Часть IV
Автоматизация

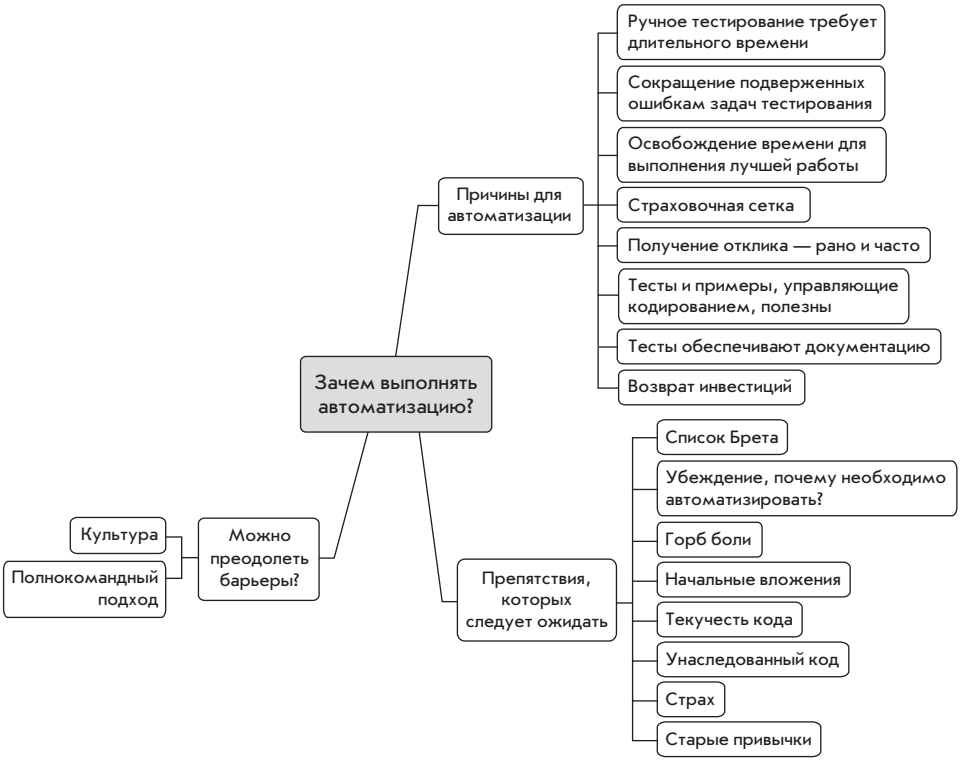
Автоматизация тестов — основная практика гибкой методологии. Гибкие проекты зависят от автоматизации. Достаточно хорошая автоматизация позволяет команде часто выдавать высококачественный код. Она обеспечивает каркас, благодаря которому команда наращивает скорость при соблюдении высоких стандартов качества. Система управления версиями исходного кода, автоматизированные сборки и комплекты тестов, развертывание, мониторинг и широкое разнообразие сценариев и инструментов избавляют от рутины, гарантируют надежность и позволяют команде постоянно выполнять свою работу наилучшим образом.

Тема автоматизации также является весьма обширной. Она включает такие задачи, как написание простых сценариев оболочки, установка свойств сеансов и создание устойчивых автоматизированных тестов. Диапазон и численность инструментов автоматизации тестирования растет экспоненциально по мере того, как открываются лучшие способы производства программного обеспечения. К счастью, также быстро растет и количество отличных книг, посвященных автоматизации тестирования.

Наша книга сосредоточена на роли тестировщика в гибкой разработке. Поскольку автоматизация — ключ к успешной гибкой разработке, нам следует поговорить о ней, хотя мы и не сможем раскрыть абсолютно все аспекты этой темы. Прежде всего, мы хотим объяснить, почему вы, как тестировщик, должны стремиться к автоматизации, и как вы вместе с командой можете преодолеть многие препятствия, которые мешают усилиям в этом направлении. В этой части будет показано, как применять гибкие ценности, принципы и приемы для развития практической стратегии автоматизации, преодоления барьеров и выработки стремления к автоматизации тестов.

Глава 13

Причины и препятствия к внедрению автоматизации тестов



Зачем мы автоматизируем тестирование, процесс сборки, развертывание и прочие задачи? Гибкие команды сосредоточены на том, чтобы постоянно иметь работающее программное обеспечение, что позволяет им выдавать выпуски с нужной частотой. Достижение этой цели требует постоянного тестирования. В настоящей главе мы рассмотрим причины, по которым мы стремимся к автоматизации, и сложности, которые препятствуют нам на этом пути.

Зачем выполнять автоматизацию?

Для проведения автоматизации существует множество причин, помимо той, что была упомянута: автоматизация нужна, чтобы достичь успеха в применении гибкой методики. Ниже приведен наш список причин.

