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.

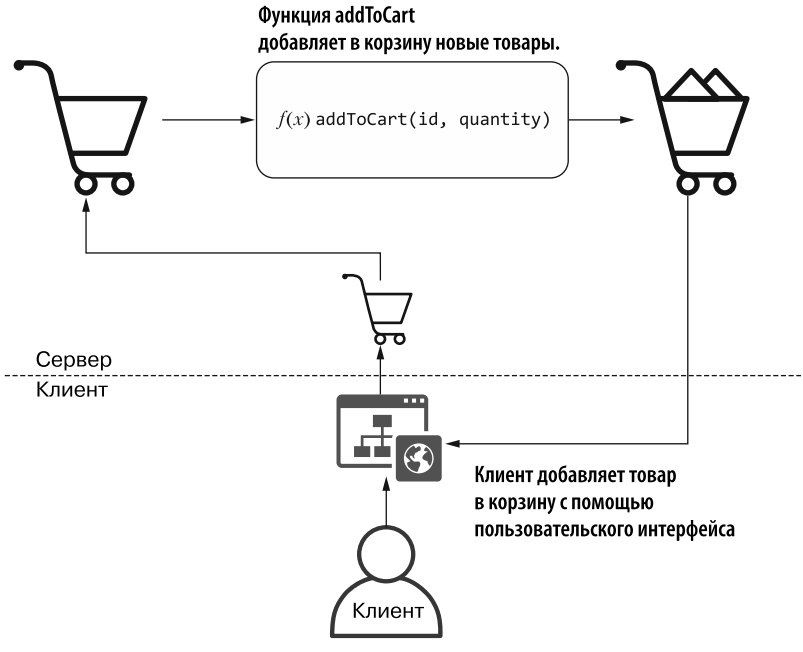

Рис. 1.1. Последовательность действий при создании заказа

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).

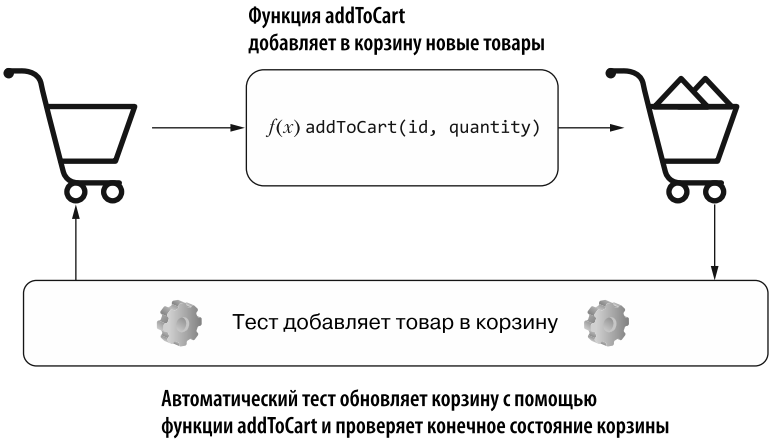

Рис. 1.2. Последовательность действий для тестирования addToCart

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.

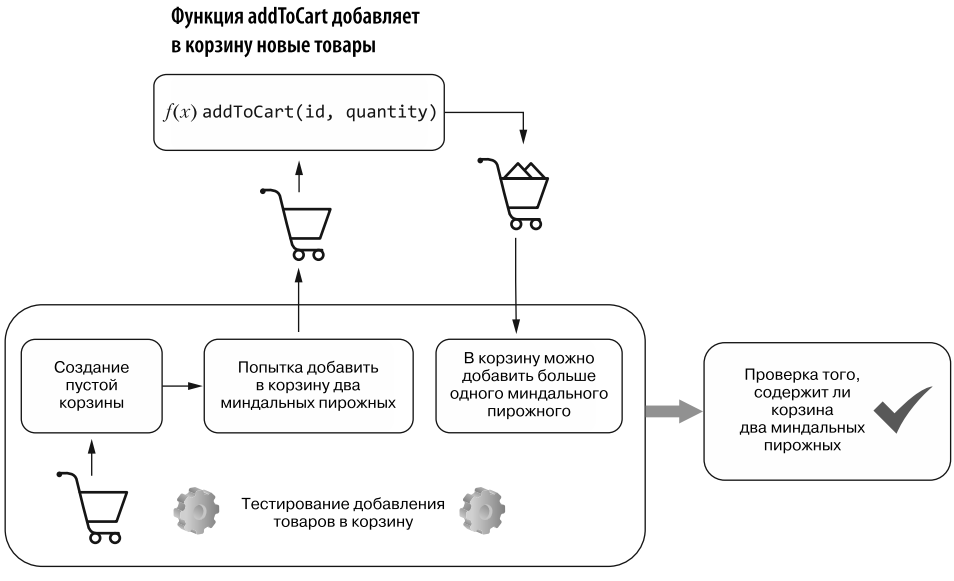

Рис. 1.3. Последовательность действий для теста, который проверяет, можно ли добавить в корзину несколько миндальных пирожных

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.

