждать было некогда (см. предыдущий абзац). Я написал снова. И снова. Подключил к этому наше руководство. Дважды мне звонили, и дважды я слышал: «Напомните, что вам было нужно?» МНЕ БЫЛО НУЖНО ОПИСАНИЕ ПРИЛОЖЕНИЯ!

Через неделю началось тестирование приложения в Apple. Обычно это время радости, но для меня это было время смертельного ужаса. Как и ожидалось, через день приложение было отклонено по самой жалкой и неубедительной причине, которую я только могу себе представить: «У приложения отсутствует описание». С функциональностью все в порядке; нет описания. И по этой причине приложение «Горилла Маркет» не было готово к «черной пятнице». Меня это порядком раздражало.

Я пожертвовал своим общением с семьей ради двухнедельного рабочего марафона, а в «Горилла Маркете» за целую неделю никто не побеспокоился

[10] Возможно, за исключением непосредственного работодателя Джона, хотя я готов поспорить, что он тоже оказался в проигрыше.

создать описание приложения! Мы получили описание через час после отказа Apple.

И если до этого я испытывал раздражение, то через полторы недели я пришел в полную ярость. Оказалось, что заказчик не предоставил нам реальные данные. Продукты и купоны на сервере были фиктивными. Условными, если хотите. Код купона был равен 1234567890 – просто взят с потолка.

А потом наступило судьбоносное утро, когда я зашел на Портал – И ПРИЛОЖЕНИЕ БЫЛО ДОСТУПНО! С фиктивными данными и всем прочим! Я в ужасе названивал всем, кому было можно, и вопил: «МНЕ НУЖНЫ ДАННЫЕ!» Женский голос спросил, с кем меня соединить – с пожарными или с полицией, и я повесил трубку. Но потом я все-таки дозвонился в «Горилла Маркет» со своим «МНЕ НУЖНЫ ДАННЫЕ!» И я никогда не забуду ответ:

«Здравствуйте, это Джон. У нас сменился вице-президент, и мы решили отказаться от выпуска. Отзовите его из App Store, хорошо?»

В итоге оказалось, что не менее 11 людей зарегистрировали свои адреса в базе данных. Это означало, что теоретически 11 людей могут заявиться в «Горилла Маркет» с фальшивым купоном. Весело, правда?

В общем, во всех утверждениях заказчика правдой было только одно: код действительно писался «на выброс». Единственная проблема заключалась в том, что он вообще не был использован.

РЕЗУЛЬТАТ? СПЕШКА С ЗАВЕРШЕНИЕМ, МЕДЛЕННЫЙ ВЫХОД НА РЫНОК

Мораль этой истории: ключевые участники проекта (будь то внешний заказчик или внутренний руководитель) придумали схему, которая заставит разработчиков быстро писать код. Эффективно? Нет. Быстро? Да. Вот как работает эта схема.

– Сказать разработчику, что приложение очень простое. Это создает у группы разработки искаженное представление о масштабах работы. Кроме того, разработчики быстро берутся за работу, а тем временем…

– Функциональность проекта расширяется, причем рабочая группа оказывается виноватой в том, что не распознала эту необходимость заранее. В нашем случае жесткое кодирование контента должно было привести к усложнению обновлений. Как я мог этого не понять сразу? Я понял, но до этого я получил лживые обещания от заказчика. Или другой вариант: заказчик нанимает «нового человека», который находит какое-нибудь явное упущение. А завтра заказчик скажет, что они приняли на работу Стива Джобса, и в приложение нужно добавить алхимические трансформации? Далее…

– Проект постоянно подгоняется, чтобы работа была завершена к исходному сроку. Разработчики трудятся на максимальной скорости (и с максимальным риском ошибок, но кто станет обращать на это внимание?). До срока остается пара дней? Зачем говорить, что срок сдачи можно перенести, если работа идет так продуктивно? Нужно использовать это в своих интересах!

Потом срок наступает, добавляется еще несколько дней, потом неделя – и это после того, как вы отработаете 20-часовую смену, чтобы все было сделано вовремя. Все как на знаменитой картинке с ослом и морковкой – не считая того, что с ослом обращаются намного лучше, чем с вами.

