Тестирование JavaScript

Лукас да Коста

MANNING

ББК 32.973.2-018-07
УДК 004.415.53
К71

Коста Лукас да

К71 Тестирование JavaScript. — СПб.: Питер, 2023. — 592 с.: ил. — (Серия «Библиотека программиста»).

ISBN 978-5-4461-3913-2

Автоматизированное тестирование — залог стабильной разработки качественных приложений. Полноценное тестирование должно охватывать отдельные функции, проверять интеграцию разных частей вашего кода и обеспечивать корректность с точки зрения пользователя. Книга научит вас быстро и уверенно создавать надежное программное обеспечение. Вы узнаете, как реализовать план автоматизированного тестирования для JavaScript-приложений. В издании описываются стратегии тестирования, обсуждаются полезные инструменты и библиотеки, а также объясняется, как развивать культуру, ориентированную на качество. Вы исследуете подходы к тестированию как серверных, так и клиентских приложений, а также научитесь проверять свое программное обеспечение быстрее и надежнее.

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

ББК 32.973.2-018-07
УДК 004.415.53

Права на издание получены по соглашению с Manning Publications. Все права защищены. Никакая часть данной книги не может быть воспроизведена в какой бы то ни было форме без письменного разрешения владельцев авторских прав.

Информация, содержащаяся в данной книге, получена из источников, рассматриваемых издательством как надежные. Тем не менее, имея в виду возможные человеческие или технические ошибки, издательство не может гарантировать абсолютную точность и полноту приводимых сведений и не несет ответственности за возможные ошибки, связанные с использованием книги.

Издательство не несет ответственности за доступность материалов, ссылки на которые вы можете найти в этой книге. На момент подготовки книги к изданию все ссылки на интернет-ресурсы были действующими.

ISBN 978-1617297915 англ.

ISBN 978-5-4461-3913-2

Original English language edition published by Manning Publications USA.
© 2021 by Manning Publications.

© Перевод на русский язык ООО «Прогресс книга», 2022

© Издание на русском языке, оформление ООО «Прогресс книга», 2022

© Серия «Библиотека программиста», 2022

Оглавление

Предисловие 14

Благодарности 16

О книге. 18
- Кому следует прочитать эту книгу 18
- Структура издания: дорожная карта 19
- О коде 21
- Системные требования. 21
- От издательства 22

Об авторе 23

Иллюстрация на обложке 24

Часть I
Тестирование приложений на JavaScript

Глава 1. Введение в автоматизированное тестирование 26
- 1.1. Что такое автоматизированный тест 28
- 1.2. Почему автоматизированные тесты важны. 34
  - 1.2.1. Предсказуемость 34
  - 1.2.2. Воспроизводимость 38
  - 1.2.3. Совместная работа 39
  - 1.2.4. Скорость 41
- Резюме 43

Глава 2. Что и когда тестировать. 45
2.1. Пирамида тестов 46
2.2. Модульные тесты. 52
2.3. Интеграционные тесты 61
2.4. Сквозные тесты 68
2.4.1. Тестирование API на основе HTTP 70
2.4.2. Тестирование GUI 75
2.4.3. Приемочные и сквозные тесты не одно и то же 77
2.5. Исследовательское тестирование и ценность обеспечения качества . . 78
2.6. Тесты, расходы и доходы 81
Резюме 89

Часть II
Написание тестов

Глава 3. Методы тестирования 93
3.1. Организация наборов тестов. 95
3.1.1. Разбивка тестов 100
3.1.2. Параллелизм. 104
3.1.3. Глобальные хуки 106
3.1.4. Атомарность 107
3.2. Написание полезных утверждений. 109
3.2.1. Утверждения и обработка ошибок 110
3.2.2. Нестрогие утверждения 113
3.2.3. Пользовательские сопоставители. 118
3.2.4. Циклические утверждения 120
3.3. Тестовые дублеры: макеты, заглушки и шпионы 122
3.3.1. Макетирование операций импорта 135
3.4. Выбор участков кода для тестирования 139
3.4.1. Не тестируйте стороннее ПО 140
3.4.2. Макетировать или не макетировать, вот в чем вопрос 141
3.4.3. Если сомневаетесь, выбирайте интеграционные тесты 144
3.5. Покрытие кода тестами 145
3.5.1. Автоматические отчеты о покрытии кода тестами 147
3.5.2. Виды покрытия 147
3.5.3. О чем говорит покрытие кода, а о чем — нет 148
Резюме 150

Глава 4. Тестирование серверных приложений. 153
4.1. Структурирование среды тестирования. .155
4.1.1. Сквозное тестирование 157
4.1.2. Интеграционное тестирование 160
4.1.3. Модульное тестирование 166
4.2. Тестирование HTTP-путей. 169
4.2.1. Тестирование промежуточного слоя .174
4.3. Работа с внешними зависимостями 184
4.3.1. Интеграция с базой данных 184
4.3.2. Интеграция с другими API. 197
Резюме 206

Глава 5. Передовые методы тестирования серверных приложений 207
5.1. Устранение недетерминированности 208
5.1.1. Параллелизм и разделяемые ресурсы 210
5.1.2. Работа с временем 214
5.2. Снижение расходов без ущерба для качества 225
5.2.1. Уменьшение количества повторяющегося кода в разных тестах 226
5.2.2. Создание транзитивных гарантий 231
5.2.3. Превращение утверждений в предусловия 232
Резюме 234

Глава 6. Тестирование клиентских приложений 236
6.1. Знакомство с JSDOM 238
6.2. Утверждения об элементах DOM. 246
6.2.1. Упрощение поиска элементов 255
6.2.2. Написание более совершенных утверждений 257
6.3. Обработка событий 261
6.4. Тестирование и API браузера 272
6.4.1. Тестирование интеграции с localStorage. 273
6.4.2. Тестирование интеграции с History API 277
6.5. Работа с веб-сокетами и HTTP-запросами 290
6.5.1. Тесты с использованием HTTP-запросов. 291
6.5.2. Тесты с использованием веб-сокетов. 297
Резюме 305

Глава 7. Экосистема тестирования React 307
7.1. Подготовка тестового окружения для React 308
7.1.1. Создание приложения на основе React 309
7.1.2. Подготовка тестового окружения. 315
7.2. Краткий обзор библиотек для тестирования React. 318
7.2.1. DOM и отрисовка компонентов. 319
7.2.2. Библиотека для тестирования React 325
7.2.3. Enzyme 339
7.2.4. Тестовый метод отрисовки React 341
Резюме 342

Глава 8. Тестирование приложений на основе React 344
8.1. Тестирование интеграции между компонентами 345
8.1.1. Создание заглушек для компонентов 355
8.2. Тестирование с использованием снимков. 360
8.2.1. Использование снимков для любых типов данных. 371
8.2.2. Средства сериализации 374
8.3. Тестирование стилей. 374
8.4. Истории и приемочное тестирование на уровне
отдельного компонента. 385
8.4.1. Написание историй 386
8.4.2. Написание документации 395
Резюме 398

