Раздел 3: автоматизация тестирования

3.1. Выгоды и риски автоматизации

3.1.1. Преимущества и недостатки автоматизации

В разделе, посвящённом подробной классификации тестирования{63}, мы кратко рассматривали, что собой представляет автоматизированное тестирование{69}: это набор техник, подходов и инструментальных средств, позволяющий исключить человека из выполнения некоторых задач в процессе тестирования. В таблице 2.3.b{70} был приведён краткий список преимуществ и недостатков автоматизации, который сейчас мы рассмотрим подробно.

- Скорость выполнения тест-кейсов может в разы и на порядки превосходить возможности человека. Если представить, что человеку придётся вручную сверять несколько файлов размером в несколько десятков мегабайт каждый, оценка времени ручного выполнения становится пугающей: месяцы или даже годы. При этом 36 проверок, реализуемых в рамках дымового тестирования командными скриптами{271}, выполняются менее чем за пять секунд и требуют от тестировщика только одного действия — запустить скрипт.
- Отсутствует влияние человеческого фактора в процессе выполнения тест-кейсов (усталости, невнимательности и т.д.) Продолжим пример из предыдущего пункта: какова вероятность, что человек ошибётся, сравнивая (посимвольно!) даже два обычных текста размером в 100 страниц каждый? А если таких текстов 10? 20? И проверки нужно повторять раз за разом? Можно смело утверждать, что человек ошибётся гарантированно. Автоматика не ошибётся.
- Средства автоматизации способны выполнить тест-кейсы, в принципе непосильные для человека в силу своей сложности, скорости или иных факторов. И снова наш пример со сравнением больших текстов является актуальным: мы не можем позволить себе потратить годы, раз за разом выполняя крайне сложную рутинную операцию, в которой мы к тому же будем гарантированно допускать ошибки. Другим прекрасным примером непосильных для человека тест-кейсов является исследование производительности{85}, в рамках которого необходимо с высокой скоростью выполнять определённые действия, а также фиксировать значения широкого набора параметров. Сможет ли человек, например, сто раз в секунду измерять и записывать объём оперативной памяти, занимаемой приложением? Нет. Автоматика сможет.
- Средства автоматизации способны собирать, сохранять, анализировать, агрегировать и представлять в удобной для восприятия человеком форме колоссальные объёмы данных. В нашем примере с дымовым тестированием «Конвертера файлов» объём данных, полученный в результате тестирования, невелик — его вполне можно обработать вручную. Но если обратиться к реальным проектным ситуациям, журналы работы систем автоматизированного тестирования могут занимать десятки гигабайт по каждой итерации. Логично, что человек не в состоянии вручную проанализировать такие объёмы данных, но правильно настроенная среда автоматизации сделает это сама, предоставив на выход аккуратные отчёты в 2–3 страницы, удобные графики и таблицы, а также возможность погружаться в детали, переходя от агрегированных данных к подробностям, если в этом возникнет необходимость.
- Средства автоматизации способны выполнять низкоуровневые действия с приложением, операционной системой, каналами передачи данных и т.д. В

одном из предыдущих пунктов мы упоминали такую задачу, как «сто раз в секунду измерить и записать объём оперативной памяти, занимаемой приложением». Подобная задача сбора информации об используемых приложением ресурсах является классическим примером. Однако средства автоматизации могут не только собирать подобную информацию, но и воздействовать на среду исполнения приложения или само приложение, эмулируя типичные события (например, нехватку оперативной памяти или процессорного времени) и фиксируя реакцию приложения. Даже если у тестировщика будет достаточно квалификации, чтобы самостоятельно выполнить подобные операции, ему всё равно понадобится то или иное инструментальное средство — так почему не решить эту задачу сразу на уровне автоматизации тестирования?

Итак, с использованием автоматизации мы получаем возможность увеличить тестовое покрытие{203} за счёт:

- выполнения тест-кейсов, о которых раньше не стоило и думать;
- многократного повторения тест-кейсов с разными входными данными;
- высвобождения времени на создание новых тест-кейсов.

Но всё ли так хорошо с автоматизацией? Увы, нет. Очень наглядно одну из серьёзных проблем можно представить рисунком 3.1.а:

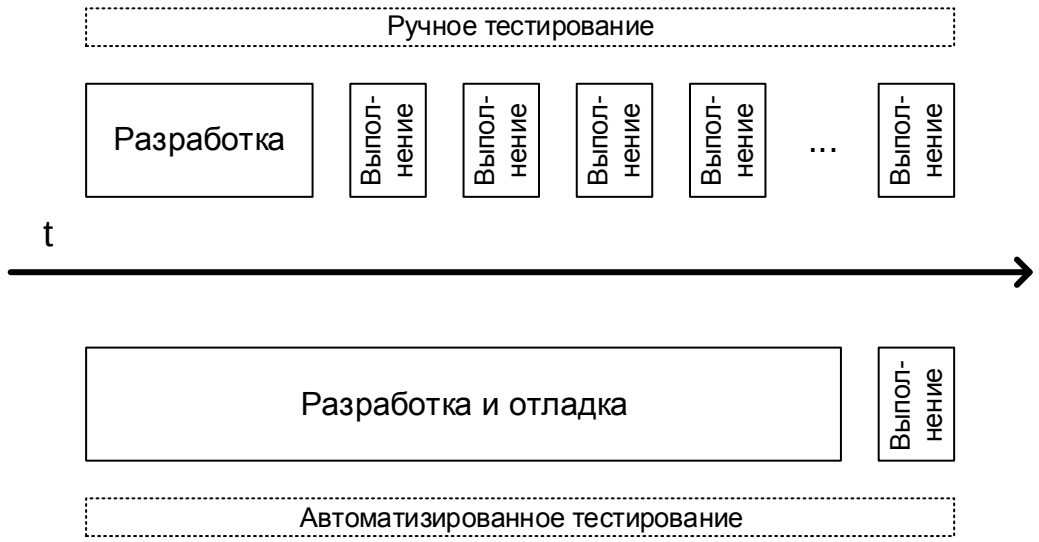


Рисунок 3.1.а — Соотношение времени разработки и выполнения тест-кейсов в ручном и автоматизированном тестировании

В первую очередь следует осознать, что автоматизация не происходит сама по себе, не существует волшебной кнопки, нажатием которой решаются все проблемы. Более того, с автоматизацией тестирования связана серия серьёзных недостатков и рисков:

- Необходимость наличия высококвалифицированного персонала в силу того факта, что автоматизация — это «проект внутри проекта» (со своими требованиями, планами, кодом и т.д.). Даже если забыть на мгновение про «проект внутри проекта», техническая квалификация сотрудников, занимающихся автоматизацией, как правило, должна быть ощутимо выше, чем у их коллег, занимающихся ручным тестированием.
- Разработка и сопровождение как самих автоматизированных тест-кейсов, так и всей необходимой инфраструктуры занимает очень много времени. Ситуация усугубляется тем, что в некоторых случаях (при серьёзных изменениях в

проекте или в случае ошибок в стратегии) всю соответствующую работу приходится выполнять заново с нуля: в случае ощутимого изменения требований, смены технологического домена, переработки интерфейсов (как пользовательских, так и программных) многие тест-кейсы становятся безнадёжно устаревшими и требуют создания заново.

