

2.4.2. Тест-кейсы

Терминология и общие сведения

Для начала определимся с терминологией, поскольку здесь есть много путаницы, вызванной разными переводами англоязычных терминов на русский язык и разными традициями в тех или иных странах, фирмах и отдельных командах.

Во главе всего лежит термин «тест». Официальное определение звучит так.



Тест (test²⁷⁹) — набор из одного или нескольких тест-кейсов.

Поскольку среди всех прочих терминов этот легче и быстрее всего произносить, в зависимости от контекста под ним могут понимать и отдельный пункт чек-листа, и отдельный шаг в тест-кейсе, и сам тест-кейс, и набор тест-кейсов и... продолжать можно долго. Главное здесь одно: если вы слышите или видите слово «тест», воспринимайте его в контексте.

Теперь рассмотрим самый главный для нас термин — «тест-кейс».



Тест-кейс (test case²⁸⁰) — набор входных данных, условий выполнения и ожидаемых результатов, разработанный с целью проверки того или иного свойства или поведения программного средства.

Под тест-кейсом также может пониматься соответствующий документ, представляющий формальную запись тест-кейса.

Мы ещё вернёмся к этой мысли⁽¹⁴³⁾, но уже сейчас критически важно понять и запомнить: если у тест-кейса не указаны входные данные, условия выполнения и ожидаемые результаты, и/или не ясна цель тест-кейса — это плохой тест-кейс (иногда он не имеет смысла, иногда его и вовсе невозможно выполнить).

Примечание: иногда термин «test case» на русский язык переводят как «тестовый случай». Это вполне адекватный перевод, но из-за того, что «тест-кейс» короче произносить, наибольшее распространение получил именно этот вариант.



Остальные термины, связанные с тестами, тест-кейсами и тестовыми сценариями, на данном этапе можно прочитать просто в ознакомительных целях. Если вы откроете ISTQB-гlossарий на букву «Т», вы увидите огромное количество терминов, тесно связанных друг с другом перекрёстными ссылками: на начальном этапе изучения тестирования нет необходимости глубоко рассматривать их все, однако некоторые всё же заслуживают внимания. Они представлены ниже.

Высокоуровневый тест-кейс (high level test case²⁸¹) — тест-кейс без конкретных входных данных и ожидаемых результатов.

Как правило, ограничивается общими идеями и операциями, схож по своей сути с подробно описанным пунктом чек-листа. Достаточно часто встречается в интеграционном тестировании⁽⁷¹⁾ и системном тестировании⁽⁷¹⁾, а также на уровне дымового тестирования⁽⁷³⁾. Может служить отправной точкой для проведения исследовательского тестирования⁽⁷⁸⁾ или для создания низкоуровневых тест-кейсов.

²⁷⁹ **Test.** A set of one or more test cases. [ISTQB Glossary]

²⁸⁰ **Test case.** A set of input values, execution preconditions, expected results and execution postconditions, developed for a particular objective or test condition, such as to exercise a particular program path or to verify compliance with a specific requirement. [ISTQB Glossary]

²⁸¹ **High level test case (logical test case).** A test case without concrete (implementation level) values for input data and expected results. Logical operators are used; instances of the actual values are not yet defined and/or available. [ISTQB Glossary]

Низкоуровневый тест-кейс (low level test case²⁸²) — тест-кейс с конкретными входными данными и ожидаемыми результатами.

Представляет собой «полностью готовый к выполнению» тест-кейс и вообще является наиболее классическим видом тест-кейсов. Начинающих тестировщиков чаще всего учат писать именно такие тесты, т.к. прописать все данные подробно — намного проще, чем понять, какой информацией можно пренебречь, при этом не снизив ценность тест-кейса.

Спецификация тест-кейса (test case specification²⁸³) — документ, описывающий набор тест-кейсов (включая их цели, входные данные, условия и шаги выполнения, ожидаемые результаты) для тестируемого элемента (test item²⁸⁴, test object²⁸⁵).

Спецификация теста (test specification²⁸⁶) — документ, состоящий из спецификации тест-дизайна (test design specification²⁸⁷), спецификации тест-кейса (test case specification²⁸³) и/или спецификации тест-процедуры (test procedure specification²⁸⁸).

Тест-сценарий (test scenario²⁸⁹, test procedure specification, test script) — документ, описывающий последовательность действий по выполнению теста (также известен как «тест-скрипт»).



Внимание! Из-за особенностей перевода очень часто под термином «тест-сценарий» («тестовый сценарий») имеют в виду набор тест-кейсов⁽¹³⁷⁾.

Цель написания тест-кейсов

Тестирование можно проводить и без тест-кейсов (не нужно, но можно; да, эффективность такого подхода варьируется в очень широком диапазоне в зависимости от множества факторов). Наличие же тест-кейсов позволяет:

- Структурировать и систематизировать подход к тестированию (без чего крупный проект почти гарантированно обречён на провал).
- Вычислять метрики тестового покрытия (test coverage²⁹⁰ metrics) и принимать меры по его увеличению (тест-кейсы здесь являются главным источником информации, без которого существование подобных метрик теряет смысл).
- Отслеживать соответствие текущей ситуации плану (сколько примерно понадобится тест-кейсов, сколько уже есть, сколько выполнено из запланированного на данном этапе количества и т.д.).
- Уточнить взаимопонимание между заказчиком, разработчиками и тестиров-

²⁸² **Low level test case.** A test case with concrete (implementation level) values for input data and expected results. Logical operators from high level test cases are replaced by actual values that correspond to the objectives of the logical operators. [ISTQB Glossary]

²⁸³ **Test case specification.** A document specifying a set of test cases (objective, inputs, test actions, expected results, and execution preconditions) for a test item. [ISTQB Glossary]

²⁸⁴ **Test item.** The individual element to be tested. There usually is one test object and many test items. [ISTQB Glossary]

²⁸⁵ **Test object.** The component or system to be tested. [ISTQB Glossary]

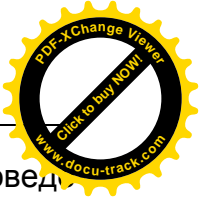
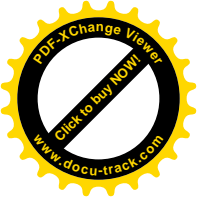
²⁸⁶ **Test specification.** A document that consists of a test design specification, test case specification and/or test procedure specification. [ISTQB Glossary]

²⁸⁷ **Test design specification.** A document specifying the test conditions (coverage items) for a test item, the detailed test approach and identifying the associated high level test cases. [ISTQB Glossary]

²⁸⁸ **Test procedure specification (test procedure).** A document specifying a sequence of actions for the execution of a test. Also known as test script or manual test script. [ISTQB Glossary]

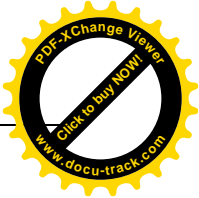
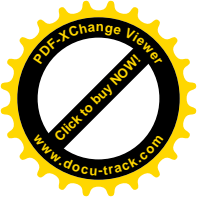
²⁸⁹ **Test scenario.** A document specifying a sequence of actions for the execution of a test. Also known as test script or manual test script. [ISTQB Glossary]

²⁹⁰ **Coverage (test coverage).** The degree, expressed as a percentage, to which a specified coverage item (an entity or property used as a basis for test coverage, e.g. equivalence partitions or code statements) has been exercised by a test suite. [ISTQB Glossary]



щиками (тест-кейсы зачастую намного более наглядно показывают поведение приложения, чем это отражено в требованиях).

- Хранить информацию для длительного использования и обмена опытом между сотрудниками и командами (или как минимум — не пытаться удержать в голове сотни страниц текста).
- Проводить регрессионное тестирование^[80] и повторное тестирование^[81] (которые без тест-кейсов было бы вообще невозможно выполнить).
- Повышать качество требований (мы это уже рассматривали: написание чек-листов и тест-кейсов — хорошая техника тестирования требований^[46]).
- Быстро вводить в курс дела нового сотрудника, недавно подключившегося к проекту.



2.4.3. Атрибуты (поля) тест-кейса

Как уже было сказано выше, термин «тест-кейс» может относиться к формальной записи тест-кейса в виде технического документа. Эта запись имеет общепринятую структуру, компоненты которой называются атрибутами (полями) тест-кейса.

В зависимости от инструмента управления тест-кейсам внешний вид их записи может немного отличаться, могут быть добавлены или убраны отдельные поля, но концепция остаётся неизменной.

Общий вид всей структуры тест-кейса представлен на рисунке 2.4.а.

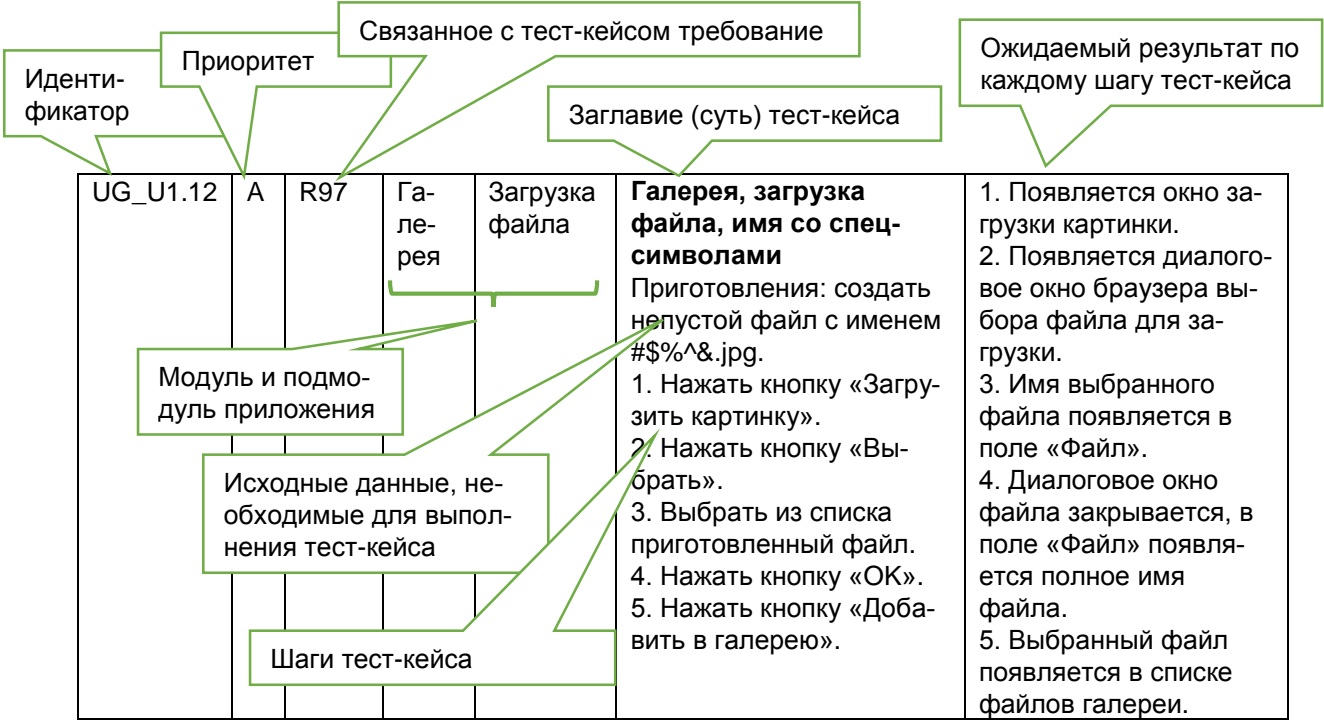


Рисунок 2.4.а — Общий вид тест-кейса

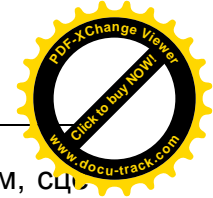
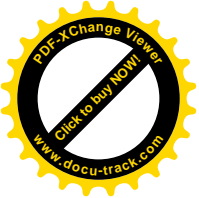
Теперь рассмотрим каждый атрибут подробно.

Идентификатор (identifier) представляет собой уникальное значение, позволяющее однозначно отличить один тест-кейс от другого и используемое во всевозможных ссылках. В общем случае идентификатор тест-кейса может представлять собой просто уникальный номер, но (если позволяет инструментальное средство управления тест-кейсами) может быть и куда сложнее: включать префиксы, суффиксы и иные осмысленные компоненты, позволяющие быстро определить цель тест-кейса и часть приложения (или требований), к которой он относится.

Приоритет (priority) показывает важность тест-кейса. Он может быть выражен буквами (A, B, C, D, E), цифрами (1, 2, 3, 4, 5), словами («крайне высокий», «высокий», «средний», «низкий», «крайне низкий») или иным удобным способом. Количество градаций также не фиксировано, но чаще всего лежит в диапазоне от трёх до пяти.

Приоритет тест-кейса может коррелировать с:

- важностью требования, пользовательского сценария⁽¹³⁷⁾ или функции, с которыми связан тест-кейс;
- потенциальной важностью дефекта⁽¹⁶⁹⁾, на поиск которого направлен тест-кейс;



- степенью риска, связанного с проверяемым тест-кейсом требованием, сценарием или функцией.

Основная задача этого атрибута — упрощение распределения внимания и усилий команды (более высокоприоритетные тест-кейсы получают их больше), а также упрощение планирования и принятия решения о том, чем можно пожертвовать в некоей форс-мажорной ситуации, не позволяющей выполнить все запланированные тест-кейсы.

Связанное с тест-кейсом требование (requirement) показывает то основное требование, проверке выполнения которого посвящён тест-кейс (основное — потому, что один тест-кейс может затрагивать несколько требований). Наличие этого поля улучшает такое свойство тест-кейса, как прослеживаемость^[135].