Глава 9. Разработка через тестирование 400
9.1. Философия, лежащая в основе TDD. 402
9.1.1. Что такое TDD 403
9.1.2. Выбор масштабов итераций 411
9.1.3. Зачем применять разработку через тестирование. 414
9.1.4. Когда не следует применять TDD 418
9.2. Написание модуля JavaScript с помощью TDD 419
9.3. Нисходящее и восходящее тестирование 430
9.3.1. Что такое нисходящее и восходящее тестирование. 431
9.3.2. Как нисходящее и восходящее тестирование влияет
на рабочий процесс TDD. 436
9.3.3. Преимущества и недостатки нисходящего
и восходящего подходов 438

9.4. Баланс между расходами на обслуживание, темпами разработки и хрупкостью тестов 444
9.4.1. Реализация через тестирование 445
9.4.2. Поддержка кода через тестирование 448
9.5. Подготовка окружения для успешного использования TDD 451
9.5.1. Внедрение в рамках группы разработки. 451
9.5.2. Соблюдение четких границ. 454
9.5.3. Работа в паре. 456
9.5.4. Дополнительное тестирование 457
9.6. TDD, BDD и проверки 458
Резюме 461

Глава 10. Сквозные тесты на основе пользовательского интерфейса 464
10.1. Что такое сквозные тесты на основе пользовательского интерфейса 465
10.2. Когда уместен тот или иной вид тестов 467
10.2.1. Сквозные тесты на основе UI 468
10.2.2. Чистые сквозные тесты 470
10.2.3. Чистые UI-тесты 471
10.2.4. Примечание о приемочном тестировании и о названии главы 472
10.3. Обзор инструментов для сквозного тестирования 473
10.3.1. Selenium 474
10.3.2. Puppeteer 479
10.3.3. Cypress. 481
10.3.4. Когда следует использовать Cypress 483
Резюме 485

Глава 11. Написание сквозных тестов на основе пользовательского интерфейса 486
11.1. Ваши первые сквозные тесты на основе UI 488
11.1.1. Подготовка тестового окружения 488
11.1.2. Написание ваших первых тестов 491
11.1.3. Отправка HTTP-запросов 501
11.1.4. Выполнение действий в определенном порядке 505
11.2. Рекомендации по написанию сквозных тестов. 509
11.2.1. Объекты страницы 509
11.2.2. Действия приложения 518

11.3. Борьба с хрупкостью тестов 523
11.3.1. Отказ от фиксированных интервалов ожидания 524
11.3.2. Создание заглушек для неподконтрольных вам факторов 528
11.3.3. Повторные попытки выполнения тестов 539
11.4. Выполнение тестов в разных браузерах 542
11.4.1. Использование средств тестирования для выполнения тестов в браузере 544
11.4.2. Выполнение тестов на основе UI в разных браузерах 545
11.5. Визуальные регрессионные тесты. 546
Резюме 550

Часть III
Бизнес-аспекты тестирования

Глава 12. Непрерывная интеграция и непрерывная доставка. 554
12.1. Что такое непрерывная интеграция и непрерывная доставка 556
12.1.1. Непрерывная интеграция 557
12.1.2. Непрерывная доставка. 561
12.2. Роль автоматизированных тестов в процессе CI/CD. 565
12.3. Проверки на уровне системы контроля версий 567
Резюме 569

Глава 13. Культура, ориентированная на качество. 571
13.1. Использование системы типов для исключения представления некорректных состояний .572
13.2. Разбор кода для выявления проблем, которые не могут быть обнаружены компьютером 578
13.3. Использование средств анализа и форматирования для получения однородного кода 581
13.4. Мониторинг систем для понимания того, как они ведут себя на самом деле 583
13.5. Описание программного обеспечения с помощью хорошей документации 585
Резюме 586

1 Введение в автоматизированное тестирование

В этой главе

- ✓ Что такое автоматизированный тест.
- ✓ Для чего пишутся автоматизированные тесты.
- ✓ Как автоматизированные тесты могут помочь в написании лучшего кода за меньшее время и с большей уверенностью.

Когда программное обеспечение пронизывает все вокруг, от кондитерской вашего дяди до экономики вашей страны, спрос на новые возможности растет экспоненциально. Тем большую важность приобретает частая доставка рабочего кода — желательно по нескольку раз в день. Именно для этого нужны автоматизированные тесты. Уже давно прошли те времена, когда программисты могли позволить себе вручную и нерегулярно тестировать код. В наши дни написание тестов — это не просто рекомендуемый подход, а отраслевой стандарт. Если прямо сейчас поискать актуальные вакансии, то почти все они требуют той или иной степени осведомленности об автоматизированном тестировании ПО.

Неважно, сколько у вас клиентов и с какими объемами данных вы имеете дело. Написание эффективных тестов является полезной практикой для компаний любого масштаба, от гигантов Кремниевой долины, имеющих венчурный капитал, до индивидуального стартапа, который вы недавно запустили. Тесты способствуют общению между разработчиками и помогают избегать ошибок,

поэтому их рекомендуется применять в проектах любых размеров. Чем больше разработчиков занимаются проектом и чем выше цена неудачи, тем важнее наличие тестов.

Книга «Тестирование JavaScript» нацелена на профессионалов, уже умеющих писать программное обеспечение, но еще не знающих, как писать тесты или почему это так важно. При работе над книгой я ориентировался на людей, которые или только что окончили курсы программирования, или недавно начали карьеру программиста и хотят профессионально расти. Я рассчитываю, что читатели знают основы JavaScript и понимают такие концепции, как объекты `Promise` и функции обратного вызова. Не обязательно быть специалистом в JavaScript — достаточно уметь писать программы, которые работают. Если все сходится и вы заинтересованы в создании самого ценного вида ПО — работающего, то эта книга для вас.

Данный материал *не* предназначен для профессионалов в области обеспечения качества или нетехнических руководителей: темы разбираются с точки зрения разработчика и акцент делается на использовании результатов тестирования для написания более качественного кода за меньшее время. Я *не стану* рассказывать о том, как проводить ручное или исследовательское тестирование. Здесь вы также не найдете информации об оформлении отчетов об ошибках или управлении рабочими процессами тестирования. Эти задачи пока что *нет возможности* автоматизировать. Если хотите узнать о них больше, советую поискать книгу, предназначенную для тех, кто работает в сфере QA.

Основным инструментом, который мы будем использовать, является Jest. Вы овладеете им в ходе написания рабочих автоматизированных тестов для нескольких небольших приложений. Приложения будут написаны как на чистом JavaScript, так и с использованием популярных библиотек, таких как Express и React. Предварительный опыт с Express и особенно с React будет полезен, но краткого знакомства с этими инструментами тоже достаточно. Все примеры я буду разрабатывать с нуля, требуя от вас как можно меньше предварительных знаний, поэтому советую учиться на ходу и не готовиться заранее.