- Автоматизация требует более тщательного планирования и управления рисками, т.к. в противном случае проекту может быть нанесён серьёзный ущерб (см. предыдущий пункт про переделку с нуля всех наработок).
- Коммерческие средства автоматизации стоят ощутимо дорого, а имеющиеся бесплатные аналоги не всегда позволяют эффективно решать поставленные задачи. И здесь мы снова вынуждены вернуться к вопросу ошибок в планировании: если изначально набор технологий и средств автоматизации был выбран неверно, придётся не только переделывать всю работу, но и покупать новые средства автоматизации.
- Средств автоматизации крайне много, что усложняет проблему выбора того или иного средства, затрудняет планирование и определение стратегии тестирования, может повлечь за собой дополнительные временные и финансовые затраты, а также необходимость обучения персонала или найма соответствующих специалистов.

Итак, автоматизация тестирования требует ощутимых инвестиций и сильно повышает проектные риски, а потому существуют специальные подходы[361, 362, 363, 364] по оценке применимости и эффективности автоматизированного тестирования. Если выразить всю их суть очень кратко, то в первую очередь следует учесть:

- Затраты времени на ручное выполнение тест-кейсов и на выполнение этих же тест-кейсов, но уже автоматизированных. Чем ощутимее разница, тем более выгодной представляется автоматизация.
- Количество повторений выполнения одних и тех же тест-кейсов. Чем оно больше, тем больше времени мы сможем сэкономить за счёт автоматизации.
- Затраты времени на отладку, обновление и поддержку автоматизированных тест-кейсов. Этот параметр сложнее всего оценить, и именно он представляет наибольшую угрозу успеху автоматизации, потому здесь для проведения оценки следует привлекать наиболее опытных специалистов.
- Наличие в команде соответствующих специалистов и их рабочую загрузку. Автоматизацией занимаются самые квалифицированные сотрудники, которые в это время не могут решать иные задачи.

[361] «Implementing Automated Software Testing – Continuously Track Progress and Adjust Accordingly», Thom Garrett [http://www.methodsandtools.com/archive/archive.php?id=94]

[362] «The Return of Investment (ROI) of Test Automation», Stefan Münch and others.[https://www.ispe.org/pe-ja/roi-of-test-automation.pdf]

[363] «The ROI of Test Automation», Michael Kelly [http://www.sqetraining.com/sites/default/files/articles/XDD8502filelistfilename1_0.pdf]

[364] «Cost Benefits Analysis of Test Automation», Douglas Hoffman [http://www.softwarequalitymethods.com/papers/star99%20model%20paper.pdf]

В качестве небольшого примера беглой оценки эффективности автоматизации можно привести следующую формулу[365]:

$A_{effect} = \frac{N \cdot T_{manual}^{overall}}{N \cdot T_{automated}^{run\ and\ analyse} + T_{automated}^{development\ and\ support}}$, где

A_{effect} — коэффициент выгоды от использования автоматизации,

N — планируемое количество билдов приложения;

$T_{manual}^{overall}$ — расчётное время разработки, выполнения и анализа результатов ручного тестирования;

$T_{automated}^{run\ and\ analyse}$ — расчётное время выполнения и анализа результатов автоматизированного тестирования;

$T_{automated}^{development\ and\ support}$ — расчётное время разработки и сопровождения автоматизированного тестирования.

Чтобы нагляднее представить, как эта формула может помочь, изобразим график коэффициента выгоды автоматизации в зависимости от количества билдов (рисунок 3.1.b). Допустим, что в некотором проекте значения параметров таковы:

$T_{manual}^{overall}$ = 30 часов на каждый билд;

$T_{automated}^{run\ and\ analyse}$ = 5 часов на каждый билд;

$T_{automated}^{development\ and\ support}$ = 300 часов на весь проект.

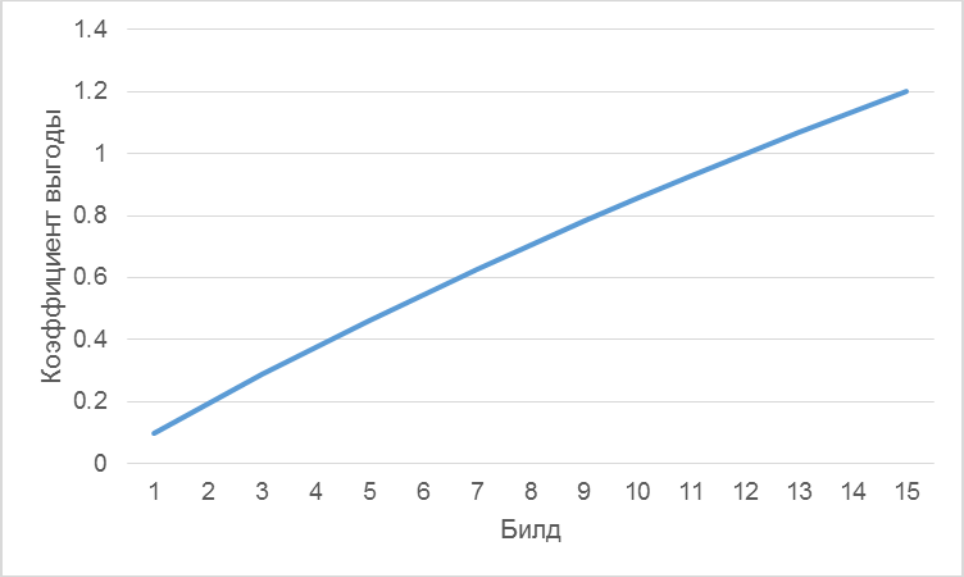


Рисунок 3.1.b — Коэффициент выгоды автоматизации в зависимости от количества билдов

Как видно на рисунке 3.1.b, лишь к 12-му билду автоматизация окупит вложения и с 13-го билда начнёт приносить пользу. И тем не менее существуют области, в которых автоматизация даёт ощутимый эффект почти сразу. Их рассмотрению посвящена следующая глава.

[365] «Introduction to automation», Vitaliy Zhyrytskyy.

3.1.2. Области применения автоматизации

Сначала мы ещё раз посмотрим на список задач, решить которые помогает автоматизация:

- Выполнение тест-кейсов, непосильных человеку.
- Решение рутинных задач.
- Ускорение выполнения тестирования.
- Высвобождение человеческих ресурсов для интеллектуальной работы.
- Увеличение тестового покрытия.
- Улучшение кода за счёт увеличения тестового покрытия и применения специальных техник автоматизации.

Эти задачи чаще всего встречаются и проще всего решаются в следующих случаях (см. таблицу 3.1.a).