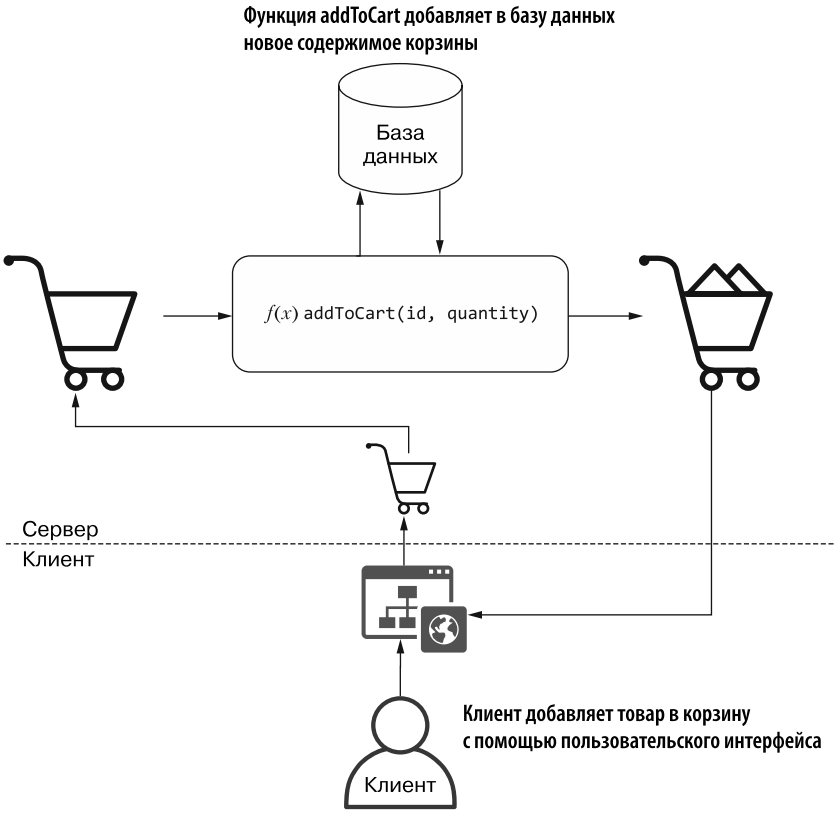

Рис. 1.4. Желаемая последовательность действий для добавления в корзину только доступных товаров

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).

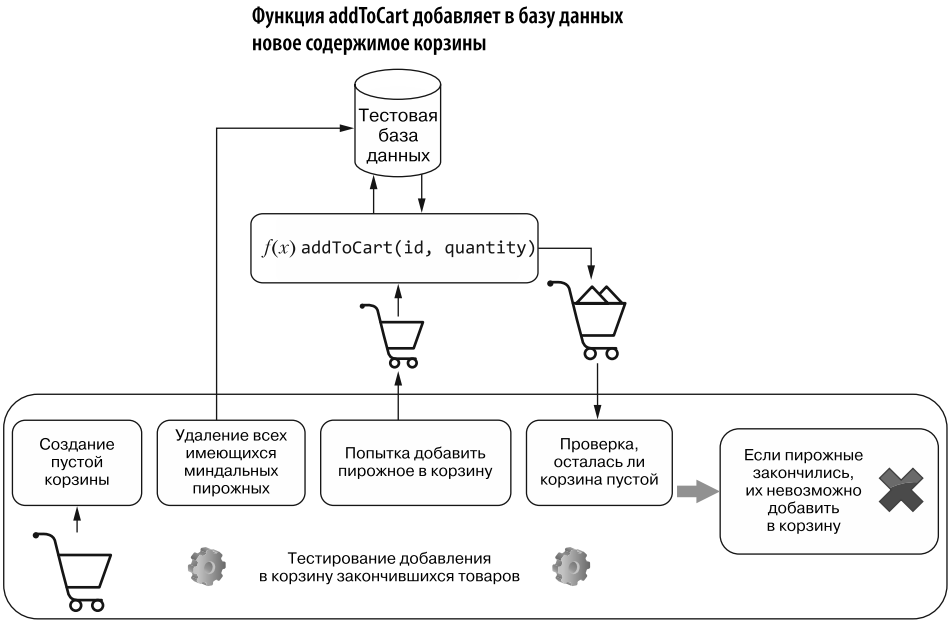

Рис. 1.5. Необходимые шаги теста, проверяющего, можно ли добавить в корзину товары, которые уже закончились

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.