В главе 1 мы пройдемся по концепциям, лежащим в основе всех последующих примеров. Как показывает мой опыт, самая главная причина написания плохих тестов связана с непониманием того, что они собой представляют и какие задачи перед ними можно и нужно ставить. Именно с этого мы и начнем.

Разобравшись с тем, что такое тесты и для чего их пишут, мы обсудим разные ситуации, в которых написание тестов может способствовать как написанию более качественного ПО за меньшее время, так и взаимодействию между разработчиками. Эти основополагающие идеи будут играть ключевую роль в главе 2, когда мы начнем писать первые тесты.

1.1. ЧТО ТАКОЕ АВТОМАТИЗИРОВАННЫЙ ТЕСТ

У дяди Луиса не было ни единого шанса преуспеть в Нью-Йорке, а вот в Лондоне он стал известен своими ванильными творожными десертами. Ввиду их необычайной популярности дядя Луис быстро понял, что управление кондитерской с помощью ручки и бумаги уже не отвечает масштабам продаж. Чтобы поспевать за растущим спросом, он нанял лучшего известного ему программиста (вас) и поручил создать интернет-магазин.

Требования были простыми: у клиентов должна быть возможность заказывать кондитерские изделия, вводить адрес доставки и оплачивать заказы по интернету. Выполнив поставленную задачу, вы решили убедиться в том, что магазин работает как следует. Вы создали базы данных (БД), наполнили их тестовой информацией, запустили сервер и открыли созданный сайт у себя на компьютере. Попытавшись заказать парочку пирожных, вы обнаружили ошибку: например, заметили, что в корзину покупок можно добавить только одну единицу какого-либо товара.

Если бы эта ошибка осталась в готовом сайте, дядя Луис столкнулся бы с огромными проблемами. Таким образом, вы решили, что возможность добавления нескольких единиц товара *всегда* нужно проверять.

Вы могли бы вручную тестировать каждую версию, как делалось когда-то в старых сборочных цехах. Но такой подход не масштабируется: он занимает слишком много времени и, как это свойственно любым ручным процессам, чреват ошибками. Чтобы можно было решить проблему, вместо вас роль покупателя должен играть код.

Подумаем, как пользователь сообщает программе о том, что в корзину нужно что-то добавить. Это упражнение поможет определить, какие этапы последовательности действий необходимо заменить автоматизированными тестами.

Пользователи взаимодействуют с вашим приложением через сайт, который шлет HTTP-запрос на сервер. Запрос информирует функцию `addToCart` о том, какой товар и в каком количестве нужно добавить в корзину. Корзина покупателя идентифицируется по сеансу отправителя. После добавления товара в корзину сайт обновляется в соответствии с ответом сервера. Этот процесс показан на рис. 1.1.

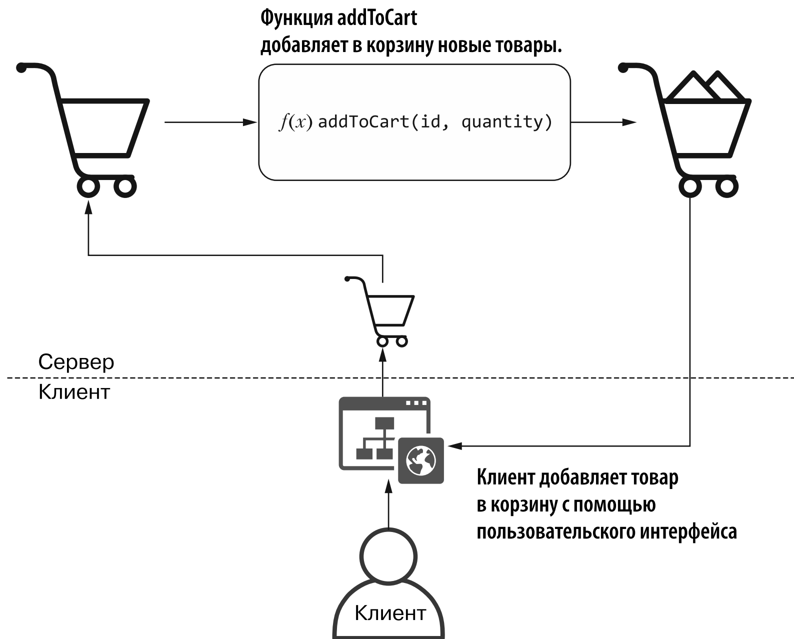

Рис. 1.1. Последовательность действий при создании заказа

ПРИМЕЧАНИЕ

f(x) — это просто обозначение функций, которое будет использоваться в схемах. Оно ничего не говорит о том, какие у функции параметры.

Заменим покупателя программным кодом, способным вызывать функцию `addToCartFunction`. Таким образом, вам не нужно будет полагаться на то, что кто-то вручную добавит товар в корзину, чтобы потом проверить ответ: у вас будет фрагмент кода, выполняющий проверку. Это и есть автоматизированный тест.

АВТОМАТИЗИРОВАННЫЙ ТЕСТ

Автоматизированные тесты — программы, которые автоматизируют процесс тестирования ПО. Они взаимодействуют с вашим приложением для выполнения каких-то действий и затем сравнивают полученный результат с ожидаемым выводом, определенным заранее.

Код вашего теста создает корзину покупок и просит функцию `addToCart` добавить в нее товары. Затем тест проверяет, присутствуют ли в полученном ответе запрошенные элементы (рис. 1.2).

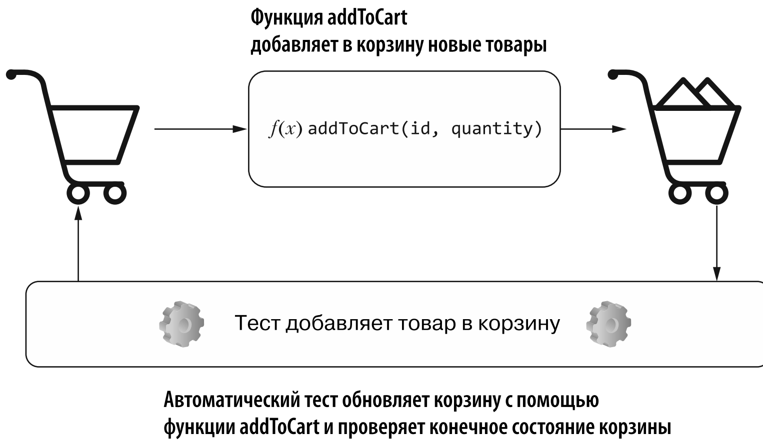

Рис. 1.2. Последовательность действий для тестирования addToCart

В тесте можно смоделировать ситуацию, в которой пользователи смогут добавить в корзину всего одно миндальное пирожное.

1. Создайте экземпляр корзины.
2. Вызовите функцию `addToCart` и укажите ей добавить в корзину миндальное пирожное.
3. Проверьте, есть ли в корзине два миндальных пирожных.

Заставив тест воспроизвести шаги, которые ранее привели к ошибке, вы можете доказать, что этой конкретной ошибки больше не происходит.