Таблица 3.1.a — Случаи наибольшей применимости автоматизации

Случай / задача	Какую проблему решает автоматизация
Регрессионное тестирование{80}.	Необходимость выполнять вручную тесты, количество которых неуклонно растёт с каждым билдом, но вся суть которых сводится к проверке того факта, что ранее работавшая функциональность продолжает работать корректно.
Инсталляционное тестирование{80} и настройка тестового окружения.	Множество часто повторяющихся рутинных операций по проверке работы инсталлятора, размещения файлов в файловой системе, содержимого конфигурационных файлов, реестра и т.д. Подготовка приложения в заданной среде и с заданными настройками для проведения основного тестирования.
Конфигурационное тестирование{83} и тестирование совместимости{83}.	Выполнение одних и тех же тест-кейсов на большом множестве входных данных, под разными платформами и в разных условиях. Классический пример: есть файл настроек, в нём сто параметров, каждый может принимать сто значений: существует 10’000 вариантов конфигурационного файла — все их нужно проверить.
Использование комбинаторных техник тестирования{101} (в т.ч. доменного тестирования{89}, {229}).	Генерация комбинаций значений и многократное выполнение тест-кейсов с использованием этих сгенерированных комбинаций в качестве входных данных.
Модульное тестирование{71}.	Проверка корректности работы атомарных участков кода и элементарных взаимодействий таких участков кода — практически невыполнимая для человека задача при условии, что нужно выполнить тысячи таких проверок и нигде не ошибиться.
Интеграционное тестирование{71}.	Глубокая проверка взаимодействия компонентов в ситуации, когда человеку почти нечего наблюдать, т.к. все представляющие интерес и подвергаемые тестированию процессы проходят на уровнях более глубоких, чем пользовательский интерфейс.
Тестирование безопасности{83}.	Необходимость проверки прав доступа, паролей по умолчанию, открытых портов, уязвимостей текущих версий ПО и т.д., т.е. быстрое выполнение очень большого количества проверок, в процессе которого нельзя что-то пропустить, забыть или «не так понять».
Тестирование производительности{85}.	Создание нагрузки с интенсивностью и точностью, недоступной человеку. Сбор с высокой скоростью большого набора параметров работы приложения. Анализ большого объёма данных из журналов работы системы автоматизации.
Дымовой тест{73} для крупных систем.	Выполнение при получении каждого билда большого количества достаточно простых для автоматизации тест-кейсов.
Приложения (или их части) без графического интерфейса.	Проверка консольных приложений на больших наборах значений параметров командной строки (и их комбинаций). Проверка приложений и их компонентов, вообще не предназначенных для взаимодействия с человеком (веб-сервисы, серверы, библиотеки и т.д.)

Длительные, рутинные, утомительные для человека и/или требующие повышенного внимания операции.	Проверки, требующие сравнения больших объёмов данных, высокой точности вычислений, обработки большого количества размещённых по всему дереву каталогов файлов, ощутимо большого времени выполнения и т.д. Особенно, когда такие проверки повторяются очень часто.
Проверка «внутренней функциональности» веб-приложений (ссылок, доступности страниц и т.д.)	Автоматизация предельно рутинных действий (например, проверить все 30’000+ ссылок на предмет того, что все они ведут на реально существующие страницы). Автоматизация здесь упрощается в силу стандартности задачи — существует много готовых решений.
Стандартная, однотипная для многих проектов функциональность.	Даже высокая сложность при первичной автоматизации в таком случае окупится за счёт простоты многократного использования полученных решений в разных проектах.
«Технические задачи».	Проверки корректности протоколирования, работы с базами данных, корректности поиска, файловых операций, корректности форматов и содержимого генерируемых документов и т.д.

С другой стороны, существуют случаи, в которых автоматизация, скорее всего, приведёт только к ухудшению ситуации. Вкратце — это все те области, где требуется человеческое мышление, а также некоторый перечень технологических областей. Чуть более подробно список выглядит так (таблица 3.1.b):

Таблица 3.1.b — Случаи наименьшей применимости автоматизации

Случай / задача	**В чём проблема автоматизации**
Планирование{196}.	Компьютер пока не научился думать.
Разработка тест-кейсов{114}.	
Написание отчётов о дефектах{160}.	
Анализ результатов тестирования и отчётность{196}.	
Функциональность, которую нужно (достаточно) проверить всего несколько раз.	Затраты на автоматизацию не окупятся.
Тест-кейсы, которые нужно выполнить всего несколько раз (если человек может их выполнить).	
Низкий уровень абстракции в имеющихся инструментах автоматизации.	Придётся писать очень много кода, что не только сложно и долго, но и приводит к появлению множества ошибок в самих тест-кейсах.
Слабые возможности средства автоматизации по протоколированию процесса тестирования и сбору технических данных о приложении и окружении.	Есть риск получить данные в виде «что-то где-то сломалось», что не помогает в диагностике проблемы.
Низкая стабильность требований.	Придётся очень многое переделывать, что в случае автоматизации обходится дороже, чем в случае ручного тестирования.
Сложные комбинации большого количества технологий.	Высокая сложность автоматизации, низкая надёжность тест-кейсов, высокая сложность оценки трудозатрат и прогнозирования рисков.
Проблемы с планированием и ручным тестированием.	Автоматизация хаоса приводит к появлению автоматизированного хаоса, но при этом ещё и требует трудозатрат. Сначала стоит решить имеющиеся проблемы, а потом включаться в автоматизацию.
Нехватка времени и угроза срыва сроков	Автоматизация не приносит мгновенных результатов. Поначалу она лишь потребляет ресурсы команды (в том числе время). Также есть универсальный афоризм: «лучше руками протестировать хоть что-то, чем автоматизированно протестировать ничего».
Области тестирования, требующие оценки	В принципе, можно разработать некие алгоритмы, оценивающие ситуацию так, как её мог бы оценить

ситуации человеком (тестирование удобства использования{82}, тестирование доступности{82} и т.д.)	человек. На практике живой человек сделать это быстрее, проще, надёжнее и дешевле.

Вывод: стоит помнить, что эффект от автоматизации наступает не сразу и не всегда. Как и любой дорогостоящий инструмент, автоматизация при верном применении может дать ощутимую выгоду, но при неверном принесёт лишь весьма ощутимые затраты.

3.2. Особенности автоматизированного тестирования

3.2.1. Необходимые знания и навыки

Во множестве источников, посвящённых основам автоматизации тестирования, можно встретить схемы наподобие представленной на рисунке 3.2.a — то есть автоматизация тестирования представляет собой сочетание программирования и тестирования в разных масштабах (в зависимости от проекта и конкретных задач).

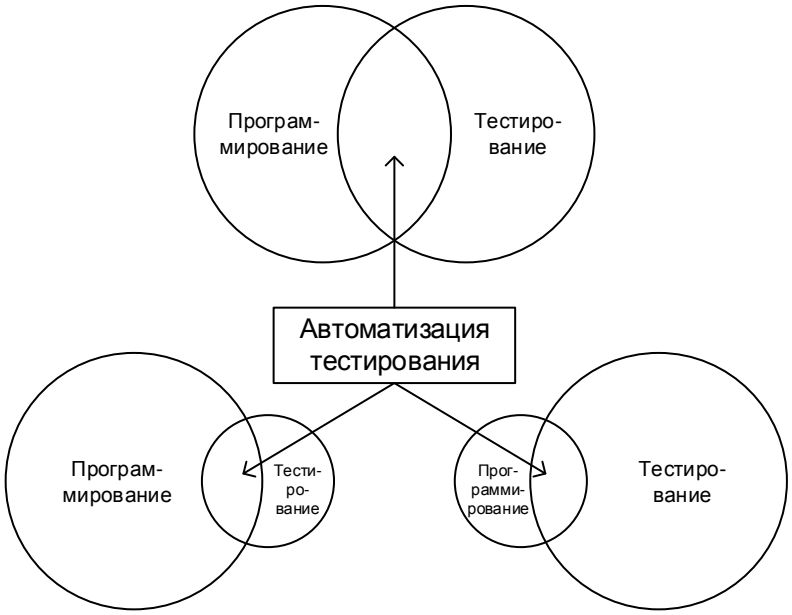


Рисунок 3.2.a — Сочетание программирования и тестирования в автоматизации тестирования

Отсюда следует простой вывод, что специалист по автоматизации тестирования должен сочетать в себе навыки и умения как программиста, так и тестировщика. Но этим перечень не заканчивается: умение администрировать операционные системы, сети, различные серверы, умение работать с базами данных, понимание мобильных платформ и т.д. — всё это может пригодиться.