Частые вопросы, связанные с заполнением этого поля, таковы:

- Можно ли его оставить пустым? Да. Тест-кейс вполне мог разрабатываться вне прямой привязки к требованиям, и (пока?) значение этого поля определить сложно. Хотя такой вариант и не считается хорошим, он достаточно распространён.
- Можно ли в этом поле указывать несколько требований? Да, но чаще всего стараются выбрать одно самое главное или «более высокоуровневое» (например, вместо того, чтобы перечислять R56.1, R56.2, R56.3 и т.д., можно просто написать R56). Чаще всего в инструментах управления тестами это поле представляет собой выпадающий список, где можно выбрать только одно значение, и этот вопрос становится неактуальным. К тому же многие тест-кейсы всё же направлены на проверку строго одного требования, и для них этот вопрос также неактуален.

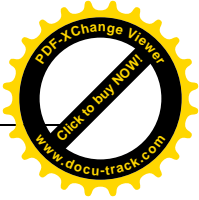
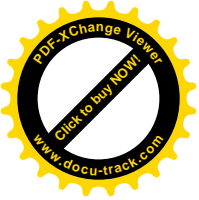
Модуль и подмодуль приложения (module and submodule) указывают на части приложения, к которым относится тест-кейс, и позволяют лучше понять его цель.

Идея деления приложения на модули и подмодули проистекает из того, что в сложных системах практически невозможно охватить взглядом весь проект целиком, и вопрос «как протестировать это приложение» становится недопустимо сложным. Тогда приложение логически разделяется на компоненты (модули), а те, в свою очередь, на более мелкие компоненты (подмодули). И вот уже для таких небольших частей приложения придумать чек-листы и создать хорошие тест-кейсы становится намного проще.

Как правило, иерархия модулей и подмодулей создаётся как единый набор для всей проектной команды, чтобы исключить путаницу из-за того, что разные люди будут использовать разные подходы к такому разделению или даже просто разные названия одних и тех же частей приложения.

Теперь — самое сложное: как выбираются модули и подмодули. В реальности проще всего отталкиваться от архитектуры и дизайна приложения. Например, в уже знакомом нам приложении^[54] можно выделить такую иерархию модулей и подмодулей:

- Механизм запуска:
 - механизм анализа параметров;
 - механизм сборки приложения;
 - механизм обработки ошибочных ситуаций.
- Механизм взаимодействия с файловой системой:
 - механизм обхода дерева SOURCE_DIR;
 - механизм обработки ошибочных ситуаций.
- Механизм преобразования файлов:



- механизм определения кодировок;
 - механизм преобразования кодировок;
 - механизм обработки ошибочных ситуаций.
- Механизм ведения журнала:
 - механизм записи журнала;
 - механизм отображения журнала в консоли;
 - механизм обработки ошибочных ситуаций.

Согласитесь, что такие длинные названия с постоянно повторяющимся словом «механизм» читать и запоминать сложно. Перепишем:

- Стартер:
 - анализатор параметров;
 - сборщик приложения;
 - обработчик ошибок.
- Сканер:
 - обходчик;
 - обработчик ошибок.
- Преобразователь:
 - детектор;
 - конвертер;
 - обработчик ошибок.
- Регистратор:
 - дисковый регистратор;
 - консольный регистратор;
 - обработчик ошибок.

Но что делать, если мы не знаем «внутренностей» приложения (или не очень разбираемся в программировании)? Модули и подмодули можно выделять на основе графического интерфейса пользователя (крупные области и элементы внутри них), на основе решаемых приложением задач и подзадач и т.д. Главное, чтобы эта логика была одинаковым образом применена ко всему приложению.



Внимание! Частая ошибка! Модуль и подмодуль приложения — это НЕ действия, это именно структурные части, «куски» приложения. В заблуждение вас могут ввести такие названия, как, например, «печать, настройка принтера» (но здесь имеются в виду именно части приложения, отвечающие за печать и за настройку принтера (и названы они отглагольными существительными), а не процесс печати или настройки принтера).

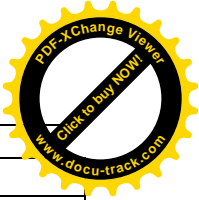
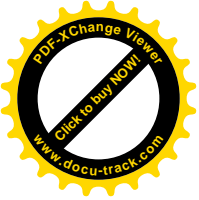
Сравните (на примере человека): «дыхательная система, лёгкие» — это модуль и подмодуль, а «дыхание», «сопение», «чихание» — нет; «голова, мозг» — это модуль и подмодуль, а «кивание», «думание» — нет.

Наличие полей «Модуль» и «Подмодуль» улучшает такое свойство тест-кейса, как прослеживаемость⁽¹³⁵⁾.

Заглавие (суть) тест-кейса (title) призвано упростить быстрое понимание основной идеи тест-кейса без обращения к его остальным атрибутам. Именно это поле является наиболее информативным при просмотре списка тест-кейсов.

Сравните.

Плохо	Хорошо
Тест 1	Запуск, одна копия, верные параметры
Тест 2	Запуск одной копии с неверными путями




Атрибуты (поля) тест-кейса

Тест 78 (улучшенный)	Запуск, много копий, без конфликтов
Остановка	Остановка по Ctrl+C
Закрытие	Остановка закрытием консоли
...	...

Заглавие тест-кейса может быть полноценным предложением, фразой, набором словосочетаний — главное, чтобы выполнялись следующие условия:

- Информативность.
- Хотя бы относительная уникальность (чтобы не путать разные тест-кейсы).




Внимание! Частая ошибка! Если инструмент управления тест-кейсами не требует писать заглавие, его **всё равно надо писать**. Тест-кейсы без заглавий превращаются в мешанину информации, использование которой сопряжено с колоссальными и совершенно бессмысленными затратами.

И ещё одна небольшая мысль, которая может помочь лучше формулировать заглавия. В дословном переводе с английского «test case» обозначает «тестовый случай (ситуация)». Так вот, заглавие как раз и описывает этот случай (ситуацию), т.е. что происходит в тест-кейсе, какую ситуацию он проверяет.

Исходные данные, необходимые для выполнения тест-кейса (precondition, preparation, initial data, setup), позволяют описать всё то, что должно быть подготовлено до начала выполнения тест-кейса, например:

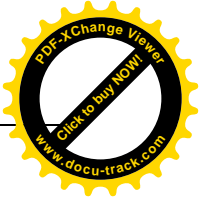
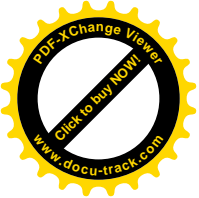
- Состояние базы данных.
- Состояние файловой системы и её объектов.
- Состояние серверов и сетевой инфраструктуры.



ОЧЕНЬ ВАЖНО! Всё, что описывается в этом поле, готовится БЕЗ использования тестируемого приложения, и таким образом, если здесь возникают проблемы, нельзя писать отчёт о дефекте в приложении. Эта мысль очень и очень важна, потому поясним её простым жизненным примером. Представьте, что вы дегустируете конфеты. В поле «исходные данные» можно прописать «купить конфеты таких-то сортов в таком-то количестве». Если таких конфет нет в продаже, если закрыт магазин, если не хватило денег и т.д. — всё это НЕ проблемы вкуса конфет, и нельзя писать отчёт о дефекте конфет вида «конфеты невкусные потому, что закрыт магазин».

Шаги тест-кейса (steps) описывают последовательность действий, которые необходимо реализовать в процессе выполнения тест-кейса. Общие рекомендации по написанию шагов таковы:

- начинайте с понятного и очевидного места, не пишите лишних начальных шагов (запуск приложения, очевидные операции с интерфейсом и т.п.);
- даже если в тест-кейсе всего один шаг, нумеруйте его (иначе возрастает вероятность в будущем случайно «приклеить» описание этого шага к новому тексту);
- если вы пишете на русском языке, используйте безличную форму (например, «открыть», «ввести», «добавить» вместо «откройте», «введите», «добавьте»);
- соотносите степень детализации шагов и их параметров с целью тест-кейса, его сложностью, уровнем^[73] и т.д. — в зависимости от этих и многих других факторов степень детализации может варьироваться от общих идей до предельно чётко прописанных значений и указаний;
- ссылайтесь на предыдущие шаги и их диапазоны для сокращения объёма



текста (например, «повторить шаги 3–5 со значением...»);

- пишите шаги последовательно, без условных конструкций вида «если... то...».



Внимание! Частая ошибка! Категорически запрещено ссылаться на шаги из других тест-кейсов и другие тест-кейсы целиком: если те, другие тест-кейсы будут изменены или удалены, ваш тест-кейс начнёт ссылаться на неверные данные или в пустоту, а если в процессе выполнения те, другие тест-кейсы или шаги приведут к возникновению ошибки, вы не сможете закончить выполнение вашего тест-кейса.

Ожидаемые результаты (expected results) по каждому шагу тест-кейса описывают реакцию приложения на действия, описанные в поле «шаги тест-кейса». Номер шага соответствует номеру результата.

По написанию ожидаемых результатов можно порекомендовать следующее:

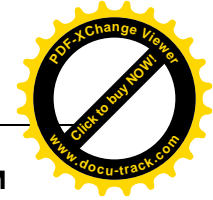
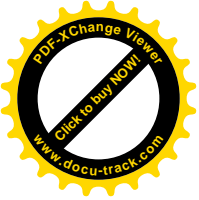
- описывайте поведение системы так, чтобы исключить субъективное толкование (например, «приложение работает верно» — плохо, «появляется окно с надписью...» — хорошо);
- пишите ожидаемый результат по всем шагам без исключения, если у вас есть хоть малейшие сомнения в том, что результат некоего шага будет совершенно тривиальным и очевидным (если вы всё же пропускаете ожидаемый результат для какого-то тривиального действия, лучше оставить в списке ожидаемых результатов пустую строку — это облегчает восприятие);
- пишите кратко, но не в ущерб информативности;
- избегайте условных конструкций вида «если... то...».



Внимание! Частая ошибка! В ожидаемых результатах ВСЕГДА описывается КОРРЕКТНАЯ работа приложения. Нет и не может быть ожидаемого результата в виде «приложение вызывает ошибку в операционной системе и аварийно завершается с потерей всех пользовательских данных».

При этом корректная работа приложения вполне может предполагать отображение сообщений о неверных действиях пользователя или неких критических ситуациях. Так, сообщение «Невозможно сохранить файл по указанному пути: на целевом носителе недостаточно свободного места» — это не ошибка приложения, это его совершенно нормальная и правильная работа. Ошибкой приложения (в этой же ситуации) было бы отсутствие такого сообщения, и/или повреждение, или потеря записываемых данных.

Для более глубокого понимания принципов оформления тест-кейсов рекомендуется прямо сейчас ознакомиться с главой «Типичные ошибки при разработке чек-листов, тест-кейсов и наборов тест-кейсов»⁽¹⁵¹⁾.



2.4.4. Инструментальные средства управления тестированием

Инструментальных средств управления тестированием (test management tool²⁹¹) очень много²⁹², к тому же многие компании разрабатывают свои внутренние средства решения этой задачи.

Не имеет смысла заучивать, как работать с тест-кейсами в том или ином инструментальном средстве — принцип везде един, и соответствующие навыки нарабатываются буквально за пару дней. Что на текущий момент важно понимать, так это общий набор функций, реализуемых такими инструментальными средствами (конечно, то или иное средство может не реализовывать какую-то функцию из этого списка и/или реализовывать не вошедшие в список функции):

- создание тест-кейсов и наборов тест-кейсов;
- контроль версий документов с возможностью определить, кто внёс те или иные изменения, и отменить эти изменения, если требуется;
- формирование и отслеживание реализации плана тестирования, сбор и визуализация разнообразных метрик, генерирование отчётов;
- интеграция с системами управления дефектами, фиксация взаимосвязи между выполнением тест-кейсов и созданными отчётами о дефектах;
- интеграция с системами управления проектами;
- интеграция с инструментами автоматизированного тестирования, управление выполнением автоматизированных тест-кейсов.

Иными словами, хорошее инструментальное средство управления тестированием берёт на себя все рутинные технические операции, которые объективно необходимо выполнять в процессе реализации жизненного цикла тестирования⁽²⁶⁾. Огромным преимуществом также является способность таких инструментальных средств отслеживать взаимосвязи между различными документами и иными артефактами, взаимосвязи между артефактами и процессами и т.д., подчиняя эту логику системе разграничения прав доступа и гарантируя сохранность и корректность информации.

Для общего развития и лучшего закрепления темы об оформлении тест-кейсов⁽¹¹⁷⁾ мы сейчас рассмотрим несколько картинок с формами из разных инструментальных средств.

Здесь вполне сознательно не приведено никакого сравнения или подробного описания — подобных обзоров достаточно в Интернете, и они стремительно устаревают по мере выхода новых версий обозреваемых продуктов.

Но интерес представляют отдельные особенности интерфейса, на которые мы обратим внимание в каждом из примеров (важно: если вас интересует подробное описание каждого поля, связанных с ним процессов и т.д., обратитесь к официальной документации — здесь будут лишь самые краткие пояснения).