Теперь напишите тест, гарантирующий, что в корзину можно добавить больше одного миндального пирожного: он будет создавать собственный экземпляр корзины и использовать функцию `addToCart` для того, чтобы добавить в нее два миндальных пирожных. После вызова функции `addToCart` тест проверит содержимое корзины: если оно соответствует ожиданиям, значит, все работает правильно. Это позволит вам удостовериться в том, что в корзину можно добавить два миндальных пирожных (рис. 1.3).

Теперь, когда клиенты, как и положено, могут купить столько миндальных пирожных, сколько пожелают, представьте, что вы пытаетесь смоделировать покупку клиентом 10 000 миндальных пирожных. К вашему удивлению, заказ без проблем проходит. Однако у дяди Луиса нет в наличии такого количества пирожных! Его кондитерская еще небольшая и не в состоянии выполнять огромные заказы в короткие сроки. Чтобы вовремя доставлять безупречные десерты всем желающим, Луис просит вас дать клиентам возможность заказывать только то, что есть в наличии.

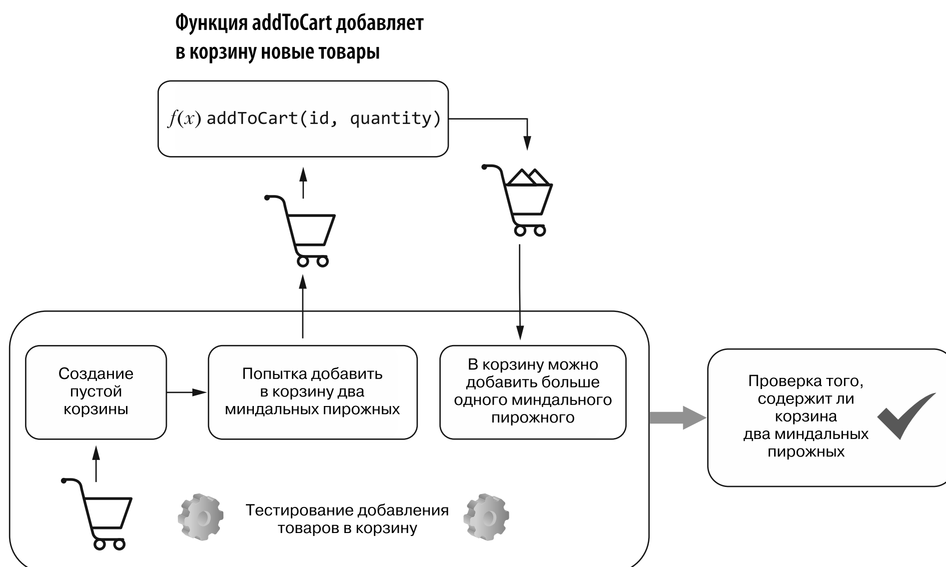

Рис. 1.3. Последовательность действий для теста, который проверяет, можно ли добавить в корзину несколько миндальных пирожных

Для определения того, какую часть последовательности действий нужно заменить автоматизированными тестами, подумайте, что должно происходить, когда клиенты добавляют товары в свои корзины, и соответствующим образом адаптируйте приложение.

Когда покупатель нажимает на сайте кнопку Добавить в корзину, как показано на рис. 1.4, клиентский код должен отправить серверу HTTP-запрос с указанием добавить в корзину 10 000 миндальных пирожных. Прежде чем выполнить просьбу, серверу нужно свериться с базой данных и убедиться в том, что товар имеется в наличии в достаточном количестве. Если количество имеющихся пирожных не меньше числа, указанного в запросе, товар добавляется в корзину и сервер возвращает ответ клиентскому коду, который обновляется соответствующим образом.

ПРИМЕЧАНИЕ

Для тестирования следует использовать отдельную базу данных. Не засоряйте свою производственную базу тестовой информацией.

Тесты добавляют и изменяют всевозможные данные, что может привести к их потере или к несогласованности содержимого БД.

Применение отдельной базы данных также упрощает поиск первопричины программной ошибки. Поскольку вы полностью контролируете состояние БД, действия клиентов не будут искажать результаты ваших тестов.

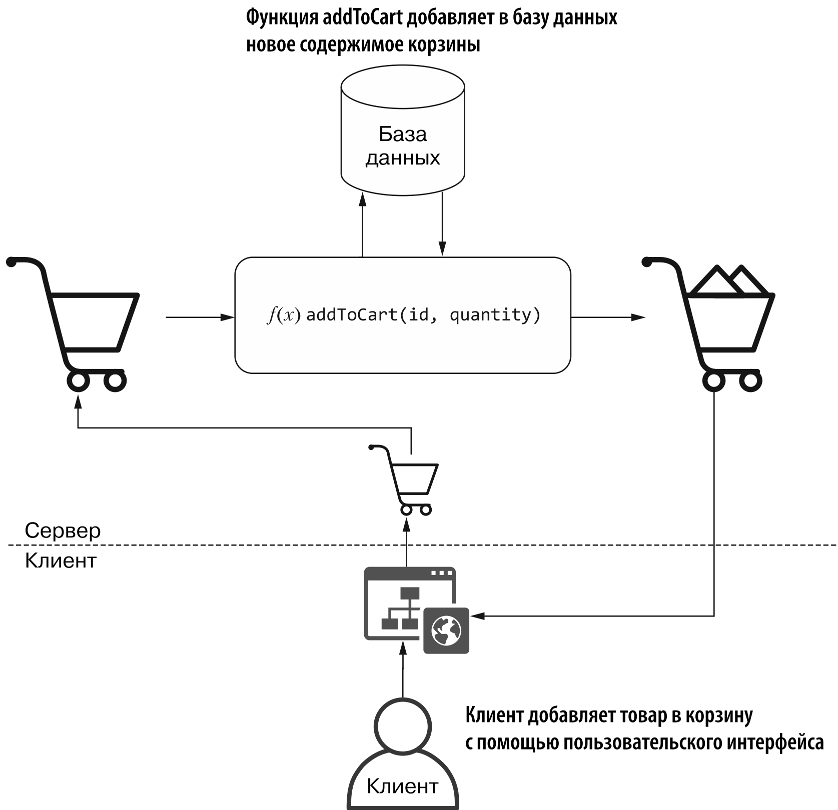

Рис. 1.4. Желаемая последовательность действий для добавления в корзину только доступных товаров

Эта ошибка еще критичнее, поэтому следует проявить повышенную бдительность. Для большей уверенности в тесте его можно написать еще до фактического исправления ошибки, чтобы увидеть, когда он работает не так, как вы того ожидаете.

Тест можно считать полезным только в том случае, если он проваливается, когда ваше приложение не работает.

Данный тест работает по тому же принципу, что и предыдущий: заменяет пользователя программным кодом и имитирует его действия. Разница в том, что здесь необходимо предусмотреть дополнительный шаг для удаления всех миндальных пирожных из списка имеющихся в наличии. Тест должен подготовить подходящие условия *и* смоделировать действия, которые приводят к проявлению ошибки (рис. 1.5).