Но даже если остановиться только на навыках программирования и тестирования, в автоматизации тоже есть свои особенности — набор технологий. В классическом ручном тестировании развитие происходит постепенно и эволюционно — проходят годы и даже десятилетия между появлением новых подходов, завоёвывающих популярность. В программировании прогресс идёт чуть быстрее, но и там специалистов выручает согласованность и схожесть технологий.

В автоматизации тестирования ситуация выглядит иначе: десятки и сотни технологий и подходов (как заимствованных из смежных дисциплин, так и уникальных) появляются и исчезают очень стремительно. Количество инструментальных средств автоматизации тестирования уже исчисляется тысячами и продолжает неуклонно расти.

Потому к списку навыков программирования и тестирования можно смело добавить крайне высокую обучаемость и способность в предельно сжатые сроки самостоятельно найти, изучить, понять и начать применять на практике совершенно новую информацию из, возможно, ранее абсолютно незнакомой области. Звучит немного пугающе, но одно можно гарантировать: скучно не будет точно.

О нескольких наиболее распространённых технологиях мы поговорим в главе «Технологии автоматизации тестирования»[{256}].

3.2.2. Особенности тест-кейсов в автоматизации

Часто (а в некоторых проектах и «как правило») автоматизации подвергаются тест-кейсы, изначально написанные простым человеческим языком (и, в принципе, пригодные для выполнения вручную) — т.е. обычные классические тест-кейсы, которые мы уже рассматривали подробно в соответствующей главе{114}.

И всё же есть несколько важных моментов, которые стоит учитывать при разработке (или доработке) тест-кейсов, предназначенных для дальнейшей автоматизации.

Главная проблема состоит в том, что компьютер — это не человек, и соответствующие тест-кейсы не могут оперировать «интуитивно понятными описаниями», а специалисты по автоматизации совершенно справедливо не хотят тратить время на то, чтобы дополнить такие тест-кейсы необходимыми для выполнения автоматизации техническими подробностями, — у них хватает собственных задач.

Отсюда следует список рекомендаций по подготовке тест-кейсов к автоматизации и непосредственно самой автоматизации:

- Ожидаемый результат в автоматизированных тест-кейсах должен быть описан предельно чётко с указанием конкретных признаков его корректности. Сравните:

Плохо	Хорошо
… 7. Загружается стандартная страница поиска.	… 7. Загружается страница поиска: title = «Search page», присутствует форма с полями «input type=”text”», «input type=“submit” value=“Go!”», присутствует логотип «logo.jpg» и отсутствуют иные графические элементы («img»).

- Поскольку тест-кейс может быть автоматизирован с использованием различных инструментальных средств, следует описывать его, избегая специфических для того или иного инструментального средства решений. Сравните:

Плохо	Хорошо
1. Кликнуть по ссылке «Search». 2. Использовать clickAndWait для синхронизации тайминга.	1. Кликнуть по ссылке «Search». 2. Дождаться завершения загрузки страницы.

- В продолжение предыдущего пункта: тест-кейс может быть автоматизирован для выполнения под разными аппаратными и программными платформами, потому не стоит изначально прописывать в него что-то, характерное лишь для одной платформы. Сравните:

Плохо	Хорошо
… 8. Отправить приложению сообщение WM_CLICK в любое из видимых окон.	… 8. Передать фокус ввода любому из не свёрнутых окон приложения (если таких нет — развернуть любое из окон). 9. Проэмулировать событие «клик левой кнопкой мыши» для активного окна.

- Одной из неожиданно проявляющихся проблем до сих пор является синхронизация средства автоматизации и тестируемого приложения по времени: в случаях, когда для человека ситуация является понятной, средство автоматизации тестирования может среагировать неверно, «не дождавшись» определённого состояния тестируемого приложения. Это приводит к завершению неудачей тест-кейсов на корректно работающем приложении. Сравните:

Плохо	Хорошо
1. Кликнуть по ссылке «Expand data». 2. Выбрать из появившегося списка значение «Unknown».	1. Кликнуть по ссылке «Expand data». 2. Дождаться завершения загрузки данных в список «Extended data» (select id=”extended_data”): список перейдёт в состояние enabled. 3. Выбрать в списке «Extended data» значение «Unknown».

- Не стоит искушать специалиста по автоматизации на вписывание в код тест-кейса константных значений (т.н. «хардкодинг»). Если вы можете понятно описать словами значение и/или смысл некоей переменной, сделайте это. Сравните:

Плохо	Хорошо
1. Открыть http://application/.	1. Открыть главную страницу приложения.

- По возможности следует использовать наиболее универсальные способы взаимодействия с тестируемым приложением. Это значительно сократит время поддержки тест-кейсов в случае, если изменится набор технологий, с помощью которых реализовано приложение. Сравните:

Плохо	Хорошо
… 8. Передать в поле «Search» набор событий WM_KEY_DOWN, {знак}, WM_KEY_UP, в результате чего в поле должен быть введён поисковый запрос.	… 8. Проэмулировать ввод значения поля «Search» с клавиатуры (не годится вставка значения из буфера или прямое присваивание значения!)

- Автоматизированные тест-кейсы должны быть независимыми. Из любого правила бывают исключения, но в абсолютном большинстве случаев следует предполагать, что мы не знаем, какие тест-кейсы будут выполнены до и после нашего тест-кейса. Сравните:

Плохо	Хорошо
1. Из файла, созданного предыдущим тестом…	1. Перевести чек-бокс «Use stream buffer file» в состояние checked. 2. Активировать процесс передачи данных (кликнуть по кнопке «Start»). 3. Из файла буферизации прочитать…

- Стоит помнить, что автоматизированный тест-кейс — это программа, и стоит учитывать хорошие практики программирования хотя бы на уровне отсутствия т.н. «магических значений», «хардкодинга» и тому подобного. Сравните:

Плохо	Хорошо

if ($date_value == ‘2015.06.18’)
{
…
}
«Хардкодинг»
«Магическое значение»
if ($status = 42)
{
…
}
Ошибка в выражении (= вместо ==)
	if ($date_value == date(‘Y.m.d’))
	{
	…
	}
	Осмысленная константа
	Актуальные данные
	if (POWER_USER == $status)
	{
	…
	}
	Ошибка исправлена, к тому же константа в сравнении находится слева от переменной

- Стоит внимательно изучать документацию по используемому средству автоматизации, чтобы избежать ситуации, когда из-за неверно выбранной команды тест-кейс становится ложно положительным, т.е. успешно проходит в ситуации, когда приложение работает неверно.



Так называемые ложно положительные тест-кейсы — едва ли не самое страшное, что бывает в автоматизации тестирования: они вселяют в проектную команду ложную уверенность в том, что приложение работает корректно, т.е. фактически прячут дефекты, вместо того, чтобы обнаруживать их.

Поскольку для многих начинающих тестировщиков первым учебным средством автоматизации тестирования является Selenium IDE[366], приведём пример с его использованием. Допустим, в некотором шаге тест-кейса нужно было проверить, что чек-бокс с id=cb выбран (checked). По какой-то причине тестировщик выбрал неверную команду, и сейчас на этом шаге проверятся, что чек-бокс позволяет изменять своё состояние (enabled, editable), а не что он выбран.