- Ручное тестирование требует длительного времени.
- Ручной процесс подвержен ошибкам.
- Автоматизация освобождает людей для выполнения лучшей работы.
- Автоматизированные регрессивные тесты предоставляют "страховочную сетку".
- Автоматизированные тесты обеспечивают ранний и частый отклик.
- Тесты и примеры, управляющие кодированием, позволяют выполнить больше работы.
- Тесты обеспечивают документацию.
- Автоматизация может обеспечить хороший возврат инвестиций.

Рассмотрим каждую из этих причин более подробно.

Ручное тестирование требует длительного времени

Самая главная причина, по которой команды стремятся к автоматизации, состоит в том, что ручное тестирование просто отнимает очень много времени. По мере роста приложения время на полное его тестирование также возрастает, иногда экспоненциально, в зависимости от сложности тестируемого приложения.

Гибкие команды в состоянии поставлять работоспособное программное обеспечение в конце каждой итерации благодаря тому, что каждый день имеют это работоспособное программное обеспечение. Запуск полного комплекта регрессивных тестов — по крайней мере, ежедневно — обязательное условие, и вы просто не в состоянии выполнить все эти тесты вручную. Если не предусмотрено никакой автоматизации, то регрессивные тесты придется выполнять вручную.

Выполнение регрессивных тестов вручную отнимает все больше и больше времени. Чтобы тестирование не отставало от кодирования, либо программистам придется помогать в проведении ручных тестов, либо команде придется нанимать дополнительных тестировщиков. В любом случае неизбежно будет возрастать технический долг и чувство неудовлетворенности.

Если код не проходит комплект автоматизированных регрессивных тестов уровня модулей, тестировщики, скорее всего, будут тратить большую часть своего времени на исследования, попытки воспроизведения и документирования этих простых ошибок, и гораздо меньше времени уйдет на поиск потенциально опасных серьезных ошибок уровня системы. Вдобавок, поскольку команда не придерживается принципа разработки "сначала тесты", сам дизайн кода будет намного менее тестируемым и может не обеспечивать функциональности, нужной заказчику.

Ручное тестирование множества разнообразных сценариев может занять уйму времени, особенно, если приходится вручную вводить данные в пользовательском интерфейсе. Подготовка данных для множества различных сценариев может стать неподъемной задачей, если нет автоматического способа ускорить ее. В результате может быть протестировано лишь ограниченное количество сценариев, и важные дефекты останутся незамеченными.

Ручной процесс подвержен ошибкам

Повторяющееся ручное тестирование, особенно по написанным сценариям, очень быстро надоедает. Это прямой путь к ошибкам и пропуску даже простейших дефектов. Люди, выполняющие ручное тестирование, пропускают шаги, а то и целые тесты. Если перед командой стоят жесткие сроки, возникает соблазн "срезать углы", в результате чего серьезные проблемы остаются незамеченными.

Поскольку ручное тестирование проходит медленно, вы вполне можете заниматься им и в последнюю ночь перед завершением итерации. Сколько ошибок вы отловите в таких условиях?

Автоматизированная сборка, развертывание, управление версиями и мониторинг также позволяют сократить степень риска и сделать процесс разработки более согласованным. Автоматизируя эти сценарные тесты, вы исключаете возможность ошибок, поскольку тест выполняется каждый раз в точности одинаково.

Принцип "собрать однажды, развернуть многократно" (build once, deploy to many) — это исполнившаяся мечта тестировщика. Автоматизация процесса сборки и развертывания позволяет точно знать, что именно тестируется в каждой заданной среде.

Автоматизация освобождает людей для выполнения лучшей работы

Написание кода по принципу "сначала тесты" позволяет программистам понять требования и проектировать код в соответствии с ними. Запуск всех модульных и регрессивных тестов в рамках непрерывной сборки означает, что больше времени можно уделить исследовательскому тестированию. Автоматизация подготовки исследовательских тестов означает, что вы сможете уделить еще больше внимания испытанию потенциально слабых частей системы. Поскольку вы не тратите время на выполнение утомительных ручных сценариев, ваша энергия сохраняется для более сложной и творческой работы, обдумывания различных сценариев и изучения работы приложения.

Если постоянно думать о том, как автоматизировать тесты для исправлений или новых средств кода, то, скорее всего, мы будем думать также о тестируемости и качестве дизайна, а не о быстрых черновых решениях, которые могут оказаться хрупкими. Это означает лучший код и лучшие тесты.

Автоматизированные тесты могут в действительности помочь обеспечить согласованность по всему приложению.

История от Джанет

Джейсон (один из моих приятелей, работающий тестировщиком) и я занимались сценариями автоматизации графического интерфейса пользователя с помощью Ruby and Watir. Мы добавили константы для имен кнопок, используемые в тестах. Мы быстро поняли, что кнопки на каждой странице были названы несогласованно. Мы смогли исправить это и устранить несогласованность очень быстро, а также выработать общие соглашения по именованию.