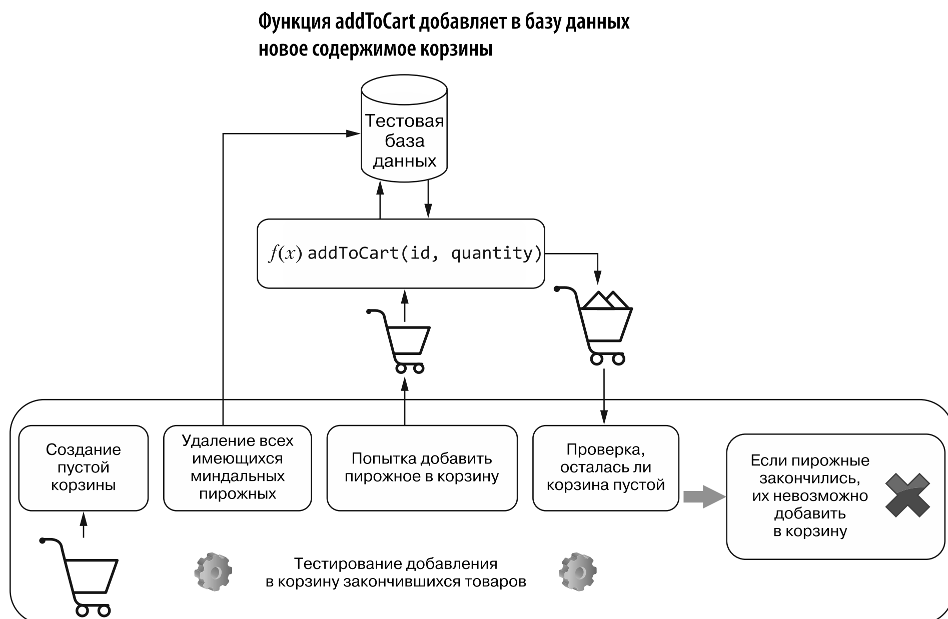

Рис. 1.5. Необходимые шаги теста, проверяющего, можно ли добавить в корзину товары, которые уже закончились

Наличие такого теста помогает быстрее устранить дефект. При внесении любого изменения тест будет сигнализировать о том, исправлена ли ошибка. Вам не нужно самостоятельно заходить в базу данных, удалять все миндальные пирожные, открывать сайт и пытаться добавить их в свою корзину: тест сделает это намного быстрее.

Поскольку вы уже написали тест для проверки возможности добавления покупателем нескольких единиц товара в корзину, он предупредит вас при повторном возникновении ошибки из-за внесенных исправлений. Тесты предоставляют быструю обратную связь и укрепляют вашу уверенность в том, что ПО действительно работает.

Но должен предупредить: автоматизированные тесты не гарантируют создания работающего ПО. **Тесты не могут доказать, что ПО работает, — они могут доказать только обратное.** Если бы при добавлении в корзину 10 001 миндального пирожного наличие товара по-прежнему игнорировалось, вы бы могли об этом узнать только после проверки этого конкретного ввода.

Тесты подобны экспериментам. Свои ожидания от того, как должно работать программное обеспечение, вы кодируете в тесты и хотите верить, что если

ранее они отрабатывали без заминки, то и дальше приложение будет вести себя подобным образом. Но это не всегда так. Чем больше у вас тестов, тем лучше они имитируют поведение настоящих пользователей, тем больше гарантий вы получаете.

Автоматизированные тесты при этом не избавляют от необходимости ручного тестирования. Проверка кода от имени конечного пользователя и проведение исследовательского тестирования по-прежнему незаменимы. Поскольку книга ориентирована на разработчиков ПО, а не на тестировщиков, в контексте этой главы я буду называть *ненужный* процесс ручного тестирования, который часто выполняется во время разработки, просто *ручным тестированием*.

1.2. ПОЧЕМУ АВТОМАТИЗИРОВАННЫЕ ТЕСТЫ ВАЖНЫ

Такие тесты важны тем, что дают быструю и безотказную обратную связь. Мы детально обсудим, как своевременный и точный отклик улучшает процесс разработки ПО, делая его единообразным и предсказуемым. Это позволяет легко воспроизводить проблемы и документировать тестовые случаи, что упрощает совместную работу внутри команды (или разных команд) и сокращает время, необходимое для создания высококачественных программных продуктов.

1.2.1. Предсказуемость

Наличие предсказуемого процесса разработки позволяет избежать ситуации, когда реализация какой-то функциональной возможности или исправление ошибки приводят к непредвиденному поведению. Уменьшение количества сюрпризов, возникающих в ходе разработки, облегчает оценку стоящих перед вами задач и позволяет не так часто переписывать код.

Чтобы вручную проверить работоспособность всего программного продукта, требуется много времени, и это чревато ошибками. Тесты оптимизируют процесс, сокращая время получения отклика о коде, над которым вы работаете, и ускоряя тем самым исправление ошибок. **Чем меньше задержка между написанием кода и получением обратной связи, тем более предсказуемой становится разработка.**

Чтобы проиллюстрировать, как именно тесты делают разработку более предсказуемой, представим, что дядя Луис попросил вас дать клиентам возможность отслеживать состояние заказов. Это позволило бы ему уделять больше времени приготовлению сладостей, вместо того чтобы отвечать на звонки и уверять

клиентов в том, что их заказ будет доставлен вовремя. Луис увлекается творожными десертами, а не телефонными переговорами.

Если бы вы реализовали функцию отслеживания без автоматизированных тестов, вам пришлось бы вручную пройти весь процесс покупки и убедиться в том, что он работает (рис. 1.6). И при этом каждый раз вам нужно было бы не только перезагружать сервер, но также очищать базы данных, чтобы гарантировать их согласованность, открывать браузер, добавлять товары в корзину, указывать время доставки, проходить процедуру оплаты. И только после всего этого у вас была бы возможность проверить, отслеживается ли ваш заказ.

Но, чтобы протестировать эту функцию даже вручную, она должна быть доступна на сайте. Вам нужно написать для нее интерфейс и довольно существенную часть серверного кода, с которым будет взаимодействовать клиент.