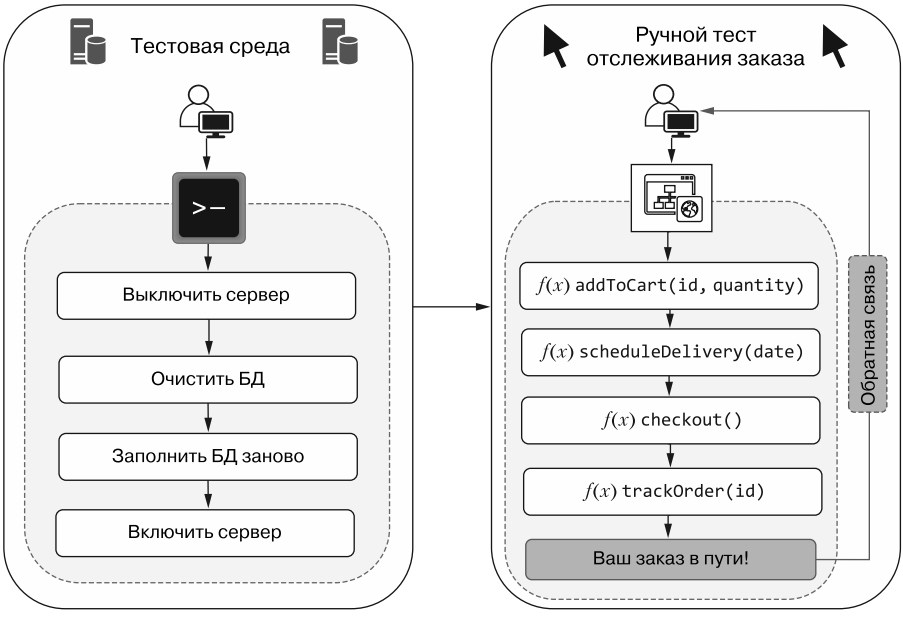

Рис. 1.6. Этапы проверки отслеживания заказа

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?

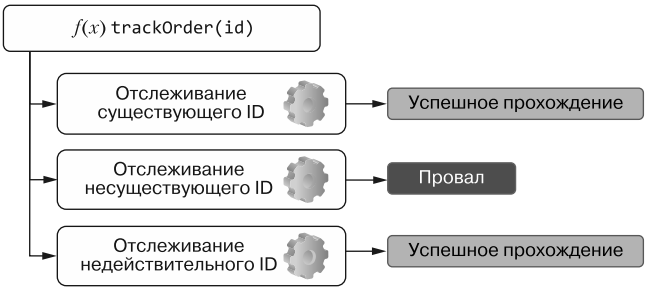

Рис. 1.7. Тесты для функции trackOrder могут вызывать ее напрямую, поэтому вам не нужно трогать другие части приложения

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?

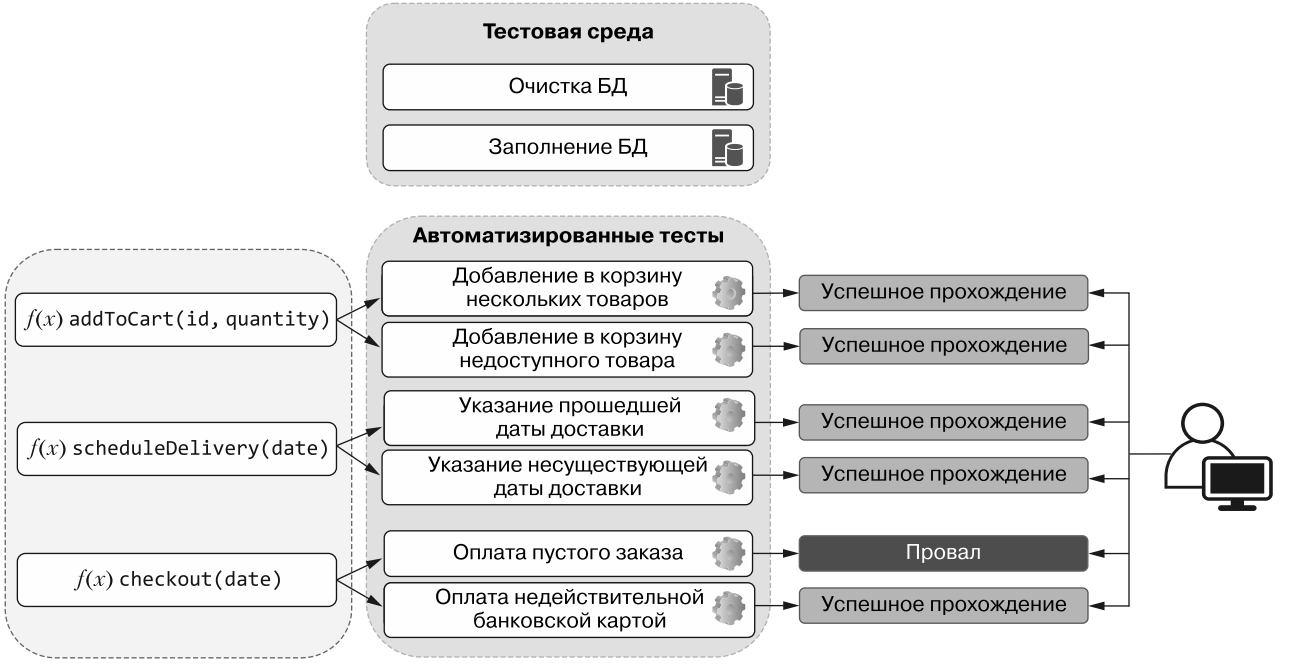

Рис. 1.8. Автоматизированные тесты могут проверять участки вашего кода по отдельности, предоставляя точные сведения о том, что сломалось, непосредственно в момент поломки