📖 За более подробной информацией о Ruby and Watir обращайтесь в главы 9, 12 и 14.

С помощью книг вроде *Pragmatic Project Automation* [13] можно научиться автоматизировать ежедневные рутинные операции разработки и освободить команду для более важной деятельности, такой как исследовательское тестирование.

Обеспечение лучшей работы для тестировщиков

Крис Мак-Махон (Chris McMahon) так описывает преимущества, которые принесла автоматизация регрессивных тестов, в списке рассылки за ноябрь 2007 г., посвященной гибкому тестированию:

> "Объем наших автоматизированных регрессивных тестов с апреля 2007 г. вырос на 500%. Это позволило сосредоточить внимание на более интересном тестировании, которое может выполнять только человек".
>
> Дальше Крис поясняет: "Теперь, когда степень автоматизации высока, у нас появилось свободное время, чтобы подумать о том, какие именно тесты, проводимые человеком, могут понадобиться. Для любого нетривиального тестирования мы сначала проводим сеанс мозгового штурма, после чего приступаем к его выполнению". Обычно Крис и его коллеги объединяют для этого двух тестировщиков или тестировщика с разработчиком. Иногда тестировщик генерирует идеи и предлагает их на обсуждение через различные онлайновые средства. Крис замечает: "Мы почти всегда вырабатываем хорошие идеи относительно тестов, работая парами — такие, что не могли бы появиться у каждого члена команды по отдельности".
>
> Говоря об их частых выпусках и существенных средствах, Крис заявляет: "Благодаря хорошей автоматизации тестирования, у нас есть время позаботиться о том, чтобы весь продукт был привлекательным и функциональным для реальных пользователей. Без автоматизации тестирование этого продукта было бы утомительным. А так при подготовке каждого выпуска наши тестировщики имеют возможность заниматься важной и интересной работой".

Мы согласны с Крисом, что наиболее впечатляющая часть автоматизации тестирования состоит в том, что оно расширяет возможности совершенствования продукта через инновационное исследовательское тестирование.

Проекты бывают успешными тогда, когда хорошие люди имеют возможность выполнять наилучшую работу, причем максимально эффективным образом. Автоматизация тестирования как раз обеспечивает такую возможность Важнейшим компонентом всего этого являются автоматизированные регрессивные тесты, которые обнаруживают изменения в существующей функциональности и дают немедленный отклик.

Автоматизированные регрессивные тесты предоставляют "страховочную сетку"

Большинству практиков, связанных с разработкой программного обеспечения в течение нескольких лет, знакомо ощущение "волосы дыбом", когда они сталкиваются с необходимостью исправления ошибки или реализации нового средства в плохо спроектированном коде, который не охвачен автоматизированными тестами. Это все равно, что пытаться надуть дырявый воздушный шар. Получится?

Знание о том, что код имеет хорошее покрытие автоматизированными регрессивными тестами, дает замечательное чувство уверенности. Конечно, любые изменения могут дать неожиданный эффект, но мы узнаем об этом в течение каких-то минут, если речь идет об уровне модулей, либо часов, если это касается более высокого уровня функциональности. Проведение изменений в стиле "сначала тесты" означает, что вы думаете о поведении изменения еще перед написанием кода и теста для его проверки, что и обеспечивает эту уверенность.

История от Джанет

Недавно я беседовала с одним из тестировщиков в моей команде, который спросил о ценности автоматизированных тестов. Мой первый ответ был: “Это — страховочная сетка для команды”. Однако он оспорил это утверждение: “Разве в таком случае мы не полагаемся на тесты вместо того, чтобы устранить исходную причину проблемы?”

Это заставило меня задуматься. Он был прав в одном отношении: если мы успокоимся в отношении проблем тестирования и полностью будем полагаться на автоматизированные тесты в том, что касается обнаружения ошибок, с последующим решением их лишь в той степени, которая достаточна для успешного прогона тестов, то, очевидно, оказываем себе плохую услугу.

Однако если мы используем тесты для идентификации проблемных областей и либо корректно их исправляем, либо при необходимости выполняем рефакторинг, значит, мы правильно применяем “страховочную сетку” автоматизации. Автоматизация важна для успеха гибкого проекта, особенно по мере роста разрабатываемого приложения.

Не имея “страховочной сетки” в виде автоматизированного комплекта тестов, программисты могут начать воспринимать в качестве такой сетки самих тестировщиков. Легко представить ситуацию, когда Джо Программист думает так: “Мне надо бы вернуться и добавить некоторые автоматизированные модульные тесты для formalEmployeeInfo, но я знаю, что Сьюзи Тестировщик собирается проверить каждую ее страницу вручную. Если она увидит, что там что-то не так, я просто продублирую ее усилия”.