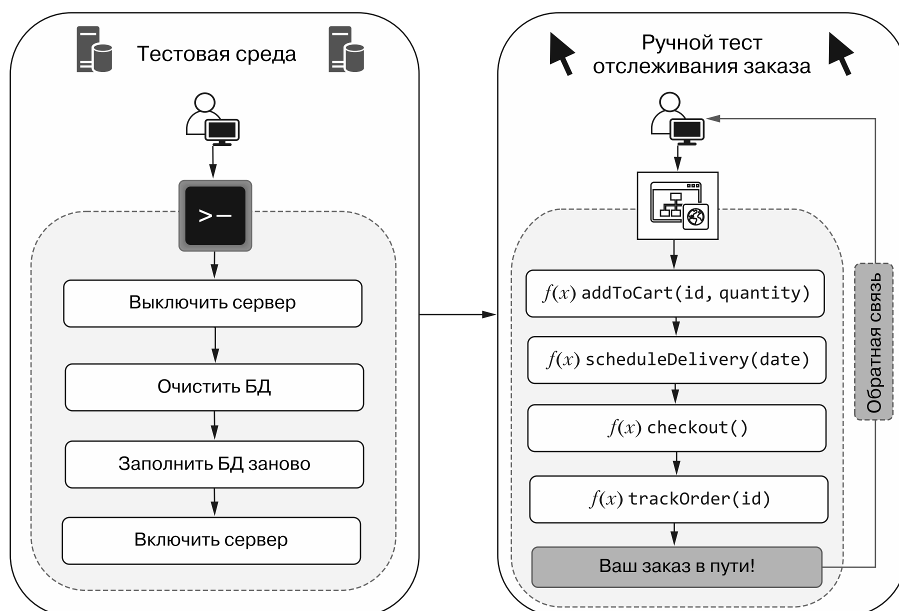

Рис. 1.6. Этапы проверки отслеживания заказа

Ввиду отсутствия автоматизированных тестов вам придется написать довольно много кода, прежде чем вы сможете проверить, работает ли данная функция. Если внесение каждого изменения сопряжено с длительным и утомительным процессом, это подталкивает к написанию более крупных фрагментов кода. Что, в свою очередь, задерживает получение обратной связи и может привести даже к тому, что вы получите ее слишком поздно. К тому же, когда увеличивается

объем написанного кода, увеличивается и количество потенциальных ошибок. В какой из тысяч новых строчек кода скрывается ошибка, которую вы только что наблюдали?

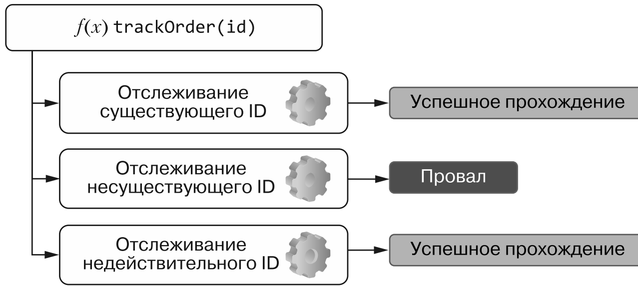

Рис. 1.7. Тесты для функции trackOrder могут вызывать ее напрямую, поэтому вам не нужно трогать другие части приложения

Автоматизированный тест наподобие показанного на рис. 1.7 позволяет получить обратную связь после написания меньшего количества кода. Такие тесты могут вызывать функцию `trackOrder` напрямую, что позволяет убедиться в ее работоспособности без необходимости трогать другие части приложения.

Когда тест проваливается после написания десяти строк кода, вам нужно волноваться только об этих десяти строках. Даже если дефект находится в другом месте, вам будет намного проще определить, что спровоцировало нежелательное поведение.

Ситуация может усугубиться, если вы нарушите работу других частей приложения. Обнаружив ошибки в процедуре оплаты, вам придется проверить, как на нее повлияли ваши изменения. Чем больше изменений вы вносите, тем сложнее определить, в чем проблема.

Автоматизированные тесты, такие как на рис. 1.8, могут сразу же оповестить о проблеме, что поможет вам принять необходимые меры. Частое выполнение тестов даст точную информацию о том, какие участки приложения сломались, непосредственно в момент поломки. Помните, что **чем меньше времени вам придется ждать обратной связи после написания кода, тем более предсказуемым будет процесс разработки**.

Я часто вижу, как разработчикам приходится выбрасывать свой код из-за того, что в него было внесено слишком много изменений за один раз. Когда изменения приводят к поломке множества разных частей приложения, разработчики не знают, за что хвататься. Им проще начать с чистого листа, чем разбираться с беспорядком, который они устроили. Сколько раз *вы* оказывались в подобной ситуации?

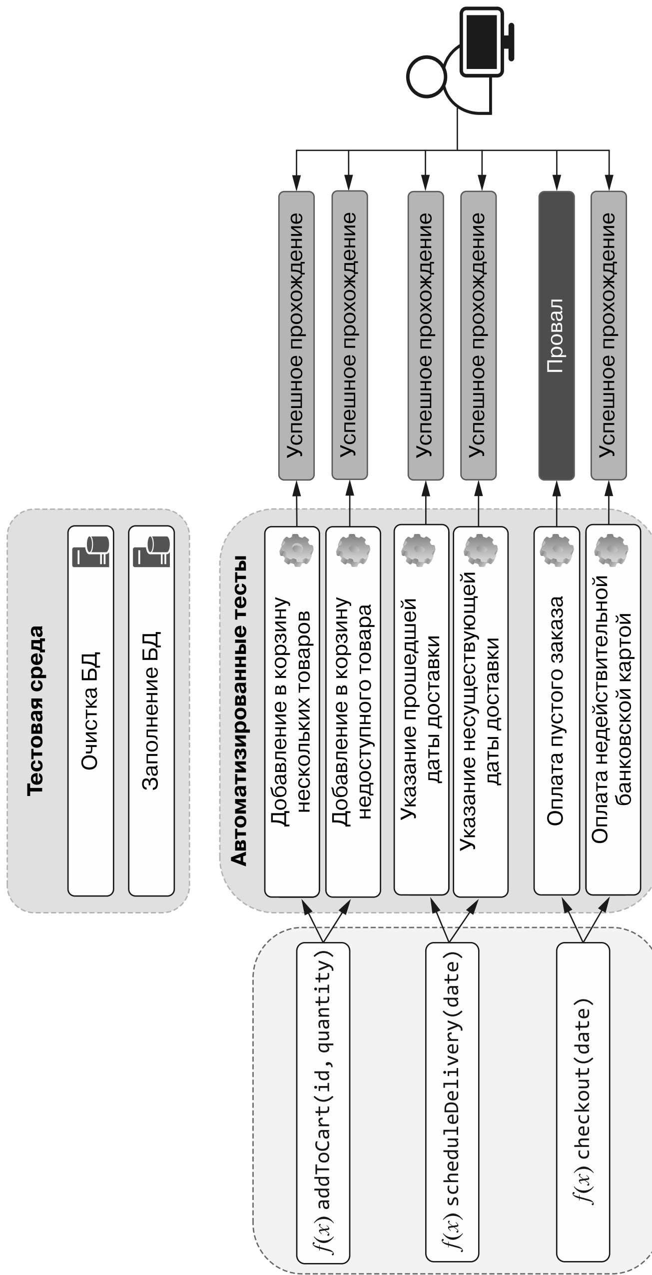

Рис. 1.8. Автоматизированные тесты могут проверять участки вашего кода по отдельности, предоставляя точные сведения о том, что сломалось, непосредственно в момент поломки

1.2.2. Воспроизводимость

Чем больше шагов у задачи, тем выше вероятность того, что человек допустит ошибки при их выполнении. Автоматизированные тесты упрощают и ускоряют воспроизведение ошибок и их устранение.