Плохо (неверная команда)			**Хорошо (верная команда)**		
…	…	…	…	…	…
verifyEditable	id=cb		verifyChecked	id=cb	
…	…	…	…	…	…

- И напоследок рассмотрим ошибку, которую по какой-то мистической причине совершает добрая половина начинающих автоматизаторов — это замена проверки действием и наоборот. Например, вместо проверки значения поля они изменяют значение. Или вместо изменения состояния чек-бокса проверяют его состояние. Здесь не будет примеров на «плохо/хорошо», т.к. хорошего варианта здесь нет — такого просто не должно быть, т.к. это — грубейшая ошибка.

Кратко подытожив рассмотренное, отметим, что тест-кейс, предназначенный для автоматизации, будет куда более похож на миниатюрное техническое задание по разработке небольшой программы, чем на описание корректного поведения тестируемого приложения, понятное человеку.

И ещё одна особенность автоматизированных тест-кейсов заслуживает отдельного рассмотрения — это источники данных и способы их генерации. Для выполняемых вручную тест-кейсов эта проблема не столь актуальна, т.к. при выполнении 3–5–10 раз мы также вручную вполне можем подготовить нужное количество вариантов входных данных. Но если мы планируем выполнить тест-кейс 50–100–500 раз с разными входными данными, вручную столько данных мы не подготовим. Источниками данных в такой ситуации могут стать:

- Случайные величины: случайные числа, случайные символы, случайные элементы из некоторого набора и т.д.
- Генерация (случайных) данных по алгоритму: случайные числа в заданном диапазоне, строки случайной длины из заданного диапазона из случайных символов из определённого набора (например, строка длиной от 10 до 100 символов, состоящая только из букв), файлы с увеличивающимся по некоему правилу размером (например, 10 КБ, 100 КБ, 1000 КБ и т.д.).

[366] Selenium IDE. [http://docs.seleniumhq.org/projects/ide/]

- Получение данных из внешних источников: извлечение данных из базы данных, обращение к некоему веб-сервису и т.д.
- Собранные рабочие данные, т.е. данные, созданные реальными пользователями в процессе их реальной работы (например, если бы мы захотели разработать собственный текстовый редактор, тысячи имеющихся у нас и наших коллег doc(x)-файлов были бы такими рабочими данными, на которых мы бы проводили тестирование).
- Ручная генерация — да, она актуальная и для автоматизированных тест-кейсов. Например, вручную создать по десять (да даже и по 50–100) корректных и некорректных e-mail-адресов получится куда быстрее, чем писать алгоритм их генерации.

Применение некоторых из этих идей по генерации данных мы рассмотрим подробнее в следующей главе.

3.2.3. Технологии автоматизации тестирования

В данной главе мы рассмотрим несколько высокоуровневых технологий автоматизации тестирования, каждая из которых, в свою очередь, базируется на своём наборе технических решений (инструментальных средствах, языках программирования, способах взаимодействия с тестируемым приложением и т.д.)

Начнём с краткого обзора эволюции высокоуровневых технологий, при этом подчеркнув, что «старые» решения по-прежнему используются (или как компоненты «новых», или самостоятельно в отдельных случаях).

Таблица 3.2.a — Эволюция высокоуровневых технологий автоматизации тестирования

№	Подход	Суть	Преимущества	Недостатки
1	Частные решения.	Для решения каждой отдельной задачи пишется отдельная программа.	Быстро, просто.	Нет системности, много времени уходит на поддержку. Почти невозможно повторное использование.
2	Тестирование под управлением данными{87} (DDT).	Из тест-кейса вовне выносятся входные данные и ожидаемые результаты.	Один и тот же тест-кейс можно повторять многократно с разными данными.	Логика тест-кейса по-прежнему строго определяется внутри, а потому для её изменения тест-кейс надо переписывать.
3	Тестирование под управлением ключевыми словами{87} (KDT).	Из тест-кейса вовне выносится описание его поведения.	Концентрация на высокоуровневых действиях. Данные и особенности поведения хранятся вовне и могут быть изменены без изменения кода тест-кейса.	Сложность выполнения низкоуровневых операций.
4	Использование фреймворков.	Конструктор, позволяющий использовать остальные подходы.	Мощность и гибкость.	Относительная сложность (особенно — в создании фреймворка).
5	Запись и воспроизведение (Record & Playback).	Средство автоматизации записывает действия тестировщика и может воспроизвести их, управляя тестируемым приложением.	Простота, высокая скорость создания тест-кейсов.	Крайне низкое качество, линейность, неподдерживаемость тест-кейсов. Требуется серьёзная доработка полученного кода.

6	Тестирование под управлением поведением{87} (BDT).	Развитие идей тестирования под управлением данными и ключевыми словами. Отличие — в концентрации на бизнес-сценариях без выполнения мелких проверок.	Высокое удобство проверки высокоуровневых пользовательских сценариев.	Такие тест-кейсы пропускают большое количество функциональных и нефункциональных дефектов, а потому должны быть дополнены классическими более низкоуровневыми тест-кейсами.

На текущем этапе развития тестирования представленные в таблице 3.2.a технологии иерархически можно изобразить следующей схемой (см. рисунок 3.2.b):



Рисунок 3.2.b — Иерархия технологий автоматизации тестирования

Сейчас мы рассмотрим эти технологии подробнее и на примерах, но сначала стоит упомянуть один основополагающий подход, который находит применение практически в любой технологии автоматизации, — функциональную декомпозицию.

Функциональная декомпозиция



Функциональная декомпозиция (functional decomposition[367]) — процесс определения функции через её разделение на несколько низкоуровневых подфункций.

Функциональная декомпозиция активно используется как в программировании, так и в автоматизации тестирования с целью упрощения решения поставленных задач и получения возможности повторного использования фрагментов кода для решения различных высокоуровневых задач.

Рассмотрим пример (рисунок 3.2.c): легко заметить, что часть низкоуровневых действий (поиск и заполнение полей, поиск и нажатие кнопок) универсальна и может быть использована при решении других задач (например, регистрация, оформление заказа и т.д.).

[367] **Functional decomposition.** The process of defining lower-level functions and sequencing relationships. [«System Engineering Fundamentals», Defense Acquisition University Press]

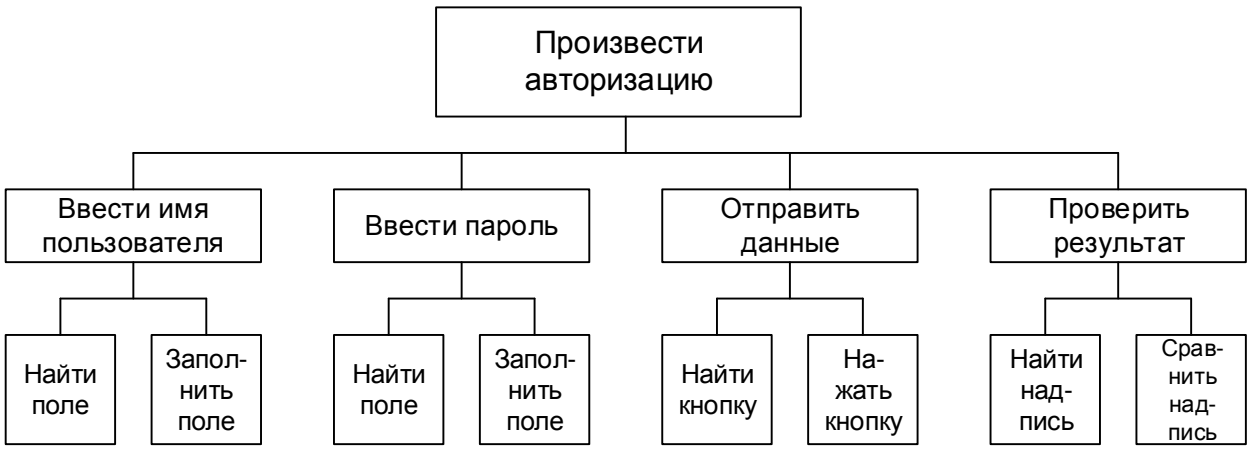


Рисунок 3.2.c — Пример функциональной декомпозиции в программировании и тестировании