Это замечательно, если программист настолько доверяет таланту тестировщика, но в этом случае Джо ступает на скользкую дорожку. Если он не автоматизирует эти модульные тесты, какие еще тесты он может пропустить? Сьюзи может просто не справиться с просмотром всех этих страниц.

Команды, которые обеспечили себе хорошее покрытие кода автоматизированными регрессивными тестами, могут без страха вносить изменения в код. Им не нужно задавать себе вопрос: “Если я изменю модуль formalEmployeeInfo, не разрушит ли это что-нибудь в пользовательском интерфейсе?”. Тесты сразу покажут им, нарушилось ли что-то в результате изменений. Поэтому они могут работать гораздо быстрее, чем команды, полагающиеся исключительно на ручное тестирование.

Автоматизированные тесты обеспечивают ранний и частый отклик

После того, как автоматизированный тест некоторой части функциональности прошел успешно, он и будет проходить успешно до тех пор, пока эта функциональность не будет намеренно изменена. Когда мы планируем изменения в приложении, то параллельно изменяем тесты, чтобы они соответствовали изменениям основного кода. Если автоматизированный тест неожиданно дает сбой, регрессивный дефект может быть вызван изменением кода. Запуск комплекта автоматизированных тестов при каждой фиксации нового кода в системе управления версиями помогает гарантировать быстрый перехват регрессивных ошибок. Быстрый отклик означает, что изменение остается свежим в сознании некоторого программиста, и потому он может быстро устранить неисправность, чем в том случае, когда она обнаруживается намного позже. Быстрое выявление ошибок означает, что их дешевле исправлять.

Автоматизированные тесты, запускаемые регулярно, часто служат детектором изменений. Они дают команде возможность узнать об изменениях, произошедших с момента последней успешной сборки. Например, не было ли каких-то отрицательных побочных эффектов с последней сборкой? Если автоматизированный комплект тестов обеспечивает хорошее покрытие, то он может легко выявить далеко идущие эффекты, которые никогда не обнаружат ручные тестировщики.

Если регрессивные тесты не автоматизированы, то, скорее всего, они не будут прогоняться при каждой итерации или каждый день. И тогда проблема немедленно возникает в конце "игры", когда команде приходится выполнять все регрессивные тесты. Ошибки, которые можно было бы выявить рано, выявляются поздно, и многие из преимуществ раннего тестирования теряются.

Тесты и примеры, управляющие кодированием, дают больше

В главе 7 мы говорили об использовании тестов и примеров для управления кодированием. Было сказано, насколько важно руководить кодированием с помощью как модульных, так и тестов заказчика. Мы также хотели подчеркнуть, что если эти тесты автоматизировать, то их ценность повышается по другой причине. Они становятся базой для очень строгого регрессивного комплекта.

История от Лайзы

После того, как моя команда овладела модульными тестами, рефакторингом, непрерывной интеграцией и другими приемами технологического плана, мы получили возможность перехвата регрессивных ошибок и неверно реализованной функциональности еще на фазе разработки.

Конечно, это не означало, что все проблемы были решены; мы все равно иногда упускали или неправильно понимали требования заказчика. Однако наличие автоматизированного каркаса позволило сосредоточиться на выполнении лучшей работы по фиксации требований в виде фронтальных тестов. У нас также появилось больше времени для проведения исследовательского тестирования. Со временем количество дефектов в наших разработках снизилось радикально, при этом клиенты были удовлетворены поставляемым им продуктом.

TDD и SDD (story test-driven development — разработка историй, управляемая тестами) заставляет команду сначала продумывать тесты. Во время плановых совещаний они говорят о тестах и лучших способах их создания. Они проектируют код таким образом, чтобы он проходил тесты, так что тестируемость никогда не становится проблемой. Автоматизируемый комплект тестов растет вместе с кодовой базой, обеспечивая "страховочную сетку" для постоянного рефакторинга. Очень важно, чтобы вся команда практиковала TDD и согласованно писала модульные тесты — иначе в страховочной сетке появятся дыры.

📖 Командный этикет TDD описан в [56].

К тому же команда не накапливает слишком большого технического долга, а скорость разработки остается стабильной или даже со временем растет. Это одна из причин, почему бизнес-менеджеры должны быть счастливы, выделяя командам разработчиков программного обеспечения время на правильную реализацию хороших приемов.

Тесты означают замечательную документацию

В части III было показано, как гибкие команды используют примеры и тесты для управления разработкой. Когда автоматизированы тесты, иллюстрирующие примеры желаемого поведения, они становятся "живой" документацией, описывающей работу системы. Иметь написанную документацию о работе каждого фрагмента функциональности, конечно, замечательно, но никто не сможет спорить с исполняемым тестом, который наглядно показывает зеленым и красным цветом, как работает код при заданном наборе входных данных.