²⁹¹ Test management tool. A tool that provides support to the test management and control part of a test process. It often has several capabilities, such as testware management, scheduling of tests, the logging of results, progress tracking, incident management and test reporting. [ISTQB Glossary]

²⁹² «Test management tools» [<http://www.opensourcetesting.org/testmgt.php>]

QAComplete²⁹³

The form contains the following fields and controls:

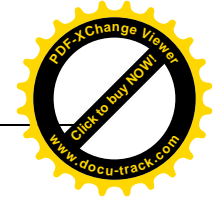
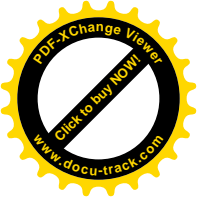
- 1. Id: (Auto Generated)
- 2. Title:
- 3. Priority:
- 4. Folder Name:
- 5. Status:
- 6. Assigned To:
- 7. Last Run Status: (Auto Calculated)
- 8. Last Run Configuration:
- 9. Avg Run Time: (Auto Calculated)
- 10. Last Run Test Set: (Auto Calculated)
- 11. Description: (with a rich text editor toolbar)
- 12. Owner:
- 13. Version:
- 14. Execution Type:
- 15. Test Type:
- 16. Latest Notes: (with an 'Add New Note' link)
- 17. Default Host Name:
- 18. Linked Items: (with a 'Link to Items...' link)
- 19. File Attachments: (with a table of 'Reset' and 'Browse...' buttons)

Buttons at the bottom: Cancel, Submit, Submit/Add another

Рисунок 2.4.b — Создание тест-кейса в QAComplete

1. Id (идентификатор), как видно из соответствующей надписи, автогенерируемый.
2. Title (заглавие), как и в большинстве систем, является обязательным для заполнения.
3. Priority (приоритет) по умолчанию предлагает значения high (высокий), medium (средний), low (низкий).
4. Folder name (расположение) является аналогом полей «Модуль» и «Подмодуль» и позволяет выбрать из выпадающего древовидного списка соответствующее значение, описывающее, к чему относится тест-кейс.
5. Status (статус) показывает текущее состояние тест-кейса: new (новый), approved (утверждён), awaiting approval (на рассмотрении), in design (в разработке), outdated (устарел), rejected (отклонён).
6. Assigned to (исполнитель) указывает, кто в данный момент является «основной рабочей силой» по данному тест-кейсу (или кто должен принять решение о, например, утверждении тест-кейса).
7. Last Run Status (результат последнего запуска) показывает, прошёл ли тест успешно (passed) или завершился неудачей (failed).
8. Last Run Configuration (конфигурация, использованная для последнего запуска) показывает, на какой аппаратно-программной платформе тест-кейс

²⁹³ QAComplete [<http://smartbear.com/product/test-management-tool/qacomplete/>]



- выполнялся в последний раз.
9. Avg Run Time (среднее время выполнения) содержит вычисленное автоматически среднее время, необходимое на выполнение тест-кейса.
 10. Last Run Test Set (в последний раз выполнялся в наборе) содержит информацию о наборе тест-кейсов, в рамках которого тест-кейс выполнялся последний раз.
 11. Last Run Release (последний раз выполнялся на выпуске) содержит информацию о выпуске (билде) программного средства, на котором тест-кейс выполнялся последний раз.
 12. Description (описание) позволяет добавить любую полезную информацию о тест-кейсе (включая особенности выполнения, приготовления и т.д.).
 13. Owner (владелец) указывает на владельца тест-кейса (как правило — его автора).
 14. Execution Type (способ выполнения) по умолчанию предлагает только значение manual (ручное выполнение), но при соответствующих настройках и интеграции с другими продуктами список можно расширить (как минимум добавить automated (автоматизированное выполнение)).
 15. Version (версия) содержит номер текущей версии тест-кейса (фактически — счётчик того, сколько раз тест-кейс редактировали). Вся история изменений сохраняется, предоставляя возможность вернуться к любой из предыдущих версий.
 16. Test Type (вид теста) по умолчанию предлагает такие варианты, как negative (негативный), positive (позитивный), regression (регрессионный), smoke-test (дымный).
 17. Default host name (имя хоста по умолчанию) в основном используется в автоматизированных тест-кейсах и предлагает выбрать из списка имя зарегистрированного компьютера, на котором установлен специальный клиент.
 18. Linked Items (связанные объекты) представляют собой ссылки на требования, отчёты о дефектах и т.д.
 19. File Attachments (вложения) могут содержать тестовые данные, поясняющие изображения, видеоролики и т.д.

Для описания шагов исполнения и ожидаемых результатов после сохранения общего описания тест-кейса становится доступным дополнительный интерфейс:



Рисунок 2.4.с — Добавление шагов тест-кейса в QAComplete

При необходимости можно добавить и настроить свои дополнительные поля, значительно расширяющие исходные возможности системы.

TestLink²⁹⁴

The screenshot shows the 'Create Test Case' interface in TestLink. It includes a 'Test Case Title' field (1), a 'Summary' text area with a rich text editor (2), 'Steps' and 'Expected Results' sections each with a text area and rich text editor (3 and 4), and 'Keywords' section with 'Available Keywords' (5) and 'Assigned Keywords' (6) lists. A 'Create' button is at the top right and bottom right.

Рисунок 2.4.d — Создание тест-кейса в TestLink

1. Title (заглавие) здесь тоже является обязательным для заполнения.
2. Summary (описание) позволяет добавить любую полезную информацию о тест-кейсе (включая особенности выполнения, приготовления и т.д.).
3. Steps (шаги выполнения) позволяет описать шаги выполнения.
4. Expected Results (ожидаемые результаты) позволяет описать ожидаемые результаты, относящиеся к шагам выполнения.
5. Available Keywords (доступные ключевые слова) содержит список ключевых слов, которые можно проассоциировать с тест-кейсом для упрощения классификации и поиска тест-кейсов. Это ещё одна вариация идеи «Модулей» и «Подмодулей» (в некоторых системах реализованы оба механизма).
6. Assigned Keywords (назначенные ключевые слова) содержит список ключевых слов, проассоциированных с тест-кейсом.

Как видите, инструментальные средства управления тест-кейсами могут быть и достаточно минималистичными.

²⁹⁴ TestLink [<http://sourceforge.net/projects/testlink/>]

TestRail²⁹⁵

Add Test Case

Title * 1

Section * 2 **Type *** 3 **Priority *** 4 **Estimate** 5

Milestone **References** 7

Preconditions 6

8

The preconditions of this test case. Reference other test cases with [C#] (e.g. [C17]).

Steps

1

Step Description 9

Expected Result

Expected Result 10

2

Step Description

Expected Result

Expected Result

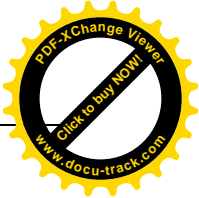
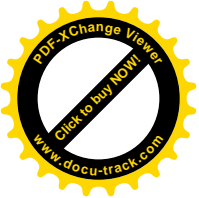
[Add Step](#)

✓ Add Test Case ✗ Cancel

Рисунок 2.4.e — Создание тест-кейса в TestRail

1. Title (заглавие) здесь тоже является обязательным для заполнения.
2. Section (секция) — очередная вариация на тему «Модуль» и «Подмодуль», позволяющая создавать иерархию секций, в которых можно размещать тест-кейсы.
3. Type (тип) здесь по умолчанию предлагает выбрать один из вариантов: automated (автоматизированный), functionality (проверка функциональности), performance (производительность), regression (регрессионный), usability (удобство использования), other (прочее).
4. Priority (приоритет) здесь представлен числами, по которым распределены следующие словесные описания: must test (обязательно выполнять), test if

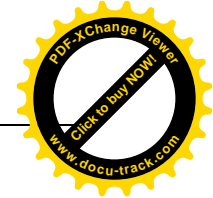
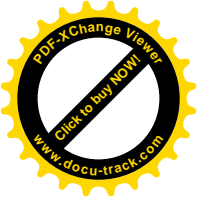
²⁹⁵ TestRail [<http://www.gurock.com/testrail/>]



- time (выполнять, если будет время), don't test (не выполнять).
5. Estimate (оценка) содержит оценку времени, которое необходимо затратить на выполнение тест-кейса.
 6. Milestone (ключевая точка) позволяет указать ключевую точку проекта, к которой данный тест-кейс должен устойчиво показывать положительный результат (выполняться успешно).
 7. References (ссылки) позволяет хранить ссылки на такие артефакты, как требования, пользовательские истории, отчёты о дефектах и иные документы (требуется дополнительной настройки).
 8. Preconditions (приготовления) представляет собой классику описания предварительных условий и необходимых приготовлений к выполнению тест-кейса.
 9. Step Description (описание шага) позволяет добавлять описание отдельного шага тест-кейса.
 10. Expected Results (ожидаемые результаты) позволяет описать ожидаемый результат по каждому шагу.



Задание 2.4.с: изучите ещё 3–5 инструментальных средств управления тест-кейсами, почитайте документацию по ним, посоздавайте в них несколько тест-кейсов.



2.4.5. Свойства качественных тест-кейсов

Даже правильно оформленный тест-кейс может оказаться некачественным, если в нём нарушено одно из следующих свойств.

Правильный технический язык. Это свойство в равной мере относится и к требованиям, и к тест-кейсам, и к отчётам о дефектах — к любой документации. Основные идеи уже были описаны (см. главу «Атрибуты (поля) тест-кейсов»⁽¹¹⁷⁾), а из самого общего и важного напомним и добавим:

- пишите лаконично, но понятно;
- используйте безличную форму глаголов (например, «открыть» вместо «откройте»);
- обязательно указывайте точные имена и технически верные названия элементов приложения;
- не объясняйте базовые принципы работы с компьютером (предполагается, что ваши коллеги знают, что такое, например, «пункт меню» и как с ним работать).

Баланс между специфичностью и общностью. Тест-кейс считается тем более специфичным, чем более детально в нём расписаны конкретные действия, конкретные значения и т.д., т.е. чем в нём больше конкретики. Соответственно, тест-кейс считается тем более общим, чем в нём меньше конкретики.

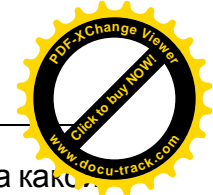
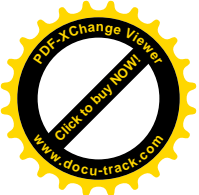
Рассмотрим поля «шаги» и «ожидаемые результаты» двух тест-кейсов (подумайте, какой тест-кейс вы бы посчитали хорошим, а какой — плохим и почему):

Тест-кейс 1:

Шаги	Ожидаемые результаты
Конвертация из всех поддерживаемых кодировок Приготовление: <ul style="list-style-type: none">• Создать папки C:/A, C:/B, C:/C, C:/D.• Разместить в папке C:/D файлы 1.html, 2.txt, 3.md из прилагаемого архива. <ol style="list-style-type: none">1. Запустить приложение, выполнив команду «php converter.php c:/a c:/b c:/c/converter.log».2. Скопировать файлы 1.html, 2.txt, 3.md из папки C:/D в папку C:/A.3. Остановить приложение нажатием Ctrl+C.	<ol style="list-style-type: none">1. Отображается консольный журнал приложения с сообщением «текущее_время started, source dir c:/a, destination dir c:/b, log file c:/c/converter.log», в папке C:/C появляется файл converter.log, в котором появляется запись «текущее_время started, source dir c:/a, destination dir c:/b, log file c:/c/converter.log».2. Файлы 1.html, 2.txt, 3.md появляются в папке C:/A, затем пропадают оттуда и появляются в папке C:/B. В консольном журнале и файле C:/C/converter.log появляются сообщения (записи) «текущее_время processing 1.html (KOI8-R)», «текущее_время processing 2.txt (CP-1251)», «текущее_время processing 3.md (CP-866)».3. В файле C:/C/converter.log появляется запись «текущее_время closed». Приложение завершает работу.

Тест-кейс 2:

Шаги	Ожидаемые результаты
Конвертация из всех поддерживаемых кодировок <ol style="list-style-type: none">1. Выполнить конвертацию трёх файлов допустимого размера в трёх разных кодировках всех трёх допустимых форматов.	<ol style="list-style-type: none">1. Файлы перемещаются в папку-приёмник, кодировка всех файлов меняется на UTF-8.



Если вернуться к вопросу «какой тест-кейс вы бы посчитали хорошим, а какой — плохим и почему», то ответ таков: оба тест-кейса плохие потому, что первый является слишком специфичным, а второй — слишком общим. Можно сказать, что здесь до абсурда доведены идеи низкоуровневых⁽¹¹⁵⁾ и высокоуровневых⁽¹¹⁴⁾ тест-кейсов.

Почему плоха излишняя специфичность (тест-кейс 1):

- при повторных выполнениях тест-кейса всегда будут выполняться строго одни и те же действия со строго одними и теми же данными, что снижает вероятность обнаружения ошибки;
- возрастает время написания, доработки и даже просто прочтения тест-кейса;
- в случае выполнения тривиальных действий опытные специалисты тратят дополнительные мыслительные ресурсы в попытках понять, что же они упустили из виду, т.к. они привыкли, что так описываются только самые сложные и неочевидные ситуации.

Почему плоха излишняя общность (тест-кейс 2):

- тест-кейс сложен для выполнения начинающими тестировщиками или даже опытными специалистами, лишь недавно подключившимися к проекту;
- недобросовестные сотрудники склонны халатно относиться к таким тест-кейсам;
- тестировщик, выполняющий тест-кейс, может понять его иначе, чем было задумано автором (и в итоге будет выполнен фактически совсем другой тест-кейс).

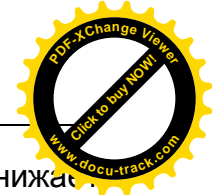
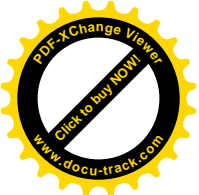
Выход из этой ситуации состоит в том, чтобы придерживаться золотой середины (хотя, конечно же, какие-то тесты будут чуть более специфичными, какие-то — чуть более общими). Вот пример такого срединного подхода:

Тест-кейс 3:

Шаги	Ожидаемые результаты
Конвертация из всех поддерживаемых кодировок Приготовление: <ul style="list-style-type: none">• Создать в корне любого диска четыре отдельные папки для входных файлов, выходных файлов, файла журнала и временного хранения тестовых файлов.• Распаковать содержимое прилагаемого архива в папку для временного хранения тестовых файлов. <ol style="list-style-type: none">1. Запустить приложение, указав в параметрах соответствующие пути из приготовления к тесту (имя файла журнала — произвольное).2. Скопировать файлы из папки для временного хранения в папку для входных файлов.3. Остановить приложение.	<ol style="list-style-type: none">1. Приложение запускается и выводит сообщение о своём запуске в консоль и файл журнала.2. Файлы из папки для входных файлов перемещаются в папку для выходных файлов, в консоль и файле журнала отображаются сообщения о конвертации каждого из файлов с указанием его исходной кодировки.3. Приложение выводит сообщение о завершении работы в файл журнала и завершает работу.

