Роберт Мартин

Идеальный программист

Как стать профессионалом разработки ПО

PH PTR

ПИТЕР

Роберт С. Мартин

Идеальный программист. Как стать профессионалом разработки ПО

Серия «Библиотека программиста (Питер)»

http://www.litres.ru/pages/biblio_book/?art=11961798
Роберт Мартин. Идеальный программист. Как стать профессионалом разработки ПО: Питер; Санкт-Петербург; 2012
ISBN 978-5-459-01044-2

Аннотация

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.

В формате ios.epub представлен издательский файл.

Содержание

Роберт Мартин
Идеальный программист. Как стать профессионалом разработки ПО

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

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 увольня-

[1] Без паники!

ли. 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

[2] Технический термин неизвестного происхождения.

остановилась. Моему папе вообще трудно отказать.) Меня приняли на работу, посадили в комнату, где хранились все руководства к компьютерам IBM, и заставили 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, со-

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

держала 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 за-

крытыми 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.

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 но-

вичками.

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 систе-

му, 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 мо-

их 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

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

сущность профессионализма. Потому что профессионализм – это ответственное отношение к делу.

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 тари-

фов. 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 функци-

ональность 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 следователь-

но, 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 долж-

но 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 безответствен-

ности. 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 автома-

тизированными 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 функци-

ональности 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

---

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

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.

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. Го-

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

ре 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?

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 вы-

полнять 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.

Конец ознакомительного фрагмента.