Если не обновлять автоматизированные тесты при изменениях системы, они перестанут корректно проходить. Это значит, что автоматизированные тесты всегда являются точным отражением работы кода. Это лишь один из путей, которыми окупаются затраты на автоматизацию.

Возврат инвестиций и окупаемость

Все перечисленные выше причины вносят свой вклад в окупаемость автоматизации. Автоматизация обеспечивает проекту согласованность и позволяет команде дифференцированно подходить к тестированию и пробовать приложение на прочность. Автоматизация высвобождает дополнительное время для тестировщиков и членов команды, чтобы сконцентрироваться на своевременном выводе на рынок правильного продукта.

Важный компонент окупаемости тестирования заключается в способе исправления дефектов. Команды, полагающиеся на ручные тесты, часто находят ошибки по истечении значительного времени после написания кода, содержащего эти ошибки. Они работают в режиме исправления "ошибки дня" вместо того, чтобы выявлять первоначальную причину ошибки с соответствующим изменением кода. Когда программисты запускают комплекты автоматизированных тестов в своих собственных "песочницах", то эти регрессивные тесты позволяют выявлять ошибки еще до фиксации кода в системе управления версиями, так что у них остается время для соответствующей корректировки. Это обеспечивает замечательную окупаемость, сокращает технический долг и позволяет разрабатывать надежный код.

Барьеры на пути автоматизации

Еще в 2001 г. Брет Петтихорд (Bret Pettichord) перечислил несколько проблем, от которых страдает автоматизация. Они до сих пор актуальны, но в большей степени касаются команд, которые не включают автоматизацию в свой процесс разработки. И, конечно, если вы отдали предпочтение гибкой методологии, то ее нужно придерживаться, верно?

Нам хотелось бы думать, что все включают задачи автоматизации в каждую историю, но в действительности вы бы вряд ли читали этот раздел, если бы все находилось под контролем. Мы включили сюда список Брета, чтобы показать, какие проблемы вы получите, если не сделаете автоматизацию неотъемлемой частью процесса работы над проектом.

Список Брета

Список Брета проблем автоматизации выглядит следующим образом.

- Выделение времени на автоматизацию тестов по остаточному принципу не позволяет сосредоточиться на ней в должной мере.
- Недостаток ясности целей.
- Недостаток опыта.
- Это большая реорганизация, потому что вы теряете опыт, который могли иметь.
- Реакция на безысходность часто служит причиной выбора в пользу автоматизации, и в этом случае она будет скорее желанием, чем реалистическим планом.
- Возможно, вам просто лень думать о тестировании; удовольствие связано с автоматизацией, а не с тестированием.
- Сосредоточение на разрешении технической проблемы может привести к утере видения того, что результат отвечает нуждам тестирования.

Мы полагаем, что существуют и другие проблемы, с которыми сталкиваются команды, пытаясь автоматизировать тестирование. Даже если мы попытаемся включить автоматизацию в цели проекта, на пути к успеху могут возникнуть и другие барьеры. В следующем разделе будет представлен список препятствий успешной автоматизации тестирования.

Список препятствий

Наш список препятствий успешной автоматизации тестов основан на опыте собственной гибкой команды, а также других известных нам команд:

- позиция программистов;
- боль перемен;
- начальные инвестиции;
- постоянно меняющийся код;
- унаследованные системы;
- страх;
- старые привычки.

Позиция программистов: “Зачем автоматизировать?”

Программисты, привыкшие работать в традиционной среде, где есть отдельная, невидимая команда обеспечения качества (QA), занимающаяся тестированием, могут вообще не задумываться об автоматизации функциональных тестов. Некоторые программисты не слишком задумываются о тестах, потому что для них “страховочной сеткой” служит команда QA, которая отлавливает ошибки до выхода выпуска. Длительные циклы каскадной разработки еще более отдаляют тестирование от программистов. К моменту, когда невидимые тестировщики приступают к своей работе, программисты уже переходят к следующему выпуску. Дефекты ставятся в очередь для последующего дорогостоящего исправления, и никто не отвечает за них. Даже программисты, которые применяют управляемую тестами разработку и применяют автоматизацию тестирования на уровне модулей, могут не думать о том, как выполнять приемочные тесты, выходящих за пределы уровня модулей.