В этом тест-кейсе есть всё необходимое для понимания и выполнения, но при этом он стал короче и проще для выполнения, а отсутствие строго указанных значений приводит к тому, что при многократном выполнении тест-кейса (особенно — разными тестировщиками) конкретные параметры будут менять свои значения, что увеличивает вероятность обнаружения ошибки.

Ещё раз главная мысль: сами по себе специфичность или общность тест-



кейса не являются чем-то плохим, но резкий перекося в ту или иную сторону снижает качество тест-кейса.

Баланс между простотой и сложностью. Здесь не существует академических определений, но принято считать, что простой тест-кейс оперирует одним объектом (или в нём явно виден главный объект), а также содержит небольшое количество тривиальных действий; сложный тест-кейс оперирует несколькими равноправными объектами и содержит много нетривиальных действий.

Преимущества простых тест-кейсов:

- их можно быстро прочесть, легко понять и выполнить;
- они понятны начинающим тестировщикам и новым людям на проекте;
- они делают наличие ошибки очевидным (как правило, в них предполагается выполнение повседневных тривиальных действий, проблемы с которыми видны невооружённым взглядом и не вызывают дискуссий);
- они упрощают начальную диагностику ошибки, т.к. сужают круг поиска.

Преимущества сложных тест-кейсов:

- при взаимодействии многих объектов повышается вероятность возникновения ошибки;
- пользователи, как правило, используют сложные сценарии, а потому сложные тесты более полноценно эмулируют работу пользователей;
- программисты редко проверяют такие сложные случаи (и они совершенно не обязаны это делать).

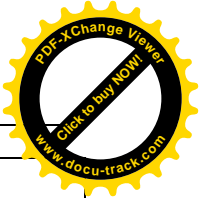
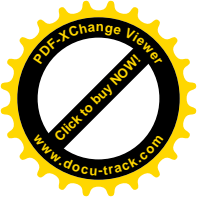
Рассмотрим примеры.

Слишком простой тест-кейс:

Шаги	Ожидаемые результаты
Запуск приложения 1. Запустить приложение.	1. Приложение запускается.

Слишком сложный тест-кейс:

Шаги	Ожидаемые результаты
Повторная конвертация Приготовление: <ul style="list-style-type: none">• Создать в корне любого диска три отдельные папки для входных файлов, выходных файлов, файла журнала.• Подготовить набор из нескольких файлов максимального поддерживаемого размера поддерживаемых форматов с поддерживаемыми кодировками, а также нескольких файлов допустимого размера, но недопустимого формата. <ol style="list-style-type: none">1. Запустить приложение, указав в параметрах соответствующие пути из приготовления к тесту (имя файла журнала — произвольное).2. Скопировать в папку для входных файлов несколько файлов допустимого формата.3. Переместить сконвертированные файлы из папки с результирующими файлами в папку для входных файлов.4. Переместить сконвертированные файлы из папки с результирующими файлами в папку	<ol style="list-style-type: none">2. Файлы постепенно перемещаются из входной в выходную папку, в консоли и файле журнала появляются сообщения об успешной конвертации файлов.3. Файлы постепенно перемещаются из входной в выходную папку, в консоли и файле журнала появляются сообщения об успешной конвертации файлов.5. Файлы постепенно перемещаются из входной в выходную папку, в консоли и файле журнала появляются сообщения об успешной конвертации файлов допустимого формата и сообщения об игнорировании файлов недопустимого формата.6. Файлы постепенно перемещаются из входной в выходную папку, в консоли и файле журнала появляются сообщения об успешной конвертации файлов допустимого формата и сообщения об игнорировании файлов недопустимого формата.



с набором файлов для теста. 5. Переместить все файлы из папки с набором файлов для теста в папку для входных файлов. 6. Переместить сконвертированные файлы из папки с результирующими файлами в папку для входных файлов.	
--	--

Этот тест-кейс одновременно является слишком сложным по избыточности действий и по спецификации лишних данных и операций.

Задание 2.4.d: перепишите этот тест-кейс, устранив его недостатки, но сохранив общую цель (проверку повторной конвертации уже ранее сконвертированных файлов).

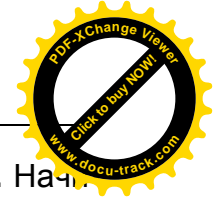
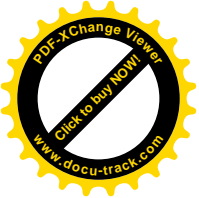
Примером хорошего простого тест-кейса может служить тест-кейс 3⁽¹²⁹⁾ из пункта про специфичность и общность.

Пример хорошего сложного тест-кейса может выглядеть так:

Шаги	Ожидаемые результаты
Много копий приложения, конфликт файловых операций Приготовления: <ul style="list-style-type: none">Создать в корне любого диска три отдельные папки для входных файлов, выходных файлов, файла журнала.Подготовить набор из нескольких файлов максимального поддерживаемого размера поддерживаемых форматов с поддерживаемыми кодировками. <ol style="list-style-type: none">Запустить первую копию приложения, указав в параметрах соответствующие пути из приготовления к тесту (имя файла журнала — произвольное).Запустить вторую копию приложения с теми же параметрами (см. шаг 1).Запустить третью копию приложения с теми же параметрами (см. шаг 1).Изменить приоритет процессов второй (“high”) и третьей (“low”) копий.Скопировать подготовленный набор исходных файлов в папку для входных файлов.	<ol style="list-style-type: none">Все три копии приложения запускаются, в файле журнала появляются последовательно три записи о запуске приложения.Файлы постепенно перемещаются из входной в выходную папку, в консоли и файле журнала появляются сообщения об успешной конвертации файлов, а также (возможно) сообщения вида:<ol style="list-style-type: none">“source file inaccessible, retrying”.“destination file inaccessible, retrying”.“log file inaccessible, retrying”. <p>Ключевым показателем корректной работы является успешная конвертация всех файлов, а также появление в консоли и файле журнала сообщений об успешной конвертации каждого файла (от одной до трёх записей на каждый файл).</p> <p>Сообщения (предупреждения) о недоступности входного файла, выходного файла или файла журнала также являются показателем корректной работы приложения, однако их количество зависит от многих внешних факторов и не может быть спрогнозировано заранее.</p>

Иногда более сложные тест-кейсы являются также и более специфичными, но это лишь общая тенденция, а не закон. Также нельзя по сложности тест-кейса однозначно судить о его приоритете (в нашем примере хорошего сложного тест-кейса он явно будет иметь очень низкий приоритет, т.к. проверяемая им ситуация является искусственной и крайне маловероятной, но бывают и сложные тесты с самым высоким приоритетом).

Как и в случае специфичности и общности, сами по себе простота или сложность тест-кейсов не являются чем-то плохим (более того — рекомендуется начинать разработку и выполнение тест-кейсов с простых, а затем переходить ко всё более и более сложным), однако излишняя простота и излишняя сложность также снижают качество тест-кейса.



«Показательность» (высокая вероятность обнаружения ошибки). Начиная с уровня тестирования критического пути^[73], можно утверждать, что тест-кейс является тем более хорошим, чем он более показателен (с большей вероятностью обнаруживает ошибку). Именно поэтому мы считаем непригодными слишком простые тест-кейсы — они непоказательны.

Пример непоказательного (плохого) тест-кейса:

Шаги	Ожидаемые результаты
Запуск и остановка приложения 1. Запустить приложение с корректными параметрами. 2. Завершить работу приложения.	1. Приложение запускается. 2. Приложение завершает работу.

Пример показательного (хорошего) тест-кейса:

Шаги	Ожидаемые результаты
Запуск с некорректными параметрами, несуществующие пути 1. Запустить приложение со всеми тремя параметрами (SOURCE_DIR, DESTINATION_DIR, LOG_FILE_NAME), значения которых указывают на несуществующие в файловой системе пути (например: z:\src\, z:\dst\, z:\log.txt при условии, что в системе нет логического диска z).	1. В консоли отображаются нижеуказанные сообщения, приложение завершает работу. Сообщения: a. SOURCE_DIR: directory not exists or inaccessible. b. DESTINATION_DIR: directory not exists or inaccessible. c. LOG_FILE_NAME: wrong file name or inaccessible path.

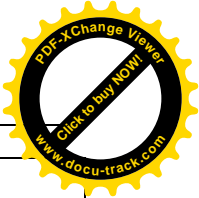
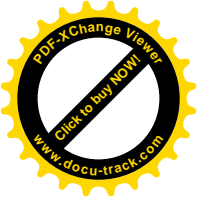
Обратите внимание, что показательный тест-кейс по-прежнему остался достаточно простым, но он проверяет ситуацию, возникновение ошибки в которой несравненно более вероятно, чем в ситуации, описываемой плохим непоказательным тест-кейсом.

Также можно сказать, что показательные тест-кейсы часто выполняют какие-то «интересные действия», т.е. такие действия, которые едва ли будут выполнены просто в процессе работы с приложением (например: «сохранить файл» — это обычное тривиальное действие, которое явно будет выполнено не одну сотню раз даже самими разработчиками, а вот «сохранить файл на носитель, защищённый от записи», «сохранить файл на носитель с недостаточным объёмом свободного пространства», «сохранить файл в папку, к которой нет доступа» — это уже гораздо более интересные и нетривиальные действия).

Последовательность в достижении цели. Суть этого свойства выражается в том, что все действия в тест-кейсе направлены на следование единой логике и достижение единой цели и не содержат никаких отклонений.

Примерами правильной реализации этого свойства могут служить представленные в этом разделе в избытке примеры хороших тест-кейсов. А нарушение может выглядеть так:

Шаги	Ожидаемые результаты
Конвертация из всех поддерживаемых кодировок Приготовление: <ul style="list-style-type: none">Создать в корне любого диска четыре отдельные папки для входных файлов, выходных файлов, файла журнала и временного хранения тестовых файлов.Распаковать содержимое прилагаемого архива в папку для временного хранения тестовых файлов.	1. Приложение запускается и выводит сообщение о своём запуске в консоль и файл журнала. 2. Файлы из папки для входных файлов перемещаются в папку для выходных файлов, в консоли и файле журнала отображаются сообщения о конвертации каждого из файлов с указанием его исходной кодировки. 3. Приложение выводит сообщение о завершении работы в файл журнала и завершает



<ol style="list-style-type: none">1. Запустить приложение, указав в параметрах соответствующие пути из приготовления к тесту (имя файла журнала — произвольное).2. Скопировать файлы из папки для временного хранения в папку для входных файлов.3. Остановить приложение.4. Удалить файл журнала.5. Повторно запустить приложение с теми же параметрами.6. Остановить приложение.	<p>работу.</p> <ol style="list-style-type: none">5. Приложение запускается и выводит сообщение о своём запуске в консоль и заново созданный файл журнала.6. Приложение выводит сообщение о завершении работы в файл журнала и завершает работу.
---	--

Шаги 3–5 никак не соответствуют цели тест-кейса, состоящей в проверке корректности конвертации входных данных, представленных во всех поддерживаемых кодировках.


Отсутствие лишних действий. Чаще всего это свойство подразумевает, что не нужно в шагах тест-кейса долго и по пунктам расписывать то, что можно заменить одной фразой:

Плохо	Хорошо
<ol style="list-style-type: none">1. Указать в качестве первого параметра приложения путь к папке с исходными файлами.2. Указать в качестве второго параметра приложения путь к папке с конечными файлами.3. Указать в качестве третьего параметра приложения путь к файлу журнала.4. Запустить приложение.	<ol style="list-style-type: none">1. Запустить приложение со всеми тремя корректными параметрами (например, <code>c:\src\</code>, <code>c:\dst\</code>, <code>c:\log.txt</code> при условии, что соответствующие папки существуют и доступны приложению).

Вторая по частоте ошибка — начало каждого тест-кейса с запуска приложения и подробного описания по приведению его в то или иное состояние. В наших примерах мы рассматриваем каждый тест-кейс как существующий в единственном виде в изолированной среде, и потому вынуждены осознанно допускать эту ошибку (иначе тест-кейс будет неполным), но в реальной жизни на запуск приложения будут свои тесты, а длинный путь из многих действий можно описать как одно действие, из контекста которого понятно, как это действие выполнить. Следующий пример тест-кейса не относится к нашему «Конвертеру файлов», но очень хорошо иллюстрирует эту мысль:

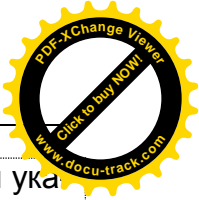
Плохо	Хорошо
<ol style="list-style-type: none">1. Запустить приложение.2. Выбрать в меню пункт «Файл».3. Выбрать подпункт «Открыть».4. Перейти в папку, в которой находится хотя бы один файл формата DOCX с тремя и более страницами.	<ol style="list-style-type: none">1. Открыть DOCX-файл с тремя и более страницами.

И сюда же можно отнести ошибку с повторением одних и тех же приготовлений во множестве тест-кейсов (да, по описанным выше причинам в примерах мы снова вынужденно делаем так, как в жизни делать не надо). Куда удобнее объединить тесты в набор¹³⁷ и указать приготовления один раз, подчеркнув, нужно или нет их выполнять перед каждым тест-кейсом в наборе.



Проблема с подготовительными (и финальными) действиями идеально решена в автоматизированном модульном тестировании²⁹⁶ с использованием фреймворков наподобие JUnit или TestNG — там существует специ-

²⁹⁶ **Unit testing (component testing).** The testing of individual software components. [ISTQB Glossary]



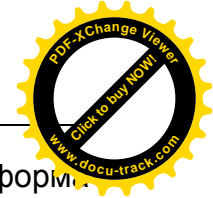
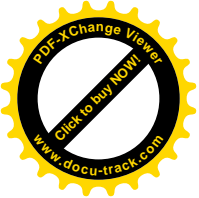
Если вы обнаруживаете несколько тест-кейсов, дублирующих задачи друг друга, лучше всего или удалить все, кроме одного, самого показательного, или перед удалением остальных на их основе доработать этот выбранный самый показательный тест-кейс.