Перед тем как отслеживать свой заказ, покупатель должен пройти через несколько этапов: добавить товар в корзину, выбрать дату доставки и оплатить. Чтобы протестировать приложение и убедиться в том, что оно будет работать для клиентов, вы должны проделать то же самое. Процесс достаточно длинный, чреватый ошибками, с большим количеством вариантов выполнения каждого шага. Автоматизированные тесты позволяют гарантировать, что эти шаги будут выполняться строго и безошибочно.

Представьте, что при тестировании приложения вы выявили ошибку: например, клиент может перейти к оплате с пустой корзиной или недействительной банковской картой. Для обнаружения этих ошибок вам придется выполнить последовательность действий вручную.

Чтобы предотвратить повторение ошибок, вы должны в точности воспроизвести те шаги, которые к ним привели. Если список тестовых случаев становится слишком длинным или у вас получается слишком много шагов, человеческий фактор даст о себе знать. Ошибки непременно закрадутся в ваш код (рис. 1.9), если только у вас нет перечня действий, которому вы всегда неукоснительно следуете.

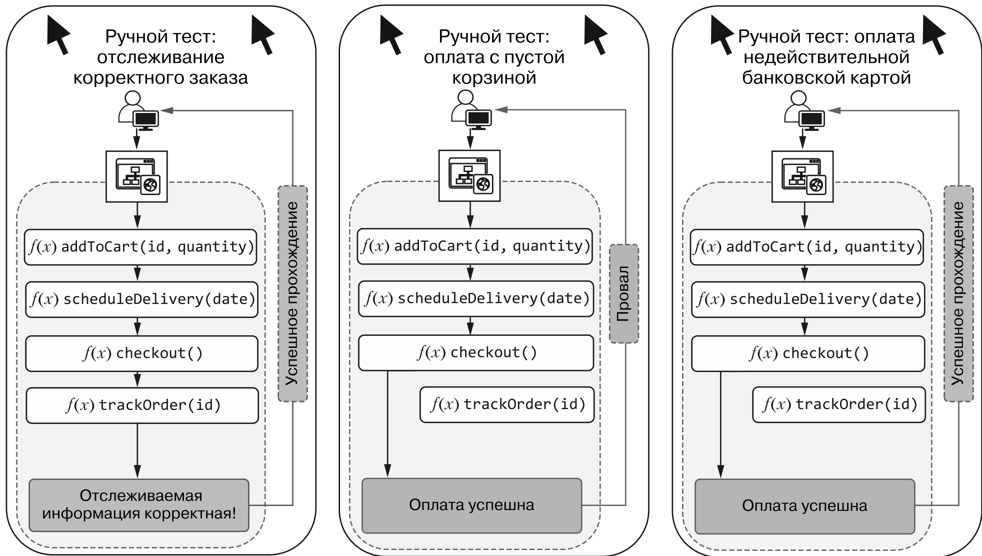

Рис. 1.9. Шаги, которые нужно выполнять при тестировании каждой функциональной возможности

Вряд ли вы забудете проверить процесс заказа торта, но что, если тортов будет `-1` или `NaN`? Людям свойственно что-то упускать из виду и ошибаться, поэтому программное обеспечение и сбоит. Человек должен заниматься теми вещами, которые у него хорошо получаются, а выполнение рутинных повторяющихся задач в их число не входит.

Даже решив вести учет всех тестовых случаев, вы должны будете прилагать дополнительные усилия для поддержания списка в актуальном состоянии. А если когда-нибудь вы забудете его обновить и с приложением произойдет что-то не отмеченное в списке, то чья это будет вина — приложения или документации?

Автоматизированные тесты выполняют одни и те же действия каждый раз, когда вы их запускаете. Компьютер, выполняющий тесты, не пропустит никаких шагов и не совершит ошибки.

1.2.3. Совместная работа

Все, кто пробовал пирог баноффи в исполнении Луиса, знают, что Луис вполне мог бы претендовать на победу в телешоу Great British Bake Off. Если вы создадите правильное ПО, однажды, возможно, кондитерские Луиса откроются по всему миру, от Сан-Франциско до Санкт-Петербурга. Но в этом случае одного разработчика будет недостаточно.

Если вы наймете других разработчиков для совместной работы, у вас внезапно появятся и новые разнообразные проблемы. Представьте, что вы, к примеру, реализуете новую систему скидок, а Элис в это время занимается механизмом генерации купонов. Как быть, если изменения, внесенные вами в процедуру оплаты, сделают невозможным применение купонов к заказам? Иными словами, как гарантировать, что ваша работа не вступит в конфликт с работой Элис и наоборот?

Если первой свои функции в кодовую базу включит Элис, вам придется спрашивать, как протестировать ее часть работы, чтобы ничего не сломать. Таким образом, включение *вашего* кода отнимет время как у *вас*, так и у *Элис*.

Но усилиями, которые вы с Элис приложили для ручного тестирования изменений, дело не ограничится. Вам еще придется вместе поработать над интеграцией вашего и ее кода, а потом и проверить, как эти изменения интегрировались (рис. 1.10).

Этот процесс не только трудоемкий, но и подверженный ошибкам. Вы должны помнить все шаги и все крайние случаи, которые нужно протестировать как в вашем коде, так и в коде Элис. Но мало того, что шаги нужно помнить, — их еще необходимо точно выполнить.

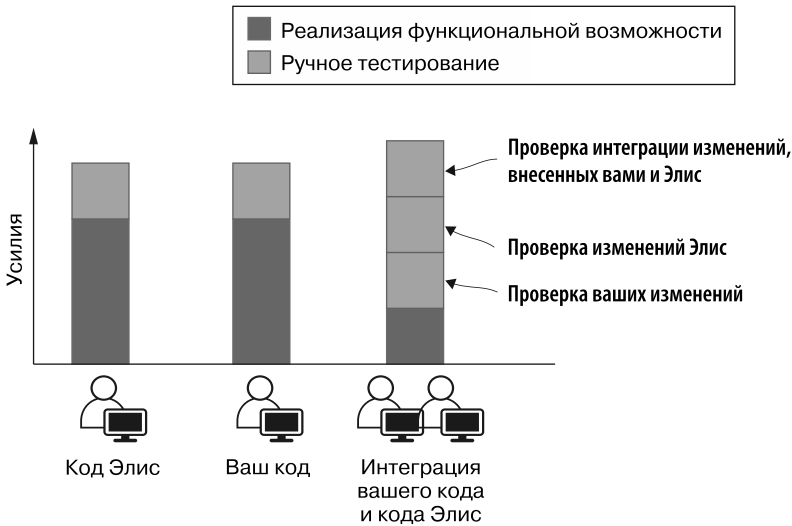

Рис. 1.10. Ручное тестирование требует проверки изменений на каждом этапе процесса разработки

Когда программист добавляет для своих функций автоматизированные тесты, от этого выигрывают все. Если для кода Элис есть тесты, вам не нужно просить ее, чтобы она проверяла свои изменения. Во время слияния обеих частей кода можно просто запустить имеющиеся автоматизированные тесты, вместо того чтобы снова проходить через весь процесс ручного тестирования.