История от Лайзы

Однажды я присоединилась к команде XP, в которой работали квалифицированные программисты, практикующие управляемую тестами разработку и имеющие подходящий комплект модульных тестов, встроенных в процесс сборки. Они никогда не автоматизировали тесты стороны заказчика, и потому однажды я завязала разговор о том, какие инструменты можно было бы применять для автоматизации функциональных регрессивных тестов, ориентированных на заказчика. Программисты недоумевали: зачем вообще автоматизировать эти тесты? В конце первой итерации, когда каждый выполнял приемочные тесты вручную, я указала, что все эти тесты на следующей итерации нужно включить в комплект регрессивных тестов — вдобавок к тестам для всех новых историй. В третьей итерации тестов было уже втрое больше. Для тестировщика все это выглядит достаточно очевидным, но программистам иногда необходимо тестировать вручную, чтобы понять необходимость автоматизации.

Ключом к убеждению программистов и прочих членов команды в пользе и важности автоматизации является образование.

Горб боли (кривая обучения)

Трудно научиться автоматизации тестирования, в особенности тому, как организовать ее таким образом, чтобы получить хорошую отдачу от затраченных ресурсов. Для описания ситуации начальной фазы автоматизации, которую приходится преодолевать разработчикам (включая тестировщиков), Брайан Марик предложил термин “горб боли” (hump of pain); на рис. 13.1 показана соответствующая иллюстрация. Это словосочетание означает борьбу, через которую приходится пройти большинству команд при внедрении автоматизации.

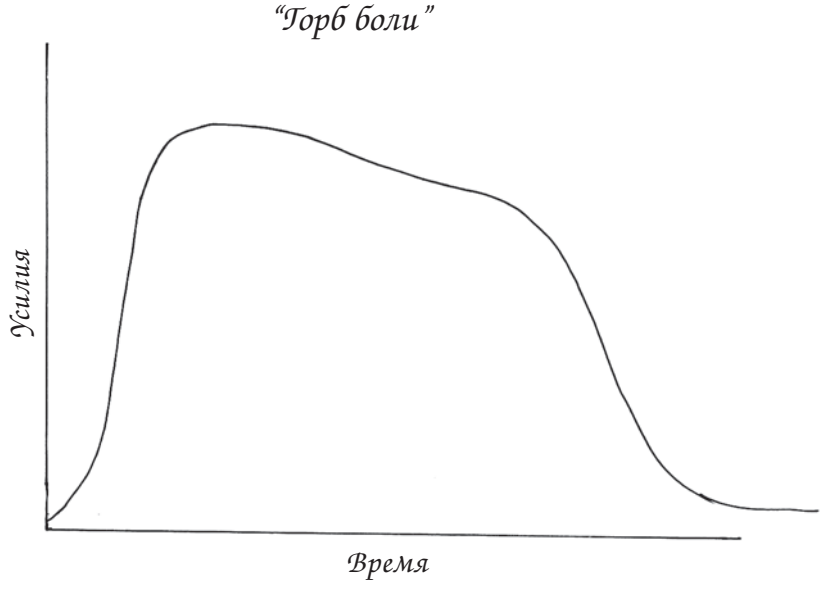


Рис. 13.1. “Горб боли” — кривая обучения автоматизации тестирования

Новые команды часто хотят внедрять приемы, подобные TDD и рефакторингу, которым нелегко научиться. Без хорошего наставника, не потратив достаточного времени для овладения новыми навыками, и не имея сильной поддержки со стороны менеджмента, они быстро утрачивают первоначальный энтузиазм.

Если же их задача осложняется дополнительными препятствиями, такими как необходимость работы с плохо спроектированным унаследованным кодом, то внедрение автоматизации тестирования становится практически невозможным.

История от Лайзы

Моя команда из ePlan Services изначально пыталась писать модульные тесты для унаследованной системы, которая определенно была разработана без учета тестируемости. Они поняли, что это будет трудной, если не невозможной задачей, поэтому решили кодировать новые истории в новой, тестируемой архитектуре. Интересно, что около года спустя они вдруг обнаружили, что было не так уж сложно писать модульные тесты для старого кода. Проблема состояла в том, что они поначалу вообще не знали, как писать модульные тесты, и научиться этому было легче на примере хорошо спроектированной архитектуры. Написание модульных тестов стало для них естественной частью написания кода.

"Горб боли" может случиться во время построения специфичного для предметной области каркаса тестирования или при изучении нового инструмента функционального тестирования. Чтобы правильно начать, может понадобиться привлечь эксперта.

Вы почувствуете, что "горб" пройден, когда автоматизация станет если не легким, то, по крайней мере, естественным и привычным процессом. Лайза работала в трех командах, которые успешно внедрили TDD и автоматизацию функциональных тестов. Каждый раз для привыкания к процессу команде требовалось много времени, тренировок, упорства и поощрений.

Начальные инвестиции

Даже когда вся команда работает над проблемой, автоматизация требует серьезных инвестиций, которые не скоро окупаются. Требуется время и исследования, чтобы выбрать подходящий каркас тестирования и решить, стоит ли его строить самостоятельно или воспользоваться готовым инструментом. Также может понадобиться новое аппаратное и программное обеспечение. Членам команды может потребоваться много времени, чтобы научиться использовать оснастку автоматизированного тестирования.