Выдержка из недемонстративного тест-кейса:

Выдержка из демонстративного тест-кейса:

- забыть сконвертировать вручную входной текст в KOI8-R;
- не заметить, что в первый раз расширение начинается с пробела;
- забыть заменить в слове «Пример» буквы «р» на английские;
- из-за расплывчатости формулировки ожидаемого результата принять ошибочное, но выглядящее правдоподобно поведение за верное.

© EPAM Systems, 2016 Стр: 134/287



Прослеживаемость. Из содержащейся в качественном тест-кейсе информации должно быть понятно, какую часть приложения, какие функции и какие требования он проверяет. Частично это свойство достигается через заполнение соответствующих полей тест-кейса⁽¹¹⁷⁾ («Ссылка на требование», «Модуль», «Подмодуль»), но и сама логика тест-кейса играет не последнюю роль, т.к. в случае серьезных нарушений этого свойства можно долго с удивлением смотреть, например, на какое требование ссылается тест-кейс, и пытаться понять, как же они друг с другом связаны.

Пример непрослеживаемого тест-кейса:

Требование	Модуль	Подмодуль	Шаги	Ожидаемые результаты
ПТ-4	Приложение		Совмещение кодировок Приготовления: файл с несколькими допустимыми и недопустимыми кодировками. 1. Передать файл на конвертацию.	1. Допустимые кодировки конвертируются верно, недопустимые остаются без изменений.

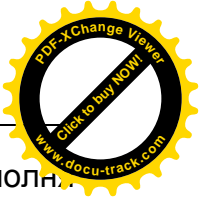
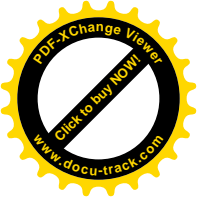
Да, этот тест-кейс плох сам по себе (в качественном тест-кейсе сложно получить ситуацию непрослеживаемости), но в нём есть и особые недостатки, затрудняющие прослеживаемость:

- Ссылка на несуществующее требование (убедитесь сами, требования ПТ-4 нет⁽⁵⁴⁾).
- В поле «Модуль» указано значение «Приложение» (по большому счёту можно было оставлять это поле пустым — это было бы столь же информативно), поле «Подмодуль» не заполнено.
- По заглавию и шагам можно предположить, что этот тест-кейс ближе всего к [ДС-5.1](#) и [ДС-5.3](#), но сформулированный ожидаемый результат не следует явно из этих требований.

Пример прослеживаемого тест-кейса:

Требование	Модуль	Подмодуль	Шаги	Ожидаемые результаты
ДС-2.4 , ДС-3.2	Стартер	Обработчик ошибок	Запуск с некорректными параметрами, несуществующие пути 1. Запустить приложение со всеми тремя параметрами, значения которых указывают на несуществующие в файловой системе пути.	1. В консоли отображаются нижеуказанные сообщения, приложение завершает работу. Сообщения a. SOURCE_DIR: directory not exists or inaccessible. b. DESTINATION_DIR: directory not exists or inaccessible. c. LOG_FILE_NAME: wrong file name or inaccessible path.

Можно подумать, что этот тест-кейс затрагивает [ДС-2](#) и [ДС-3](#) целиком, но в поле «Требование» есть вполне чёткая конкретизация, к тому же указанные модуль, подмодуль и сама логика тест-кейса устраняют оставшиеся сомнения.



Возможность повторного использования. Это свойство редко выполняется для низкоуровневых тест-кейсов⁽¹¹⁵⁾, но при создании высокоуровневых тест-кейсов⁽¹¹⁴⁾ можно добиться таких формулировок, при которых тест-кейс практически без изменений можно будет использовать для тестирования аналогичной функциональности в других проектах или других областях приложения.

Примером тест-кейса, который тяжело использовать повторно, может являться практически любой тест-кейс с высокой специфичностью.

Не самым идеальным, но очень наглядным примером тест-кейса, который может быть легко использован в разных проектах, может служить следующий тест-кейс:



Шаги	Ожидаемые результаты
Запуск, все параметры некорректны 1. Запустить приложение, указав в качестве всех параметров заведомо некорректные значения.	1. Приложение запускается, после чего выводит сообщение с описанием сути проблемы с каждым из параметров и завершает работу.

Соответствие принятым шаблонам оформления и традициям. С шаблонами оформления, как правило, проблем не возникает: они строго определены имеющимся образцом или вообще экранной формой инструментального средства управления тест-кейсами. Что же касается традиций, то они отличаются даже в разных командах в рамках одной компании, и тут невозможно дать иного совета, кроме как «почитайте уже готовые тест-кейсы перед тем как писать свои».

В данном случае обойдёмся без отдельных примеров, т.к. выше и без того приведено много правильно оформленных тест-кейсов, а что касается нарушений этого свойства, то они прямо или косвенно описаны в главе «Типичные ошибки при разработке чек-листов, тест-кейсов и наборов тест-кейсов»⁽¹⁵¹⁾.

2.4.6. Наборы тест-кейсов

Терминология и общие сведения

	Набор тест-кейсов (test case suite ²⁹⁷ , test suite, test set) — совокупность тест-кейсов, выбранных с некоторой общей целью или по некоторому общему признаку. Иногда в такой совокупности результаты завершения одного тест-кейса становятся входным состоянием приложения для следующего тест-кейса.
	Внимание! Из-за особенностей перевода очень часто вместо «набор текст-кейсов» говорят «тестовый сценарий». Формально это можно считать ошибкой, но это явление приобрело настолько широкое распространение, что стало вариантом нормы.

Как мы только что убедились на примере множества отдельных тест-кейсов, крайне неудобно (более того, это ошибка!) каждый раз писать в каждом тест-кейсе одни и те же приготовления и повторять одни и те же начальные шаги.

Намного удобнее объединить несколько тест-кейсов в набор или последовательность. И здесь мы приходим к классификации наборов тест-кейсов.

В общем случае наборы тест-кейсов можно разделить на свободные (порядок выполнения тест-кейсов не важен) и последовательные (порядок выполнения тест-кейсов важен).


Преимущества свободных наборов:

- Тест-кейсы можно выполнять в любом удобном порядке, а также создавать «наборы внутри наборов».
- Если какой-то тест-кейс завершился ошибкой, это не повлияет на возможность выполнения других тест-кейсов.

Преимущества последовательных наборов:

- Каждый следующий в наборе тест-кейс в качестве входного состояния приложения получает результат работы предыдущего тест-кейса, что позволяет сильно сократить количество шагов в отдельных тест-кейсах.
- Длинные последовательности действий куда лучше имитируют работу реальных пользователей, чем отдельные «точечные» воздействия на приложение.

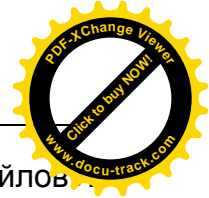
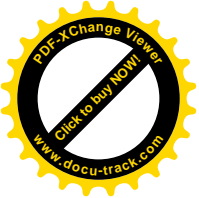
Пользовательские сценарии (сценарии использования)

	В данном случае речь НЕ идёт о use cases (вариантах использования), представляющих собой форму требований ⁽³⁵⁾ . Пользовательские сценарии как техника тестирования куда менее формализованы, хотя и могут строиться на основе вариантов использования.
---	--

К отдельному подвиду последовательных наборов тест-кейсов (или даже неоформленных идей тест-кейсов, таких, как пункты чек-листа) можно отнести пользовательские сценарии²⁹⁸ (или сценарии использования), представляющие собой цепочки действий, выполняемых пользователем в определённой ситуации для достижения определённой цели.

²⁹⁷ **Test case suite (test suite, test set).** A set of several test cases for a component or system under test, where the post condition of one test is often used as the precondition for the next one. [ISTQB Glossary]

²⁹⁸ A scenario is a hypothetical story, used to help a person think through a complex problem or system. [Cem Kaner, «An Introduction to Scenario Testing», <http://www.kaner.com/pdfs/ScenarioIntroVer4.pdf>]



Поясним это сначала на примере, не относящемся к «Конвертеру файлов». Допустим, пользователь хочет распечатать табличку на дверь кабинета с текстом «Идёт работа, не стучать!» Для этого ему нужно:

- 1) Запустить текстовый редактор.
- 2) Создать новый документ (*если редактор не делает это самостоятельно*).
- 3) Набрать в документе текст.
- 4) Отформатировать текст должным образом.
- 5) Отправить документ на печать.
- 6) Сохранить документ (*спорно, но допустим*).
- 7) Закрыть текстовый редактор.

Вот мы и получили пользовательский сценарий, пункты которого могут стать основой для шагов тест-кейса или целого набора отдельных тест-кейсов.

Сценарии могут быть достаточно длинными и сложными, могут содержать внутри себя циклы и условные ветвления, но при всём этом они обладают рядом весьма интересных преимуществ:

- Сценарии показывают реальные и понятные примеры использования продукта (в отличие от обширных чек-листов, где смысл отдельных пунктов может теряться).
- Сценарии понятны конечным пользователям и хорошо подходят для обсуждения и совместного улучшения.
- Сценарии и их части легче оценивать с точки зрения важности, чем отдельные пункты (особенно низкоуровневых) требований.
- Сценарии отлично показывают недоработки в требованиях (если становится непонятно, что делать в том или ином пункте сценария, — с требованиями явно что-то не то).
- В предельном случае (нехватка времени и прочие форс-мажоры) сценарии можно даже не прописывать подробно, а просто именовать — и само наименование уже подскажет опытному специалисту, что делать.

Последний пункт проиллюстрируем на примере. Классифицируем потенциальных пользователей нашего приложения (напомним, что в нашем случае «пользователь» — это администратор, настраивающий работу приложения) по степени квалификации и склонности к экспериментам, а затем дадим каждому «виду пользователя» запоминающееся имя.

Таблица 2.4.а — Классификация пользователей

	Низкая квалификация	Высокая квалификация
Не склонен к экспериментам	«Осторожный»	«Консервативный»
Склонен к экспериментам	«Отчаянный»	«Изощённый»

Согласитесь, уже на этой стадии не составляет труда представить себе отличия в логике работы с приложением, например, «консервативного» и «отчаянного» пользователей.

Но мы пойдём дальше и озаглавим для них сами сценарии, например, в ситуациях, когда такой пользователь позитивно и негативно относится к идее внедрения нашего приложения:

Таблица 2.4.b — Сценарии поведения на основе классификации пользователей

	«Осторожный»	«Консервативный»	«Отчаянный»	«Изощённый»
Позитивно	«А можно вот так?»	«Начнём с инструкции!»	«Гляньте, что я придумал!»	«Я всё оптимизирую!»
Негативно	«Я ничего не понимаю.»	«У вас вот здесь несоответствие...»	«Всё равно поломаю!»	«А я говорил!»

Проявив даже самую малость воображения, можно представить, что и как будет происходить в каждой из получившихся восьми ситуаций. Причём на создание пары таких таблиц уходит всего несколько минут, а эффект от их использования на порядки превосходит бездумное «кликание по кнопкам в надежде найти баг».



Куда более полное и техничное объяснение того, что такое сценарное тестирование, как его применять и выполнять должным образом, можно прочесть в статье Сэма Канера «An Introduction to Scenario Testing»²⁹⁹.

Подробная классификация наборов тест-кейсов



Представленный здесь материал сложно отнести к категории «для начинающих» (и вы можете сразу перейти к пункту «Принципы построения наборов тест-кейсов»^[141]). Но если вам любопытно, взгляните на подробную классификацию, которая представлена ниже.

Подробная классификация наборов тест-кейсов может быть выражена следующей таблицей.

Таблица 2.4.c — Подробная классификация наборов тест-кейсов

		По изолированности тест-кейсов друг от друга	
		Изолированные	Обобщённые
По образованию тест-кейсами строгой последовательности	Свободные	Изолированные свободные	Обобщённые свободные
	Последовательные	Изолированные последовательные	Обобщённые последовательные

- Набор изолированных свободных тест-кейсов (рисунок 2.4.f): действия из раздела «приготовления» нужно повторить перед каждым тест-кейсом, а сами тест-кейсы можно выполнять в любом порядке.
- Набор обобщённых свободных тест-кейсов (рисунок 2.4.g): действия из раздела «приготовления» нужно выполнить один раз (а потом просто выполнять тест-кейсы), а сами тест-кейсы можно выполнять в любом порядке.
- Набор изолированных последовательных тест-кейсов (рисунок 2.4.h): действия из раздела «приготовления» нужно повторить перед каждым тест-кейсом, а сами тест-кейсы нужно выполнять в строго определённом порядке.
- Набор обобщённых последовательных тест-кейсов (рисунок 2.4.i): действия из раздела «приготовления» нужно выполнить один раз (а потом просто выполнять тест-кейсы), а сами тест-кейсы нужно выполнять в строго определённом порядке.

²⁹⁹ «An Introduction to Scenario Testing», Cem Kaner [<http://www.kaner.com/pdfs/ScenarioIntroVer4.pdf>]

Главное преимущество изолированности: каждый тест-кейс выполняется в «чистой среде», на него не влияют результаты работы предыдущих тест-кейсов.

Главное преимущество обобщённости: приготовления не нужно повторять (экономия времени).

Главное преимущество последовательности: ощутимое сокращение шагов в каждом тест-кейсе, т.к. результат выполнения предыдущего тест-кейса является начальной ситуацией для следующего.

Главное преимущество свободы: возможность выполнять тест-кейсы в любом порядке, а также то, что при провале некоего тест-кейса (приложение не пришло в ожидаемое состояние) остальные тест-кейсы по-прежнему можно выполнять.