Применение функциональной декомпозиции позволяет не только упростить процесс решения поставленных задач, но и избавиться от необходимости самостоятельной реализации действий на самом низком уровне, т.к. они, как правило, уже решены авторами соответствующих библиотек или фреймворков.

Возвращаемся к технологиям автоматизации тестирования.

Частные решения

Иногда перед тестировщиком возникает уникальная (в том плане, что такой больше не будет) задача, для решения которой нет необходимости использовать мощные инструментальные средства, а достаточно написать небольшую программу на любом из высокоуровневых языков программирования (Java, C#, PHP и т.д.) или даже воспользоваться возможностями командных файлов операционной системы или подобными тривиальными решениями.

Ярчайшим примером такой задачи и её решения является автоматизация дымового тестирования нашего «Конвертера файлов» (код командных файлов для Windows и Linux приведён в соответствующем приложении{271}). Также сюда можно отнести задачи вида:

- Подготовить базу данных, наполнив её тестовыми данными (например, добавить в систему миллион пользователей со случайными именами).
- Подготовить файловую систему (например, создать структуру каталогов и набор файлов для выполнения тест-кейсов).
- Перезапустить набор серверов и/или привести их в требуемое состояние.

Удобство частных решений состоит в том, что их можно реализовать быстро, просто, «вот прямо сейчас». Но у них есть и огромный недостаток — это «кустарные решения», которыми может воспользоваться всего пара человек. И при появлении новой задачи, даже очень похожей на ранее решённую, скорее всего, придётся всё автоматизировать заново.

Более подробно преимущества и недостатки частных решений в автоматизации тестирования показаны в таблице 3.2.b.

Таблица 3.2.b — Преимущества и недостатки частных решений в автоматизации тестирования

Преимущества	Недостатки
• Быстрота и простота реализации. • Возможность использования любых доступных инструментов, которыми тестировщик умеет пользоваться. • Эффект от использования наступает незамедлительно. • Возможность нахождения очень эффективных решений в случае, когда основные инструменты, используемые на проекте для автоматизации тестирования, оказываются малопригодными для выполнения данной отдельной задачи. • Возможность быстрого создания и оценки прототипов перед применением более тяжеловесных решений.	• Отсутствие универсальности и, как следствие, невозможность или крайняя сложность повторного использования (адаптации для решения других задач). • Разрозненность и несогласованность решений между собой (разные подходы, технологии, инструменты, принципы решения). • Крайне высокая сложность развития, поддержки и сопровождения таких решений (чаще всего, кроме самого автора никто вообще не понимает, что и зачем было сделано, и как оно работает). • Является признаком «кустарного производства», не приветствуется в промышленной разработке программ.

Тестирование под управлением данными (DDT)

Обратите внимание, как много раз в командных файлах{271} повторяются строки, выполняющие одно и то же действие над набором файлов (и нам ещё повезло, что файлов немного). Ведь куда логичнее было бы каким-то образом подготовить список файлов и просто передать его на обработку. Это и будет тестированием под управлением данными. В качестве примера приведём небольшой скрипт на PHP, который читает CSV-файл с тестовыми данными (именами сравниваемых файлов) и выполняет сравнение файлов.

```
function compare_list_of_files($file_with_csv_data)
{
	$data = file($file_with_csv_data);
	foreach ($data as $line)
	{
		$parsed = str_csv($line);
		if (md5_file($parsed[0]) === md5_file($parsed[1])) {
			file_put_contents('smoke_test.log',
			"OK! File '".$parsed[0]."' was processed correctly!\n");
		} else {
			file_put_contents('smoke_test.log',
			"ERROR! File '".$parsed[0]."' was NOT
			processed correctly!\n");
		}
	}
}
```

Пример CSV-файла (первые две строки):

```
Test_ETALON/«Мелкий» эталон WIN1251.txt,OUT/«Мелкий» файл в WIN1251.txt
Test_ETALON/«Средний» эталон CP866.txt,OUT/«Средний» файл CP866.txt
```

Теперь нам достаточно подготовить CSV-файл с любым количеством имён сравниваемых файлов, а код тест-кейса не увеличится ни на байт.

К другим типичным примерам использования тестирования под управлением данными относится:

- Проверка авторизации и прав доступа на большом наборе имён пользователей и паролей.

- Многократное заполнение полей форм разными данными и проверка реакции приложения.
- Выполнение тест-кейса на основе данных, полученных с помощью комбинаторных техник (пример таких данных представлен в соответствующем приложении{280}).

Данные для рассматриваемого подхода к организации тест-кейсов могут поступать из файлов, баз данных и иных внешних источников или даже генерироваться в процессе выполнения тест-кейса (см. описание источников данных для автоматизированного тестирования{254}).

Преимущества и недостатки тестирования под управлением данными показаны в таблице 3.2.c.

Таблица 3.2.c — Преимущества и недостатки тестирования под управлением данными

Преимущества	Недостатки
• Устранение избыточности кода тест-кейсов. • Удобное хранение и понятный человеку формат данных. • Возможность поручения генерации данных сотруднику, не имеющему навыков программирования. • Возможность использования одного и того же набора данных для выполнения разных тест-кейсов. • Возможность повторного использования набора данных для решения новых задач. • Возможность использования одного и того же набора данных в одном и том же тест-кейсе, но реализованном под разными платформами.	• При изменении логики поведения тест-кейса его код всё равно придётся переписывать. • При неудачно выбранном формате представления данных резко снижается их понятность для неподготовленного специалиста. • Необходимость использования технологий генерации данных. • Высокая сложность кода тест-кейса в случае сложных неоднородных данных. • Риск неверной работы тест-кейсов в случае, когда несколько тест-кейсов работают с одним и тем же набором данных, и он был изменён таким образом, на который не были рассчитаны некоторые тест-кейсы. • Слабые возможности по сбору данных в случае обнаружения дефектов. • Качество тест-кейса зависит от профессионализма сотрудника, реализующего код тест-кейса.