Схема отлично придумана. Можно ли обвинять ее создателей, уверенных в том, что она работает? Просто они не видят кошмарного кода. И так происходит снова и снова, несмотря на результаты.

В условиях глобализированной экономики, когда корпорации держатся за всемогущий доллар, а повышение котировки акций связано с сокращением штата, сверхурочной работой и оффшорной разработкой, описанная стратегия экономии на разработчиках делает хороший код невозможным. Если мы, разработчики, не проявим должной осторожности, то нас просьбами/приказами/угрозами заставят писать вдвое больший объем кода за половину времени.

О невозможности хорошего кода

Когда в этой истории Джон спрашивает: «Хороший код стал невозможным?», в действительности он спрашивает: «Профессионализм стал невозможным?» В конце концов, в этой печальной истории пострадал не только код. Пострадала его семья, его работодатель, заказчики и пользователи. В проигрыше оказались все.[11] И проигрыш объясняется непрофессионализмом.

Кто здесь действовал непрофессионально? Джон недвусмысленно намекает, что это были руководители «Горилла Маркета». Схема, описанная им в конце, ясно указывает на их непорядочное поведение. Но было ли это поведение *плохим*? Я так не думаю.

Они хотели иметь приложение для iPhone к «черной пятнице». Они были готовы заплатить за него. Они нашли кого-то, кто возьмется за эту работу. Так в чем их винить?

Да, в процессе общения явно возникали проблемы. И очевидно, руководители фирмы-заказчика не знали, что такое веб-служба; это обычное дело – одно подразделение крупной корпорации не знает, чем занимается другое. Но все это следовало предвидеть. Джон даже признает это, когда говорит: «Несмотря на годы постоянных напоминаний о том, что каждая затребованная заказчиком функция всегда оказывается сложнее, чем кажется из его объяснений…»

Итак, если виновником был не «Горилла Маркет», то кто?

Возможно, непосредственный работодатель. Джон явно не говорит об этом, но намекает в своей снисходительной фразе: «В бизнесе фигура заказчика играет важную роль». Может, работодатель Джона раздавал неразумные обещания «Горилла Маркету»? Он оказывал давление на Джона (прямое или косвенное), чтобы эти обещания оправдались? Джон не говорит об этом, так что мы можем только догадываться.

Но даже если так, за что в этой истории отвечает сам Джон? Я возлагаю всю ответственность исключительно на него. Это Джон согласился на исходный двухнедельный срок, отлично зная, что проекты обычно оказываются более сложными, чем кажется на первый взгляд. Это Джон согласился написать серверную часть на PHP. Это Джон согласился на требование о регистрации по электронной почте и ограничении срока действия купона. Это Джон работал по 20 часов в сутки и по 90 часов в неделю. Это Джон отказался от своей семьи и нормальной жизни, чтобы не сорвать срок сдачи.

Почему Джон так поступил? Он об этом говорит вполне определенно: «Я нажал кнопку „Отправить“, откинулся в кресле и с довольной ухмылкой начал представлять себе, как заказчики несут меня на руках, а на 42-й улице походит парад, где меня венчают лаврами „Величайшего Разработчика Всех Времен“». Короче говоря, Джону захотелось быть героем. Он увидел шанс добиться славы и ухватился за него.

Профессионалы часто совершают героические дела, но не потому, что хотят быть героями. Профессионалы становятся героями, когда они хорошо выполняют свою работу, без нарушения сроков и бюджета. Стремясь стать «героем дня», Джон действовал непрофессионально.

Джон должен был сказать «нет» на исходный двухнедельный срок. А если не сказал – то должен был сделать это, когда обнаружил, что никакой веб-службы для получения данных не существует. Он должен был сказать «нет» в ответ на требование регистрации по электронной почте и ограничения срока действия купонов. Он должен был сказать «нет» на все, что приводит к немыслимым жертвам и перерасходу времени.

---

[11] Правда, никаких денег я на этом не потерял. Я продал свой патент Teradyne за 1 доллар согласно условиям контракта (хотя и этот доллар я не получил).