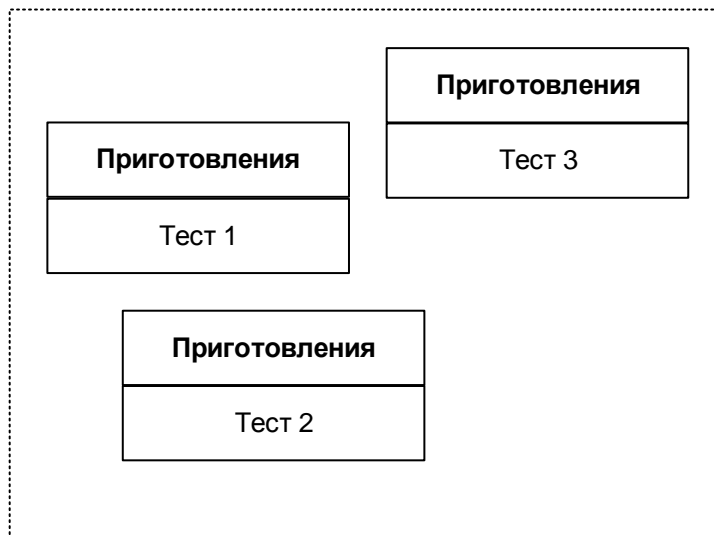


Рисунок 2.4.f — Набор изолированных свободных тест-кейсов

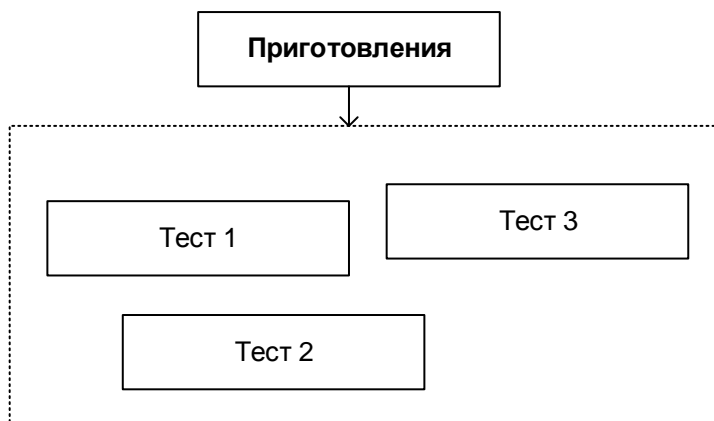


Рисунок 2.4.g — Набор обобщённых свободных тест-кейсов

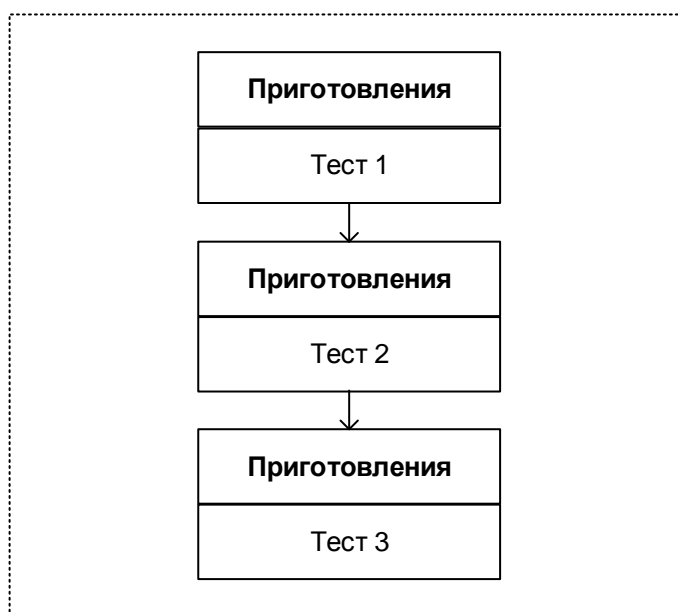


Рисунок 2.4.h — Набор изолированных последовательных тест-кейсов

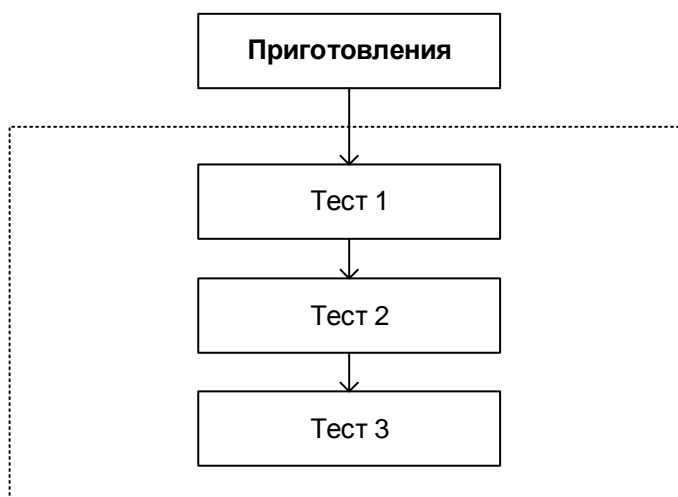


Рисунок 2.4.i — Набор обобщённых последовательных тест-кейсов

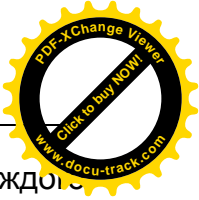
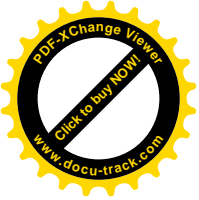
Принципы построения наборов тест-кейсов

Теперь — о самом главном: как формировать наборы тест-кейсов. Правильный ответ звучит очень кратко: логично. И это не шутка. Единственная задача наборов — повысить эффективность тестирования за счёт ускорения и упрощения выполнения тест-кейсов, увеличения глубины исследования некоей области приложения или функциональности, следования типичным пользовательским сценариям⁽¹³⁷⁾ или удобной последовательности выполнения тест-кейсов и т.д.

Набор тест-кейсов всегда создаётся с какой-то целью, на основе какой-то логики, и по этим же принципам в набор включаются тесты, обладающие подходящими свойствами.

Если же говорить о наиболее типичных подходах к составлению наборов тест-кейсов, то можно обозначить следующее:

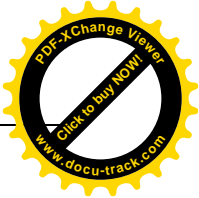
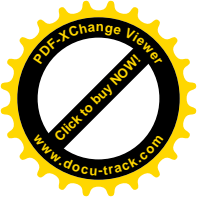
- На основе чек-листов. Посмотрите внимательно на примеры чек-листов⁽¹¹⁰⁾, которые мы разработали в соответствующем разделе⁽¹⁰⁹⁾: каждый пункт чек-листа может превратиться в несколько тест-кейсов — и вот мы получаем готовый набор.



- На основе разбиения приложения на модули и подмодули⁽¹¹⁸⁾. Для каждого модуля (или его отдельных подмодулей) можно составить свой набор тест-кейсов.
- По принципу проверки самых важных, менее важных и всех остальных функций приложения (именно по этому принципу мы составляли примеры чек-листов⁽¹¹⁰⁾).
- По принципу группировки тест-кейсов для проверки некоего уровня требований или типа требований⁽³⁵⁾, группы требований или отдельного требования.
- По принципу частоты обнаружения тест-кейсами дефектов в приложении (например, мы видим, что некоторые тест-кейсы раз за разом завершаются неудачей, значит, мы можем объединить их в набор, условно названный «проблемные места в приложении»).
- По архитектурному принципу (см. «многоуровневая архитектура»¹⁴³ самостоятельно): наборы для проверки пользовательского интерфейса и всего уровня представления, для проверки уровня бизнес-логики, для проверки уровня данных.
- По области внутренней работы приложения, например: «тест-кейсы, затрагивающие работу с базой данных», «тест-кейсы, затрагивающие работу с файловой системой», «тест-кейсы, затрагивающие работу с сетью», и т.д.
- По видам тестирования (см. главу «Подробная классификация тестирования»⁽⁶³⁾).

Не нужно заучивать этот список. Это всего лишь примеры — грубо говоря, «первое, что приходит в голову». Важен принцип: если вы видите, что выполнение некоторых тест-кейсов в виде набора принесёт вам пользу, создавайте такой набор.

Примечание: без хороших инструментальных средств управления тест-кейсами работать с наборами тест-кейсов крайне тяжело, т.к. приходится самостоятельно следить за приготовлениями, «недостающими шагами», изолированностью или обобщённостью, свободностью или последовательностью и т.д.



2.4.7. Логика создания эффективных проверок

Теперь, когда мы рассмотрели принципы построения чек-листов⁽¹⁰⁹⁾ и оформления тест-кейсов⁽¹¹⁷⁾, свойства качественных тест-кейсов⁽¹²⁸⁾, а также принципы объединения тест-кейсов в наборы⁽¹⁴¹⁾, настало время перейти к самой сложной, «философской» части, в которой мы будем рассуждать уже не о том, что и как писать, а о том, как думать.

Ранее уже было сказано: если у тест-кейса не указаны входные данные, условия выполнения и ожидаемые результаты, и/или не ясна цель тест-кейса — это плохой тест-кейс. И здесь во главе угла стоит **цель**. Если мы чётко понимаем, что и зачем мы делаем, мы или быстро находим всю остальную недостающую информацию, или столь же быстро формулируем правильные вопросы и адресуем их правильным людям.

Вся суть работы тестировщика в конечном итоге направлена на повышение качества (процессов, продуктов и т.д.) Но что такое качество? Да, существует сухое официальное определение³⁰⁰, но даже там сказано про «user/customer needs and expectations» (потребности и ожидания пользователя/заказчика).

И здесь проявляется главная мысль: **качество — это некая ценность для конечного пользователя** (заказчика). Человек в любом случае платит за использование продукта — деньгами, своим временем, какими-то усилиями (даже если вы не получаете эту «оплату», человек вполне обоснованно считает, что что-то уже на вас потратил, и он прав). Но получает ли он то, на что рассчитывал (предположим, что его ожидания — здравы и реалистичны)?

Если мы подходим к тестированию формально, мы рискуем получить продукт, который по документам (метрикам и т.д.) выглядит идеально, но в реальности никому не нужен.

Поскольку практически любой современный программный продукт представляет собой непростую систему, среди широкого множества её свойств и функций объективно есть самые важные, менее важные и совсем незначительные по важности для пользователей.

Если усилия тестировщиков будут сконцентрированы на первой и второй категории (самом важном и чуть менее важном), наши шансы создать приложение, удовлетворяющее заказчика, резко увеличиваются.

Есть простая логика:

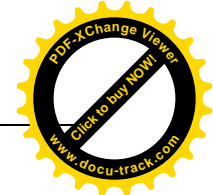
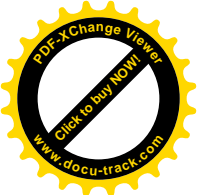
- Тесты ищут ошибки.
- Но все ошибки найти невозможно.
- Значит, наша задача — найти максимум **ВАЖНЫХ** ошибок за имеющееся время.

Под важными ошибками здесь мы понимаем такие, которые приводят к нарушению важных для пользователя функций или свойств продукта. Функции и свойства разделены не случайно — безопасность, производительность, удобство и т.д. не относятся к функциям, но играют ничуть не менее важную роль в формировании удовлетворённости заказчика и конечных пользователей.

Ситуация усугубляется следующими фактами:

- в силу множества экономических и технических причин мы не можем выполнить «все тесты, что придут нам в голову» (да ещё и многократно) — приходится тщательно выбирать, что и как мы будем тестировать, принимая во внимание только что упомянутую мысль: качество — это некая ценность для

³⁰⁰ **Quality.** The degree to which a component, system or process meets specified requirements and/or user/customer needs and expectations. [ISTQB Glossary]



конечного пользователя (заказчика);

- никогда в реальной жизни (как бы мы ни старались) мы не получим «совершенного и идеального набора требований» — там всегда будет некоторое количество недоработок, и это тоже надо принимать во внимание.

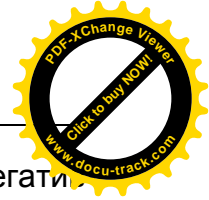
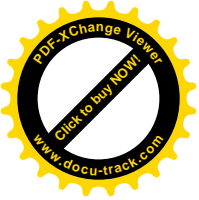
Однако существует достаточно простой алгоритм, позволяющий нам создавать эффективные проверки даже в таких условиях. Приступая к продумыванию чек-листа, тест-кейса или набора тест-кейсов, задайте себе следующие вопросы и получите чёткие ответы:

- Что перед вами? Если вы не понимаете, что вам предстоит тестировать, вы не уйдёте дальше бездумных формальных проверок.
- Кому и зачем оно нужно (и насколько это важно)? Ответ на этот вопрос позволит вам быстро придумать несколько характерных сценариев использования^[137] того, что вы собираетесь тестировать.
- Как оно обычно используется? Это уже детализация сценариев и источник идей для позитивного тестирования^[76] (их удобно оформить в виде чек-листа).
- Как оно может сломаться, т.е. начать работать неверно? Это также детализация сценариев использования, но уже в контексте негативного тестирования^[76] (их тоже удобно оформить в виде чек-листа).