Тестирование под управлением ключевыми словами

Логическим развитием идеи о вынесении вовне тест-кейса данных является вынесение вовне тест-кейса команд (описания действий). Разовьём только что показанный пример, добавив в CSV-файл ключевые слова, являющиеся описанием выполняемой проверки:

- moved — файл отсутствует в каталоге-источнике и присутствует в каталоге-приёмнике;
- intact — файл присутствует в каталоге-источнике и отсутствует в каталоге-приёмнике;
- equals — содержимое файлов идентично.

```
function execute_test($scenario)
{
	$data = file($scenario);
	foreach ($data as $line)
	{
		$parsed = str_csv($line);
		switch ($parsed[0])
		{
```

```
                    // Проверка перемещения файла
                    case 'moved':
                          if (is_file($parsed[1]))&&(!is_file($parsed[2])) {
                                file_put_contents('smoke_test.log',
                                "OK! '".$parsed[0]."' file was processed!\n");
                          } else {
                                file_put_contents('smoke_test.log',
                                "ERROR! '".$parsed[0]."' file was
                                NOT processed!\n");
                          }
                    break;

                    // Проверка отсутствия перемещения файла
                    case 'intact':
                          if (!is_file($parsed[1]))||(is_file($parsed[2])) {
                                file_put_contents('smoke_test.log',
                                "OK! '".$parsed[0]."' file was processed!\n");
                          } else {
                                file_put_contents('smoke_test.log',
                                "ERROR! '".$parsed[0]."' file was
                                NOT processed!\n");
                          }
                    break;

                    // Сравнение файлов
                    case 'equals':
                          if (md5_file($parsed[1]) === md5_file($parsed[2])) {
                                file_put_contents('smoke_test.log',
                                "OK! File '".$parsed[0]."' Was
                                processed correctly!\n");
                          } else {
                                file_put_contents('smoke_test.log',
                                "ERROR! File '".$parsed[0]."' Was
                                NOT processed correctly!\n");
                          }
                    break;
              }
        }
}
```

Пример CSV-файла (первые пять строк):

```
moved,IN/«Мелкий» эталон WIN1251.txt,OUT/«Мелкий» файл в WIN1251.txt
moved,IN/«Средний» эталон CP866.txt,OUT/«Средний» файл CP866.txt
intact,IN/Картинка.jpg,OUT/Картинка.jpg
equals,Test_ETALON/«Мелкий» эталон WIN1251.txt,OUT/«Мелкий» файл в WIN1251.txt
equals,Test_ETALON/«Средний» эталон CP866.txt,OUT/«Средний» файл CP866.txt
```

Ярчайшим примером инструментального средства автоматизации тестирования, идеально следующего идеологии тестирования под управлением ключевыми словами, является Selenium IDE[366], в котором каждая операция тест-кейса описывается в виде:

Действие (ключевое слово)	Необязательный параметр 1	Необязательный параметр 2

Тестирование под управлением ключевыми словами стало тем переломным моментом, начиная с которого стало возможным привлечение к автоматизации тестирования нетехнических специалистов. Согласитесь, что нет необходимости в знаниях программирования и тому подобных технологий, чтобы наполнять подобные только что показанному CSV-файлы или (что очень часто практикуется) XLSX-файлы.

Вторым естественным преимуществом тестирования под управлением ключевыми словами (хотя она вполне характерна и для тестирования под управлением данными) стала возможность использования различных инструментов одними и теми же наборами команд и данных. Так, например, ничто не мешает нам взять показанные CSV-файлы и написать новую логику их обработки не на PHP, а на C#, Java, Python или даже с использованием специализированных средств автоматизации тестирования.

Преимущества и недостатки тестирования под управлением ключевыми словами показаны в таблице 3.2.d.

Таблица 3.2.d — Преимущества и недостатки тестирования под управлением ключевыми словами

Преимущества	Недостатки
• Максимальное устранение избыточности кода тест-кейсов. • Возможность построения мини-фреймворков, решающих широкой спектр задач. • Повышение уровня абстракции тест-кейсов и возможность их адаптации для работы с разными техническими решениями. • Удобное хранение и понятный человеку формат данных и команд тест-кейса. • Возможность поручения описания логики тест-кейса сотруднику, не имеющему навыков программирования. • Возможность повторного использования для решения новых задач. • Расширяемость (возможность добавления нового поведения тест-кейса на основе уже реализованного поведения).	• Высокая сложность (и, возможно, длительность) разработки. • Высокая вероятность наличия ошибок в коде тест-кейса. • Высокая сложность (или невозможность) выполнения низкоуровневых операций, если код тест-кейса не поддерживает соответствующие команды. • Эффект от использования данного подхода наступает далеко не сразу (сначала идёт длительный период разработки и отладки самих тест-кейсов и вспомогательной функциональности). • Для реализации данного подхода требуется наличие высококвалифицированного персонала. • Необходимо обучить персонал языку ключевых слов, используемых в тест-кейсах.

Использование фреймворков

Фреймворки автоматизации тестирования представляют собой ничто иное, как успешно развившиеся и ставшие популярными решения, объединяющие в себе лучшие стороны тестирования под управлением данными, ключевыми словами и возможности реализации частных решений на высоком уровне абстракции.

Фреймворков автоматизации тестирования очень много, они очень разные, но их объединяет несколько общих черт:

- высокая абстракция кода (нет необходимости описывать каждое элементарное действие) с сохранением возможности выполнения низкоуровневых действий;
- универсальность и переносимость используемых подходов;
- достаточно высокое качество реализации (для популярных фреймворков).

Как правило, каждый фреймворк специализируется на своём виде тестирования, уровне тестирования, наборе технологий. Существуют фреймворки для модульного тестирования (например, семейство xUnit), тестирования веб-приложений (например, семейство Selenium), тестирования мобильных приложений, тестирования производительности и т.д.

Существуют бесплатные и платные фреймворки, оформленные в виде библиотек на некотором языке программирования или в виде привычных приложений с графическим интерфейсом, узко- и широко специализированные и т.д.



К сожалению, подробное описание даже одного фреймворка может по объёму быть сопоставимо со всем текстом данной книги. Но если вам интересно, начните хотя бы с изучения Selenium WebDriver[368].

Преимущества и недостатки фреймворков автоматизации тестирования показаны в таблице 3.2.e.

Таблица 3.2.e — Преимущества и недостатки фреймворков автоматизации тестирования

Преимущества	Недостатки
• Широкое распространение. • Универсальность в рамках своего набора технологий. • Хорошая документация и большое сообщество специалистов, с которыми можно проконсультироваться. • Высокий уровень абстракции. • Наличие большого набора готовых решений и описаний соответствующих лучших практик применения того или иного фреймворка для решения тех или иных задач.	• Требуется время на изучение фреймворка. • В случае написания собственного фреймворка де-факто получается новый проект по разработке ПО. • Высокая сложность перехода на другой фреймворк. • В случае прекращения поддержки фреймворка тест-кейсы рано или поздно придётся переписывать с использованием нового фреймворка. • Высокий риск выбора неподходящего фреймворка.

Запись и воспроизведение (Record & Playback)

Технология записи и воспроизведения (Record & Playback) стала актуальной с появлением достаточно сложных средств автоматизации, обеспечивающих глубокое взаимодействие с тестируемым приложением и операционной системой. Использование этой технологии, как правило, сводится к следующим основным шагам:

1. Тестировщик вручную выполняет тест-кейс, а средство автоматизации записывает все его действия.
2. Результаты записи представляются в виде кода на высокоуровневом языке программирования (в некоторых средствах — специально разработанном).
3. Тестировщик редактирует полученный код.
4. Готовый код автоматизированного тест-кейса выполняется для проведения тестирования в автоматизированном режиме.



Возможно, вам приходилось записывать макросы в офисных приложениях. Это тоже технология записи и воспроизведения, только применённая для автоматизации решения офисных задач.