Но самое главное, Джон должен был сказать «нет» своему внутреннему решению о том, что выполнить работу в установленный срок можно только одним способом – устроив неразбериху в коде. Обратите внимание, что говорит Джон о хорошем коде и модульных тестах: «Чтобы компенсировать возросший объем работы, нам придется программировать немного быстрее. Забудьте про паттерн „Абстрактная фабрика“. Заменяем паттерн „Компоновщик“ большим и уродливым циклом for – некогда!»

И еще раз:

«Я провел эти восемь дней за яростным программированием. Я пустил в ход все возможные средства, чтобы справиться со своей работой: копирование/вставку (AKA повторное использование кода), „волшебные числа“ (чтобы избежать дублирующихся определений констант с их последующим – о ужас! – повторным вводом) – и НИКАКИХ модульных тестов! (Кому нужны проблемы в такое время, они только отобьют охоту работать!)»

Все эти решения и стали истинной причиной катастрофы. Джон принял тот факт, что прийти к успеху можно только через непрофессиональное поведение, и получил то, чего заслужил.

Возможно, это звучит излишне сурово. Я этого не хотел. В предыдущих главах я описал, как неоднократно совершал ту же ошибку в своей карьере. Искушение «быть героем» и «решать проблемы» велико. Однако все мы должны понять, что отказ от профессиональных принципов не решает проблемы, а создает их. Учитывая сказанное, я наконец-то могу ответить на исходный вопрос Джона:

«Хороший код стал невозможным? Профессионализм стал невозможным?»

Я говорю «*нет*»!

3
Как сказать «да»

А вы знаете, что я изобрел голосовую почту? Честное слово. Вообще-то нас, владельцев патента на голосовую почту, было трое: Кен Файндер, Джерри Фитцпатрик и я. Это было в начале 80-х годов, когда мы работали на компанию Teradyne. Наш исполнительный директор поручил нам создать продукт нового типа, и мы изобрели «электронного секретаря» (сокращенно ЭС).

Все вы отлично знаете, что такое «электронный секретарь». Это одна из тех кошмарных машин, которые отвечают на звонки в компаниях и задают всевозможные идиотские вопросы, на которые нужно отвечать нажатием кнопок («Чтобы переключиться на английский язык, нажмите 1»).

Наш ЭС отвечал на звонок и просил ввести имя нужного человека, после чего вызывал его по внутренней связи. ЭС сообщал о вызове и спрашивал, соединить ли со звонившим. Если ответ был положительным, он устанавливал связь со звонившим и отключался.

Вы могли сообщить ЭС свое текущее местонахождение. Также ему можно было задать несколько телефонных номеров. Если вы были в другом офисе, ЭС находил вас. Если вы были дома, ЭС находил вас. Если вы находились в другом городе, ЭС находил вас. А если у него это все же не получилось, он оставлял сообщение. Именно здесь была впервые применена голосовая почта.

Как ни странно, фирма Teradyne не смогла придумать, как бы ей продать «электронного секретаря». Проект вышел за рамки бюджета и был преобразован в систему CDS (Craft Dispatch System) для передачи специалистам по ремонту телефонов информации о следующем задании.

Также фирма Teradyne отказалась от патента, не сообщив об этом нам (!). Текущий владелец патента подал заявку на три месяца позже нас (!!).[12]

После того как ЭС превратился в CDS (но задолго до отказа от патента), я ждал исполнительного директора компании… на дереве. Перед фасадом здания рос большой дуб, я забрался на него и ждал, пока подъедет его «Ягуар». Я перехватил его у двери и попросил уделить мне несколько минут; он согласился.

Я сказал, что проект ЭС нужно запускать заново и что мы наверняка на нем сможем заработать. Но мой собеседник удивил меня, сказав: «Хорошо, Боб, подготовь план. Покажи, как мы на этом сможем заработать. Если сделаешь, а я твоему плану поверю, мы снова запустим ЭС».

[12] Мартин Р. Чистый код. Создание, анализ и рефакторинг. СПб.: Питер, 2010.

Конец ознакомительного фрагмента.

Текст предоставлен ООО «ЛитРес».

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

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