Даже если ваши изменения основаны на изменениях, внесенных Элис, тесты будут служить актуальной документацией, на которую можно ориентироваться в дальнейшей работе. Хорошо написанные тесты — это лучшая документация, которая только может быть у разработчика. Они должны успешно выполняться, поэтому их актуальность гарантирована. Если вы все равно будете писать техническую документацию, почему бы вместо нее не создать тест?

При интеграции вашего кода с тем, что написала Элис, вам также нужно будет добавить автоматизированные тесты, проверяющие эту интеграцию. Новые тесты будут использоваться последующими разработчиками при реализации функциональных возможностей, имеющих отношение к вашему коду, и это сэкономит им время. Написание тестов при внесении каких-либо изменений порождает эффективный цикл взаимодействия, когда каждый разработчик помогает тем, кто будет работать с кодовой базой после него (рис. 1.11).

Этот подход *оптимизирует* совместную работу, но не устраняет необходимость в общении, которое лежит в основе любого успешного проекта. Автоматизированные тесты в значительной мере улучшают процесс взаимодействия между

разработчиками и в сочетании с другими методиками, такими как обзор кода, делают его еще более эффективным.

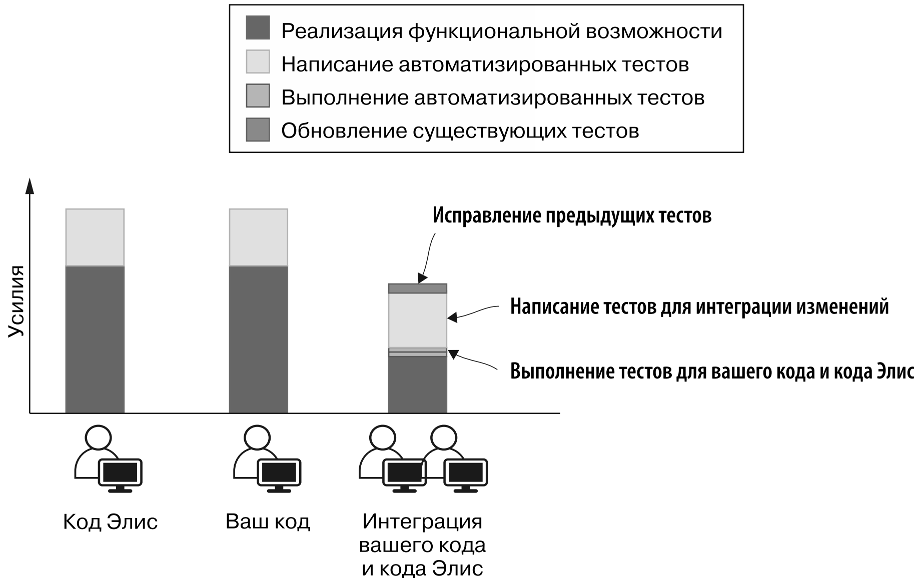

Рис. 1.11. Усилия, необходимые для проверки изменений на каждом этапе процесса разработки в случае наличия автоматизированных тестов

Одна из самых сложных задач в сфере проектирования ПО — организация эффективной совместной работы нескольких разработчиков. Тесты в этом деле являются крайне полезным инструментом.

1.2.4. Скорость

Дяде Луису все равно, какой язык вы используете, и уж точно его не заботит, сколько тестов вы написали. Он хочет продавать выпечку, торты и прочие сладкие шедевры. Его также заботит доход. Если для удовлетворения клиентов и повышения выручки требуются новые функции, он хотел бы, чтобы вы реализовали их как можно скорее. Но с одной оговоркой: эти функции должны работать.

С коммерческой точки зрения важны не сами тесты, а скорость и корректность кода. В предыдущих разделах мы говорили о том, как тесты улучшают процесс разработки, делая его более предсказуемым, воспроизводимым и дружественным к совместной работе. Однако, по большому счету, все эти преимущества ценны лишь потому, что с их помощью можно получить более качественное ПО за меньшее время.

Когда написание кода, его проверка на отсутствие определенных дефектов и его интеграция с кодом других разработчиков требуют меньше времени, это делает вашу компанию более успешной. То же самое можно сказать о предотвращении регрессий и повышении безопасности процесса развертывания.

Тесты *требуют* определенных затрат, ведь для их написания нужно время. Но они все равно нужны, так как их преимущества существенно перевешивают недостатки.

Изначально написание теста может занять много времени, даже больше, чем тестирование вручную. Но чем чаще он выполняется, тем больше пользы он приносит. Например, ручная проверка может занять одну минуту, а написание автоматизированного теста — пять минут, но после пятого выполнения этот тест себя окупит. И поверьте мне, выполнений у теста будет куда больше пяти.

Если сравнивать с ручным тестированием, которое всегда занимает одно и то же время (или больше), время и усилия, необходимые для выполнения автоматизированного теста, сводятся почти к нулю. Объем работы при ручном тестировании растет намного быстрее. Это различие в усилиях, необходимых для написания автоматизированных тестов и проведения тестирования вручную, проиллюстрировано на рис. 1.12.

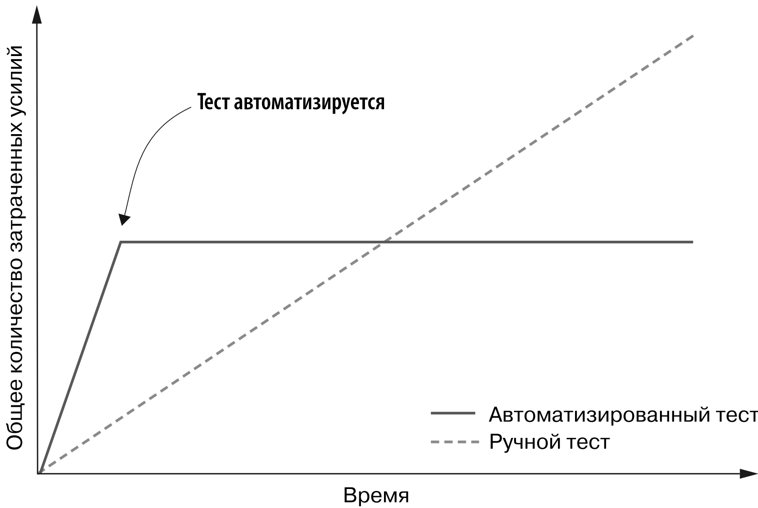

Рис. 1.12. Ручное и автоматизированное тестирование: сравнение прилагаемых усилий с течением времени

Написание тестов подобно покупке акций. В самом начале вы можете заплатить высокую цену, но потом будете получать дивиденды на протяжении долгого времени. Как и в финансовой области, дело, в которое лучше инвестировать