Сама технология при достаточно высокой сложности внутренней реализации очень проста в использовании и по самой своей сути, потому часто применяется для обучения начинающих специалистов по автоматизации тестирования. Её основные преимущества и недостатки показаны в таблице 3.2.f.

Таблица 3.2.f — Преимущества и недостатки технологии записи и воспроизведения

Преимущества	Недостатки
• Предельная простота освоения (достаточно буквально нескольких минут, чтобы начать использовать данную технологию). • Быстрое создание «скелета тест-кейса» за счёт записи ключевых действий с тестируемым приложением.	• Линейность тест-кейсов: в записи не будет циклов, условий, вызовов функций и прочих характерных для программирования и автоматизации явлений. • Запись лишних действий (как просто ошибоч-

[368] Selenium WebDriver Documentation [http://docs.seleniumhq.org/docs/03_webdriver.jsp]

• Автоматический сбор технических данных о тестируемом приложении (идентификаторов и локаторов элементов, надписей, имён и т.д.) • Автоматизация рутинных действий (заполнения полей, нажатий на ссылки, кнопки и т.д.) • В отдельных случаях допускает использование тестировщиками без навыков программирования.	ных случайных действий тестировщика с тестируемым приложением, так и (во многих случаях) переключений на другие приложения и работы с ними). • Так называемый «хардкодинг», т.е. запись внутрь кода тест-кейса конкретных значений, что потребует ручной доработки для перевода тест-кейса на технологию тестирования под управлением данными. • Неудобные имена переменных, неудобное оформление кода тест-кейса, отсутствие комментариев и прочие недостатки, усложняющие поддержку и сопровождение тест-кейса в будущем. • Низкая надёжность самих тест-кейсов в силу отсутствия обработки исключений, проверки условий и т.д.

Таким образом, технология записи и воспроизведения не является универсальным средством на все случаи жизни и не может заменить ручную работу по написанию кода автоматизированных тест-кейсов, но в отдельных ситуациях может сильно ускорить решение простых рутинных задач.

Тестирование под управлением поведением

Рассмотренные выше технологии автоматизации максимально сфокусированы на технических аспектах поведения приложения и обладают общим недостатком: с их помощью сложно проверять высокоуровневые пользовательские сценарии (а именно в них и заинтересованы заказчики и конечные пользователи). Этот недостаток призвано исправить тестирование под управлением поведением, в котором акцент делается не на отдельных технических деталях, а на общей работоспособности приложения при решении типичных пользовательских задач.

Такой подход не только упрощает выполнение целого класса проверок, но и облегчает взаимодействие между разработчиками, тестировщиками, бизнес-аналитиками и заказчиком, т.к. в основе подхода лежит очень простая формула «given-when-then»:

- Given («имея, предполагая, при условии») описывает начальную ситуацию, в которой находится пользователь в контексте работы с приложением.
- When («когда») описывает набор действий пользователя в данной ситуации.
- Then («тогда») описывает ожидаемое поведение приложения.

Рассмотрим на примере нашего «Конвертера файлов»:

- **При условии**, что приложение запущено.
- **Когда** я помещаю во входной каталог файл поддерживаемого размера и формата.
- **Тогда** файл перемещается в выходной каталог, а текст внутри файла перекодируется в UTF-8.

Такой принцип описания проверок позволяет даже участникам проекта, не имеющим глубокой технической подготовки, принимать участие в разработке и анализе тест-кейсов, а для специалистов по автоматизации упрощается создание кода автоматизированных тест-кейсов, т.к. такая форма является стандартной, единой и при этом предоставляет достаточно информации для написания высокоуровневых тест-кейсов. Существуют специальные технические решения (например, Behat, JBehave, NBehave, Cucumber), упрощающие реализацию тестирования под управлением поведением.

Преимущества и недостатки тестирования под управлением поведением показаны в таблице 3.2.g.

Таблица 3.2.g — Преимущества и недостатки тестирования под управлением поведением

Преимущества	Недостатки
• Фокусировка на потребностях конечных пользователей. • Упрощение сотрудничества между различными специалистами. • Простота и скорость создания и анализа тест-кейсов (что, в свою очередь, повышает полезный эффект автоматизации и снижает накладные расходы).	• Высокоуровневые поведенческие тест-кейсы пропускают много деталей, а потому могут не обнаружить часть проблем в приложении или не предоставить необходимой для понимания обнаруженной проблемы информации. • В некоторых случаях информации, предоставленной в поведенческом тест-кейсе, недостаточно для его непосредственной автоматизации.



К классическим технологиям автоматизации тестирования также можно отнести разработку под управлением тестированием (Test Driven Development, TDD) с её принципом «красный, зелёный, улучшенный» (Red-Green-Refactor), разработку под управлением поведением (Behavior Driven Development), модульное тестирование (Unit Testing) и т.д. Но эти технологии уже находятся на границе тестирования и разработки приложений, потому выходят за рамки данной книги.

3.3. Автоматизация вне прямых задач тестирования

На протяжении данного раздела мы рассматривали, как автоматизация может помочь в создании и выполнении тест-кейсов. Но все те же принципы можно перенести и на остальную работу тестировщика, в которой также бывают длительные и утомительные задачи, рутинные задачи или задачи, требующие предельного внимания, но не связанные с интеллектуальной работой. Всё перечисленное также можно автоматизировать.

Да, это требует технических знаний и первоначальных затрат сил и времени на реализацию, но в перспективе такой подход может экономить до нескольких часов в день. К самым типичным решениям из данной области можно отнести:

- Использование командных файлов для выполнения последовательностей операций — от копирования нескольких файлов из разных каталогов до развёртывания тестового окружения. Даже в рамках многократно рассмотренных примеров по тестированию «Конвертера файлов» запуск приложения через командный файл, в котором указаны все необходимые параметры, избавляет от необходимости вводить их каждый раз вручную.
- Генерация и оформление данных с использованием возможностей офисных приложений, баз данных, небольших программ на высокоуровневых языках программирования. Нет картины печальнее, чем тестировщик, руками нумерующий три сотни строк в таблице.
- Подготовка и оформление технических разделов для отчётов. Можно тратить часы на скрупулёзное вычитывание журналов работы некоего средства автоматизации, а можно один раз написать скрипт, который будет за мгновение готовить документ с аккуратными таблицами и графиками, и останется только запускать этот скрипт и прикреплять результаты его работы к отчёту.
- Управление своим рабочим местом: создание и проверка резервных копий, установка обновлений, очистка дисков от устаревших данных и т.д. и т.п. Компьютер всё это может (и должен!) делать сам, без участия человека.
- Сортировка и обработка почты. Даже раскладывание входящей корреспонденции по подпапкам гарантированно занимает у вас несколько минут в день. Если предположить, что настройка специальных правил в вашем почтовом клиенте сэкономит вам полчаса в неделю, за год экономия составит примерно сутки.
- Виртуализация как способ избавления от необходимости каждый раз устанавливать и настраивать необходимый набор программ. Если у вас есть несколько заранее подготовленных виртуальных машин, их запуск займёт секунды. А в случае необходимости устранения сбоев разворачивание виртуальной машины из резервной копии заменяет весь процесс установки и настройки с нуля операционной системы и всего необходимого программного обеспечения.

Иными словами, автоматизация объективно облегчает жизнь любого ИТ-специалиста, а также расширяет его кругозор, технические навыки и способствует профессиональному росту.