Многие люди сталкивались с тем, что усилия по внедрению автоматизации не оправдывали себя. Их организации покупали готовые инструменты съемки-воспроизведения, передавали их команде QA и ожидали, что они решат все проблемы автоматизации тестирования. Часто такие инструменты просто оставались пылиться на полке. В компании могли существовать тысячи строк сгенерированных сценариев тестирования графического интерфейса пользователя, но ни одной живой души, которая бы знала, что они делают; аналогично обстояли дела с тестовыми сценариями, которые невозможно поддерживать и которые стали бесполезными.

История от Джанет

Я пришла в одну организацию в качестве нового менеджера QA. Одной из моих задач была оценка имеющихся сценариев автоматизированного тестирования и расширение покрытия кода тестами. Несколькими годами ранее был приобретен коммерческий инструмент, а тестировщики, которые разработали начальный комплект, в этой организации уже не работали. Один из новых тестировщиков пытался изучить инструмент и добавлять тесты к комплекту.

Первое, что я сделала — попросила этого тестировщика оценить имеющийся комплект тестов на предмет степени покрытия кода. Она потратила неделю, только пытаясь понять, как организованы тесты. Я тоже подключилась к процессу и обнаружила, что существующие тесты были плохо спроектированы и представляли весьма незначительную ценность.

Мы прекратили добавлять новые тесты, а вместо этого потратили время на то, чтобы разобраться с целями автоматизации тестирования. Как вскоре стало ясно, имевшийся в наличии коммерческий инструмент не мог делать то, что действительно было нужно, поэтому мы отказались от лицензии и нашли инструмент с открытым исходным кодом, который отвечал существующим потребностям.

Нам пришлось потратить время на изучение этого нового инструмента, но на те же затраты пришлось бы пойти даже если бы мы остались верны прежнему коммерческому инструменту, потому что никто в команде уже не знал, как его использовать.

Квалификация проектировщика тестов оказывает громадное влияние на отдачу от автоматизации. Плохие приемы порождают тесты, которые трудно понять и поддерживать, и которые могут выдавать трудно интерпретируемые результаты, либо давать ложные сбои, на исследование которых может понадобиться дополнительное время. Команды с неадекватным уровнем подготовки и квалификацией могут решить, что попытки получить отдачу от инвестиций в тестирование не стоят их времени.

Правильные приемы проектирования тестов порождают простые, хорошо спроектированные, поддающиеся постоянному рефакторингу и легко сопровождаемые тесты. Со временем появляются библиотеки тестовых модулей и объектов, которые облегчают автоматизацию новых тестов. В главе 14 даются некоторые подсказки и руководства по проектированию тестов, подлежащих автоматизации.

Нам известно, что часто зафиксировать показатели нелегко. Например, зафиксировать время, необходимое на написание и сопровождение автоматизированных тестов, в сравнении с запуском тех же регрессивных тестов вручную практически невозможно. Также непросто зафиксировать, сколько стоит устранение дефектов через несколько минут после их появления, по сравнению со стоимостью нахождения и исправления проблем после окончания итерации. Многие команды даже не пытаются отслеживать такую информацию. Но не имея возможности наглядно продемонстрировать экономический эффект от внедрения автоматизации тестирования, трудно убедить менеджмент в целесообразности инвестиций в эту деятельность. Недостаток показателей, демонстрирующих отдачу от инвестиций, затрудняет задачу изменения старых привычек команды.

Текучесть кода

Автоматизация тестов пользовательского интерфейса — сложная задача, поскольку данные интерфейсы имеют тенденцию часто изменяться в процессе разработки. Это одна из причин, по которым простая запись и воспроизведение действий пользователя редко применяются в гибких проектах.

Если команда стремится разрабатывать хороший дизайн на базе лежащей в основе бизнес-логики и доступа к базе данных, и серьезные изменения происходят часто, то трудно поддерживать даже автоматизацию тестов, которые проверяют код, находящийся ниже графического интерфейса пользователя — на уровне API. Если при проектировании системы было уделено мало внимания возможности тестирования, найти способ автоматизации тестов может оказаться трудно и дорого. Для создания тестируемого приложения программисты и тестировщики должны работать вместе.

Хотя действительный код и реализация, подобная графическому интерфейсу пользователя, имеют тенденцию часто изменяться в гибкой разработке, назначение кода изменяется редко. Организуйте тестирование кода на основе предназначения приложения, а не на основе его реализации; это позволит не отставать от разработки.

📖 Способы организации автоматизированных тестов рассматриваются в главе 14.