К этому алгоритму можно добавить ещё небольшой перечень универсальных рекомендаций, которые позволят вам проводить тестирование лучше:

- Начинайте как можно раньше — уже с момента появления первых требований можно заниматься их тестированием и улучшением, можно писать чек-листы и тест-кейсы, можно уточнять план тестирования, готовить тестовое окружение и т.д.
- Если вам предстоит тестировать что-то большое и сложное, разбивайте его на модули и подмодули, функциональность подвергайте функциональной декомпозиции³⁰¹ — т.е. добейтесь такого уровня детализации, при котором вы можете без труда удержать в голове всю информацию об объекте тестирования.
- Обязательно пишите чек-листы. Если вам кажется, что вы сможете запомнить все идеи и потом легко их воспроизвести, вы ошибаетесь. Исключений не бывает.
- По мере создания чек-листов, тест-кейсов и т.д. прямо в текст вписывайте возникающие вопросы. Когда вопросов накопится достаточно, соберите их отдельно, уточните формулировки и обратитесь к тому, кто может дать ответы.
- Если используемое вами инструментальное средство позволяет использовать косметическое оформление текста — используйте (так текст будет лучше читаться), но старайтесь следовать общепринятым традициям и не раскрашивать каждое второе слово в свой цвет, шрифт, размер и т.д.
- Используйте технику беглого просмотра^[45] для получения отзыва от коллег и улучшения созданного вами документа.
- Планируйте время на улучшение тест-кейсов (исправление ошибок, доработку по факту изменения требований и т.д.).
- Начинайте проработку (и выполнение) тест-кейсов с простых позитивных проверок наиболее важной функциональности. Затем постепенно повы-

³⁰¹ «Functional decomposition», Wikipedia [http://en.wikipedia.org/wiki/Functional_decomposition]



шайте сложность проверок, помня не только о позитивных⁽⁷⁶⁾, но и о негативных⁽⁷⁶⁾ проверках.

- Помните, что в основе тестирования лежит цель. Если вы не можете быстро и просто сформулировать цель созданного вами тест-кейса, вы создали плохой тест-кейс.
- Избегайте избыточных, дублирующих друг друга тест-кейсов. Минимизировать их количество вам помогут техники классов эквивалентности⁽⁸⁸⁾, граничных условий⁽⁸⁸⁾, доменного тестирования⁽⁸⁹⁾.
- Если показательность⁽¹³²⁾ тест-кейса можно увеличить, при этом не сильно изменив его сложность и не отклонившись от исходной цели, сделайте это.
- Помните, что очень многие тест-кейсы требуют отдельной подготовки, которую нужно описать в соответствующем поле тест-кейса.
- Несколько позитивных тест-кейсов⁽⁷⁶⁾ можно безбоязненно объединять, но объединение негативных тест-кейсов⁽⁷⁶⁾ почти всегда запрещено.
- Подумайте, как можно оптимизировать созданный вами тест-кейс (набор тест-кейсов и т.д.) так, чтобы снизить трудозатраты на его выполнение.
- Перед тем как отправлять финальную версию созданного вами документа, ещё раз перечитайте написанное (в доброй половине случаев найдёте опечатку или иную недоработку).



Задание 2.4.е: дополните этот список идеями, которые вы почерпнули из других книг, статей и т.д.

Пример реализации логики создания эффективных проверок

Ранее мы составили подробный чек-лист⁽¹⁰⁹⁾ для тестирования нашего «Конвертера файлов»⁽⁵⁴⁾. Давайте посмотрим на него критически и подумаем: что можно сократить, чем мы в итоге пожертвуем и какой выигрыш получим.

Прежде чем мы начнём оптимизировать чек-лист, важно отметить, что решение о том, что важно и что неважно, стоит принимать на основе ранжирования требований по важности, а также согласовывать с заказчиком.

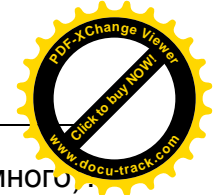
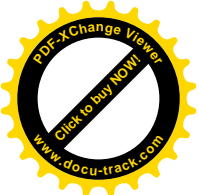
Что для пользователя САМОЕ важное? Ради чего создавалось приложение? Чтобы конвертировать файлы. Принимая во внимание тот факт, что настройку приложения будет выполнять квалифицированный технический специалист, мы можем даже «отодвинуть на второй план» реакцию приложения на ошибки стадии запуска и завершения.

И на первое место выходит:

- Обработка файлов, разные форматы, кодировки и размеры:

Таблица 2.4.d — Форматы, кодировки и размеры файлов

		Форматы входных файлов		
		TXT	HTML	MD
Кодировки входных файлов	WIN1251	100 КБ	50 МБ	10 МБ
	CP866	10 МБ	100 КБ	50 МБ
	KOI8R	50 МБ	10 МБ	100 КБ
	Любая	0 байт		
	Любая	50 МБ + 1 Б	50 МБ + 1 Б	50 МБ + 1 Б
	-	Любой недопустимый формат		
	Любая	Повреждения в допустимом формате		



Можем ли мы как-то ускорить выполнение этих проверок (ведь их много)? Можем. И у нас даже есть два взаимодополняющих инструмента:

- дальнейшая классификация по важности;
- автоматизация тестирования.

Сначала поделим таблицу на две — чуть более важное и чуть менее важное. В «чуть более важное» попадает:

Таблица 2.4.e — Форматы, кодировки и размеры файлов

		Форматы входных файлов		
		ТХТ	HTML	MD
Кодировки входных файлов	WIN1251	100 КБ	50 МБ	10 МБ
	CP866	10 МБ	100 КБ	50 МБ
	KOI8R	50 МБ	10 МБ	100 КБ

Подготовим 18 файлов — 9 исходных + 9 сконвертированных (в любом текстовом редакторе с функцией конвертации кодировок), чтобы в дальнейшем сравнивать работу нашего приложения с эталоном.

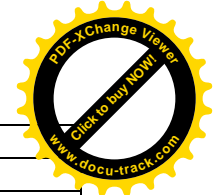
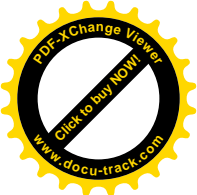
Для «чуть менее важного» осталось:

- Файл размером 0 байт (объективно для него не важна такая характеристика, как «кодировка»). *Подготовим один файл размером 0 байт.*
- Файл размером 50 МБ + 1 Б (для него тоже не важна кодировка). *Подготовим один файл размером 52'428'801 байт.*
- Любой недопустимый формат:
 - По расширению (файл с расширением, отличным от .txt, .html, .md). *Берём любой произвольный файл, например картинку (размер от 1 до 50 МБ, расширение .jpg).*
 - По внутреннему содержанию (например, .jpg переименовали в .txt). *Копии файла из предыдущего пункта даём расширение .txt.*
- Повреждения в допустимом формате. *Вычёркиваем. Вообще. Даже крайне сложные дорогие редакторы далеко не всегда способны восстановить повреждённые файлы своих форматов, наше же приложение — это просто миниатюрная утилита конвертации кодировок, и не стоит ждать от неё возможностей профессионального инструментального средства восстановления данных.*

Что мы получили в итоге? Нам нужно подготовить следующие 22 файла (поскольку у файлов всё равно есть имена, усилим этот набор тестовых данных, представив в именах файлов символы латиницы, кириллицы и спецсимволы).

Таблица 2.4.f — Итоговый набор файлов для тестирования приложения

№	Имя	Кодировка	Размер
1	«Мелкий» файл в WIN1251.txt	WIN1251	100 КБ
2	«Средний» файл в CP866.txt	CP866	10 МБ
3	«Крупный» файл в KOI8R.txt	KOI8R	50 МБ
4	«Крупный» файл в win-1251.html	WIN1251	50 МБ
5	«Мелкий» файл в cp-866.html	CP866	100 КБ
6	«Средний» файл в koi8-r.html	KOI8R	10 МБ
7	«Средний» файл в WIN_1251.md	WIN1251	10 МБ
8	«Крупный» файл в CP_866.md	CP866	50 МБ
9	«Мелкий» файл в KOI8_R.md	KOI8R	100 КБ
10	«Мелкий» эталон WIN1251.txt	UTF8	100 КБ
11	«Средний» эталон CP866.txt	UTF8	10 МБ



12	«Крупный» эталон KOI8R.txt	UTF8	50 МБ
13	«Крупный» эталон в win-1251.html	UTF8	50 МБ
14	«Мелкий» эталон в cp-866.html	UTF8	100 КБ
15	«Средний» эталон в koi8-r.html	UTF8	10 МБ
16	«Средний» эталон в WIN_1251.md	UTF8	10 МБ
17	«Крупный» эталон в CP_866.md	UTF8	50 МБ
18	«Мелкий» эталон в KOI8_R.md	UTF8	100 КБ
19	Пустой файл.md	-	0 Б
20	Слишком большой файл.txt	-	52'428'801 Б
21	Картинка.jpg	-	~ 1 МБ
22	Картинка в виде TXT.txt	-	~ 1 МБ

И только что мы упомянули автоматизацию как способ ускорения выполнения тест-кейсов. В данном случае мы можем обойтись самыми тривиальными командными файлами. В приложении «Командные файлы для Windows и Linux, автоматизирующие выполнение дымового тестирования»⁽²⁷⁰⁾ приведены скрипты, полностью автоматизирующие выполнение всего уровня дымового тестирования⁽⁷³⁾ над представленным выше набором из 22 файлов.



Задание 2.4.f: доработайте представленные в приложении⁽²⁷⁰⁾ командные файлы так, чтобы их выполнение заодно проверяло работу тестируемого приложения с пробелами, кириллическими символами и спецсимволами в путях к входному каталогу, выходному каталогу и файлу журнала. Оптимизируйте получившиеся командные файлы так, чтобы избежать многократного дублирования кода.

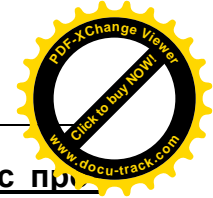
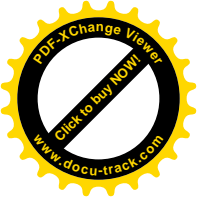
Если снова вернуться к чек-листу, то оказывается, что мы уже подготовили проверки для всего уровня дымового тестирования⁽⁷³⁾ и части уровня тестирования критического пути⁽⁷⁴⁾.

Продолжим оптимизацию. Большинство проверок не представляет особой сложности, и мы разберёмся с ними по ходу дела, но есть в чек-листе пункт, вызывающий особую тревогу: производительность.

Тестирование и оптимизация производительности⁽⁸⁵⁾ — это отдельный вид тестирования со своими достаточно непростыми правилами и подходами, к тому же разделяющийся на несколько подвидов. Нужно ли оно нам в нашем приложении? Заказчик в АК-1.1 определил минимальную производительность приложения как способность обрабатывать входные данные со скоростью не менее 5 МБ/сек. Грубые эксперименты на указанном в АК-1.1 оборудовании показывают, что даже куда более сложные операции (например, архивирование видеофайла с максимальной степенью сжатия) выполняются быстрее (пусть и ненамного). Вывод? Вычёркиваем. Вероятность встретить здесь проблему ничтожно мала, а соответствующее тестирование требует ощутимых затрат сил и времени, а также наличия соответствующих специалистов.

Вернёмся к чек-листу:

- Конфигурирование и запуск:
 - С верными параметрами
 - Значения `SOURCE_DIR`, `DESTINATION_DIR`, `LOG_FILE_NAME` указаны и содержат пробелы и кириллические символы (повторить для форматов путей в Windows и *nix файловых системах, обратить внимание на имена логических дисков и разделители имён каталогов ("/" и "\")): **(Уже учтено при автоматизации проверки работы приложения с 22 файлами.)**

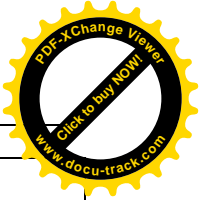
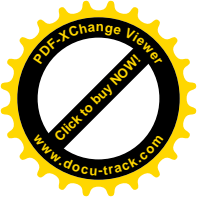


- ~~Значение LOG_FILE_NAME не указано.~~ **(Объединить с проверкой ведения самого файла журнала.)**
- Без параметров.
- С недостаточным количеством параметров.
- С неверными параметрами:
 - Недопустимый путь SOURCE_DIR.
 - Недопустимый путь DESTINATION_DIR.
 - Недопустимое имя LOG_FILE_NAME.
 - DESTINATION_DIR находится внутри SOURCE_DIR.
 - Значения DESTINATION_DIR и SOURCE_DIR совпадают.
- Обработка файлов:
 - ~~Разные форматы, кодировки и размеры.~~ **(Уже сделано.)**
 - Недоступные входные файлы:
 - Нет прав доступа.
 - Файл открыт и заблокирован.
 - Файл с атрибутом «только для чтения».
- ~~Остановка:~~
 - ~~Закрытием окна консоли.~~ **(Вычёркиваем. Не настолько важная проверка, а если и будут проблемы — технология PHP не позволит их решить.)**
- Журнал работы приложения:
 - Автоматическое создание (при отсутствии журнала), имя журнала указано явно.
 - Продолжение (дополнение журнала) при повторных запусках, имя журнала не указано.
- ~~Производительность:~~
 - ~~Элементарный тест с грубой оценкой.~~ **(Ранее решили, что наше приложение явно уложится в весьма демократичные требования заказчика.)**

Перепишем компактно то, что у нас осталось от уровня тестирования критического пути^[74]. Внимание! Это — НЕ тест-кейс! Это всего лишь ещё одна форма записи чек-листа, более удобная на данном этапе.

Таблица 2.4.g — Чек-лист для уровня критического пути

Суть проверки	Ожидаемая реакция
Запуск без параметров.	Отображение инструкции к использованию.
Запуск с недостаточным количеством параметров.	Отображение инструкции к использованию и указание имён недостающих параметров.
Запуск с неверными значениями параметров: <ul style="list-style-type: none">○ Недопустимый путь SOURCE_DIR.○ Недопустимый путь DESTINATION_DIR.○ Недопустимое имя LOG_FILE_NAME.○ DESTINATION_DIR находится внутри SOURCE_DIR.○ Значения DESTINATION_DIR и SOURCE_DIR совпадают.	Отображение инструкции к использованию и указание имени неверного параметра, значения неверного параметра и пояснения сути проблемы.
Недоступные входные файлы: <ul style="list-style-type: none">○ Нет прав доступа.○ Файл открыт и заблокирован.○ Файл с атрибутом «только для чтения».	Отображение сообщения в консоль и файл журнала, дальнейшее игнорирование недоступных файлов.
Журнал работы приложения: <ul style="list-style-type: none">○ Автоматическое создание (при отсутствии журнала), имя журнала указано явно.	Создание или продолжение ведения файла журнала по указанному или вычисленному пути.