Унаследованный код

По нашему опыту автоматизацию внедрить намного легче, если вы пишете совершенно новый код в архитектуре, спроектированной с учетом возможности тестирования. Написание тестов для существующего кода, который имеет минимум тестов или не имеет их вовсе — задача, как минимум, устрашающая. Это кажется практически невозможным для команды новичков в гибкой разработке и автоматизации тестирования.

Иногда нужно автоматизировать тесты, чтобы иметь возможность выполнить рефакторинг некоторого унаследованного кода, но этот код не спроектирован с учетом тестируемости, так что автоматизировать тесты трудно даже на уровне модулей.

Если команда столкнулась с такой проблемой, и не имеет достаточного времени для выработки тактики ее решения, ищите в главе 14 описание стратегии решения подобных проблем.

Страх

Задача автоматизации тестов пугает тех, кто не имеет соответствующих навыков, а иногда даже и тех, кто имеет. Программисты могут уметь писать продуктивный код, но не иметь опыта в написании автоматизированных тестов. Тестировщики могут не иметь богатого опыта программирования, и не уверены в своих способностях автоматизации тестов.

Тестировщики — непрограммисты часто слышат, что им нечего делать в мире гибкой разработки. Мы думаем иначе. Ни один индивидуальный тестировщик не должен беспокоиться о том, как осуществить автоматизацию. Это проблема команды в целом, и обычно в команде всегда есть масса программистов, которые могут в этом помочь. Секрет успеха — в готовности овладевать новыми идеями. Потратьте на это хотя бы день.

Старые привычки

Когда итерация идет не гладко, и команда не успевает завершить все задачи программирования и тестирования к концу итерации, члены команды могут запаниковать. Мы заметили, что когда люди впадают в панику, они возвращаются к старым привычкам, даже несмотря на то, что это не может обеспечить хороший результат.

Итак, мы можем сказать: “У нас срок сдачи 1 февраля. Если мы хотим уложиться в срок, то не можем тратить время на автоматизацию тестов. Мы должны сделать все возможное, чтобы протестировать вручную все, что успеем, и надеяться на лучшее. Мы всегда сможем автоматизировать тесты позднее”.

Это — гибельный путь. Некоторые ручные тесты могут быть выполнены, но, может быть, не останется времени на важные ручные исследовательские тесты, которые обнаружили бы ошибку, стоящую компании сотен тысяч долларов из-за потерянных продаж. Затем, поскольку мы так и не закончили автоматизированные тесты, эта задача откладывается на следующую итерацию, сокращая объем полезных средств, которые мы успеем реализовать. С каждой новой итерацией ситуация продолжит ухудшаться.

Можно ли преодолеть эти барьеры?

Гибкий полнокомандный подход — основа преодоления проблем автоматизации. Программисты-новички в гибкой разработке обычно ориентированы на поставку кода, независимо от того, полон он ошибок или нет, если они укладываются в сроки. Управляемая тестами разработка больше ориентирована на проектирование, чем на тестирование, поэтому бизнес-ориентированные тесты могут быть упущены из виду. Необходим сильный лидер и преданность команды качеству, чтобы заставить всех думать о том, как писать, использовать и запускать технологически-ориентированные и бизнес-ориентированные тесты. Вовлечение всей команды в автоматизацию тестирования может оказаться проблемой культуры.

📖 В главе 3 представлено изложение некоторых идей относительно изменений в культуре, ориентированных на облегчение гибкого процесса.

В следующей главе мы покажем, как использовать ценности и принципы гибкой методологии для преодоления некоторых проблем, описанных в этой главе.

Резюме

В этой главе были проанализированы некоторые важные факторы, касающиеся автоматизации тестирования.

- Автоматизация необходима для создания “страховочной сетки”, получения существенного отклика, сведения к минимуму уровня технического долга и помощи в управлении кодированием.
- Страх, недостаток знаний, отрицательный прошлый опыт внедрения автоматизации, быстро изменяющийся код, унаследованный код — все это распространенные барьеры на пути автоматизации.
- Автоматизация регрессивных тестов, запуск их в составе автоматизированного процесса сборки и устранение первопричин дефектов сокращает технический долг и позволяет разрабатывать надежный код.
- Автоматизация регрессивных тестов и утомительной ручной работы освобождает команду для решения более важных задач, таких как исследовательское тестирование.
- Команды, практикующие автоматизированное тестирование и автоматизированный процесс сборки, демонстрируют более стабильную производительность.
- Без автоматизации регрессивных тестов объем ручного регрессивного тестирования будет продолжать расти в объеме и в конечном итоге может просто игнорироваться.
- Культура и история команды могут затруднить программистам отдать приоритет автоматизации бизнес-ориентированных тестов перед кодированием новых средств. Использование гибких принципов и ценностей помогает всей команде преодолеть барьеры, стоящие на пути автоматизации тестирования.