- Продолжение (дополнение журнала) при повторных запусках, имя журнала не указано.

Наконец, у нас остался уровень расширенного тестирования⁽⁷⁵⁾. И сейчас мы сделаем то, чего по всем классическим книгам учат не делать, — мы откажемся от всего этого набора проверок целиком.

- ~~Конфигурирование и запуск:~~
 - ~~Значения SOURCE_DIR, DESTINATION_DIR, LOG_FILE_NAME:~~
 - * ~~В разных стилях (Windows-пути + *nix-пути) — одно в одном стиле, другое — в другом.~~
 - * ~~С использованием UNC-имён.~~
 - * ~~LOG_FILE_NAME внутри SOURCE_DIR.~~
 - * ~~LOG_FILE_NAME внутри DESTINATION_DIR.~~
 - ~~Размер LOG_FILE_NAME на момент запуска:~~
 - * ~~2–4 ГБ.~~
 - * ~~4+ ГБ.~~
 - ~~Запуск двух и более копий приложения с:~~
 - * ~~Одинаковыми параметрами SOURCE_DIR, DESTINATION_DIR, LOG_FILE_NAME.~~
 - * ~~Одинаковыми SOURCE_DIR и LOG_FILE_NAME, но разными DESTINATION_DIR.~~
 - * ~~Одинаковыми DESTINATION_DIR и LOG_FILE_NAME, но разными SOURCE_DIR.~~
- ~~Обработка файлов:~~
 - ~~Файл верного формата, в котором текст представлен в двух и более поддерживаемых кодировках одновременно.~~
 - ~~Размер входного файла:~~
 - * ~~2–4 ГБ.~~
 - * ~~4+ ГБ.~~

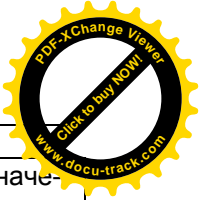
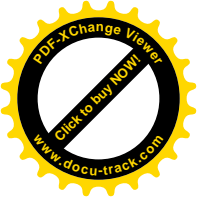
Да, мы сейчас действительно повысили риск пропустить какой-то дефект. Но — дефект, вероятность возникновения которого мала в силу малой вероятности возникновения описанных в этих проверках ситуаций. При этом по самым скромным прикидкам мы на треть сократили общее количество проверок, которые нам нужно будет выполнять, а значит — высвободили силы и время для более тщательной проработки типичных каждодневных сценариев использования⁽¹³⁷⁾ приложения.

Весь оптимизированный чек-лист (он же — и черновик для плана выполнения проверок) теперь выглядит так:

- 1) Подготовить файлы (см. таблицу 2.4.f).
- 2) Для «дымового теста» использовать командные файлы (см. приложение «Командные файлы для Windows и Linux, автоматизирующие выполнение дымового тестирования»⁽²⁷⁰⁾).
- 3) Для основных проверок использовать файлы из пункта 1 и следующие идеи (таблица 2.4.h).

Таблица 2.4.h — Основные проверки для приложения «Конвертер файлов»

Суть проверки	Ожидаемая реакция
Запуск без параметров.	Отображение инструкции к использованию.
Запуск с недостаточным количеством параметров.	Отображение инструкции к использованию и указание имён недостающих параметров.
Запуск с неверными значениями параметров:	Отображение инструкции к использованию и



Логика создания эффективных проверок

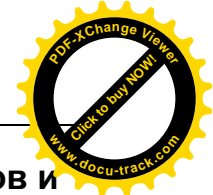
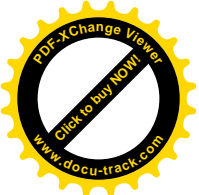
<ul style="list-style-type: none">Недопустимый путь SOURCE_DIR.Недопустимый путь DESTINATION_DIR.Недопустимое имя LOG_FILE_NAME.DESTINATION_DIR находится внутри SOURCE_DIR.Значения DESTINATION_DIR и SOURCE_DIR совпадают.	указание имени неверного параметра, значения неверного параметра и пояснения сути проблемы.
Недоступные входные файлы: <ul style="list-style-type: none">Нет прав доступа.Файл открыт и заблокирован.Файл с атрибутом «только для чтения».	Отображение сообщения в консоль и файл журнала, дальнейшее игнорирование недоступных файлов.
Журнал работы приложения: <ul style="list-style-type: none">Автоматическое создание (при отсутствии журнала), имя журнала указано явно.Продолжение (дополнение журнала) при повторных запусках, имя журнала не указано.	Создание или продолжение ведения файла журнала по указанному или вычисленному пути.

- 4) В случае наличия времени использовать первоначальную редакцию чек-листа для уровня расширенного тестирования⁽⁷⁵⁾ как основу для выполнения исследовательского тестирования⁽⁷⁸⁾.

И почти всё. Остаётся только ещё раз подчеркнуть, что представленная логика выбора проверок не претендует на то, чтобы считаться единственно верной, но она явно позволяет сэкономить огромное количество усилий, при этом практически не снизив качество тестирования наиболее востребованной заказчиком функциональности приложения.



Задание 2.4.g: подумайте, какие проверки из таблицы 2.4.h можно автоматизировать с помощью командных файлов. Напишите такие командные файлы.



2.4.8. Типичные ошибки при разработке чек-листов, тест-кейсов и наборов тест-кейсов

Ошибки оформления и формулировок

Отсутствие заглавия тест-кейса. В абсолютном большинстве систем управления тест-кейсами поле для заглавия вынесено отдельно и является обязательным к заполнению — тогда эта проблема отпадает. Если же инструментальное средство позволяет создать тест-кейс без заглавия, возникает риск получить N тест-кейсов, для понимания сути каждого из которых нужно прочесть десятки строк вместо одного предложения. Это гарантированное убийство рабочего времени и снижение производительности команды на порядок.

Если заглавие тест-кейса приходится вписывать в поле с шагами и инструментальное средство допускает форматирование текста, заглавие стоит писать **жирным шрифтом**, чтобы его было легче отделять от основного текста.

Отсутствие нумерации шагов и/или ожидаемых результатов (даже если таковой всего лишь один). Наличие этой ошибки превращает тест-кейс в «поток сознания», в котором нет структурированности, модифицируемости и прочих полезных свойств (да, многие свойства качественных требований⁽⁴⁰⁾ в полной мере применимы и к тест-кейсам) — становится очень легко перепутать, что к чему относится. Даже выполнение такого тест-кейса усложняется, а доработка и вовсе превращается в каторжный труд.

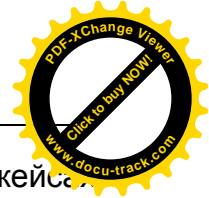
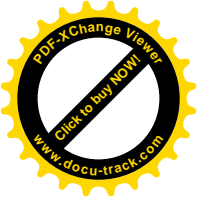
Ссылка на множество требований. Иногда высокоуровневый тест-кейс⁽¹¹⁴⁾ действительно затрагивает несколько требований, но в таком случае рекомендуется писать ссылку на максимум 2–3 самых ключевых (наиболее соответствующих цели тест-кейса), а ещё лучше — указывать общий раздел этих требований (т.е. не ссылаться, например, на требования 5.7.1, 5.7.2, 5.7.3, 5.7.7, 5.7.9, 5.7.12, а просто сослаться на раздел 5.7, включающий в себя все перечисленные пункты). В большинстве инструментальных средств управления тест-кейсами это поле представляет собой выпадающий список, и там эта проблема теряет актуальность.

Использование личной формы глаголов. Если вы пишете требования на русском, то пишете «нажать» вместо «нажмите», «ввести» вместо «введите», «перейти» вместо «перейдите» и т.д. В технической документации вообще не рекомендуется «переходить на личности», а также существует мнение, что личная форма глаголов подсознательно воспринимается как «чужие бессмысленные команды» и приводит к повышенной утомляемости и раздражительности.

Использование прошедшего или будущего времени в ожидаемых результатах. Это не очень серьёзная ошибка, но всё равно «введённое значение отображается в поле» читается лучше, чем «введённое значение отобразилось в поле» или «введённое значение отобразится в поле».

Постоянное использование слов «проверить» (и ему подобных) в чек-листах. В итоге почти каждый пункт чек-листа начинается с «проверить ...», «проверить...», «проверить...». Но ведь весь чек-лист — это и есть список проверок! Зачем писать это слово? Сравните:

Плохо	Хорошо
Проверить запуск приложения.	Запуск приложения.
Проверить открытие корректного файла.	Открытие корректного файла.
Проверить модификацию файла.	Модификация файла.
Проверить сохранение файла.	Сохранение файла.
Проверить закрытие приложения.	Закрытие приложения.



Сюда же относится типичное слово «попытаться» в негативных тест-кейсах («попытаться поделить на ноль», «попытаться открыть несуществующий файл», «попытаться ввести недопустимые символы»): «деление на ноль», «открытие несуществующего файла», «ввод спецсимволов» намного короче и информативнее. А реакцию приложения, если она не очевидна, можно указать в скобках (так будет даже информативнее): «деление на ноль» (сообщение «Division by zero detected»), «открытие несуществующего файла» (приводит к автоматическому созданию файла), «ввод спецсимволов» (символы не вводятся, отображается подсказка).

Описание стандартных элементов интерфейса вместо использования их устоявшихся названий. «Маленький крестик справа сверху окна приложения» — это системная кнопка «Закрыть» (system button «Close»), «быстро-быстро дважды нажать на левую клавишу мыши» — это двойной щелчок (double click), «маленькое окошечко с надписью появляется, когда наводишь мышь» — это всплывающая подсказка (hint).

Пунктуационные, орфографические, синтаксические и им подобные ошибки. Без комментариев.

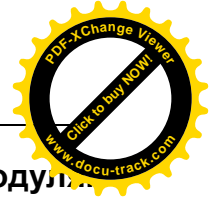
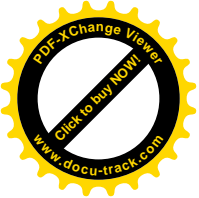
Логические ошибки

Ссылка на другие тест-кейсы или шаги других тест-кейсов. За исключением случаев написания строго оговорённого явно обозначенного набора последовательных тест-кейсов⁽¹³⁹⁾ это запрещено делать. В лучшем случае вам повезёт, и тест-кейс, на который вы ссылались, будет просто удалён — повезёт потому, что это будет сразу заметно. Не повезёт в случае, если тест-кейс, на который вы ссылаетесь, будет изменён — ссылка по-прежнему ведёт в некое существующее место, но описано там уже совершенно не то, что было в момент создания ссылки.

Детализация, не соответствующая уровню функционального тестирования⁽⁷³⁾. Например, не нужно на уровне дымового тестирования⁽⁷³⁾ проверять работоспособность каждой отдельной кнопки или прописывать некий крайне сложный, нетривиальный и редкий сценарий — поведение кнопок и без явного указания будет проверено множеством тест-кейсов, объективно задействующих эти кнопки, а сложному сценарию место на уровне тестирования критического пути⁽⁷⁴⁾ или даже на уровне расширенного тестирования⁽⁷⁵⁾ (в которых, напротив, недостатком можно считать излишнее обобщение без должной детализации).

Расплывчатые двусмысленные описания действий и ожидаемых результатов. Помните, что тест-кейс с высокой вероятностью будете выполнять не вы (автор тест-кейса), а другой сотрудник, и он — не телепат. Попробуйте догадаться по этим примерам, что имел в виду автор:

- «Установить приложение на диск С». (Т.е. в «С:\»? Прямо в корень? Или как?)
- «Нажать на иконку приложения». (Например, если у меня есть iso-файл с иконкой приложения, и я по нему кликну — это оно? Или нет?)
- «Окно приложения запустится». (Куда?)
- «Работает верно». (Ого! А верно — это, простите, как?)
- «ОК». (И? Что «ОК»?)
- «Количество найденных файлов совпадает». (С чем?)
- «Приложение отказывается выполнять команду». (Что значит «отказывается»? Как это выглядит? Что должно происходить?)



Описание действий в качестве наименований модуля/подмодуля. Например, «запуск приложения» — это НЕ модуль или подмодуль. Модуль или подмодуль⁽¹¹⁸⁾ — это всегда некие части приложения, а не его поведение. Сравните: «дыхательная система» — это модуль человека, но «дыхание» — нет.

Описание событий или процессов в качестве шагов или ожидаемых результатов. Например, в качестве шага сказано: «Ввод спецсимволов в поле X». Это было бы сносным заглавием тест-кейса, но не годится в качестве шага, который должен быть сформулирован как «Ввести спецсимволы (перечень) в поле X».

Куда страшнее, если подобное встречается в ожидаемых результатах. Например, там написано: «Отображение скорости чтения в панели X». И что? Оно должно начаться, продолжиться, завершиться, не начинаться, неким образом измениться (например, измениться должна размерность данных), как-то на что-то повлиять? Тест-кейс становится полностью бессмысленным, т.к. такой ожидаемый результат невозможно сравнить с фактическим поведением приложения.

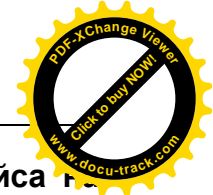
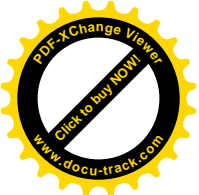
«Выдумывание» особенностей поведения приложения. Да, часто в требованиях отсутствуют самоочевидные (без кавычек, они на самом деле самоочевидные) вещи, но нередко встречаются и некачественные (например, неполные) требования, которые нужно улучшать, а не «телепатически компенсировать».

Например, в требованиях сказано, что «приложение должно отображать диалоговое окно сохранения с указанным по умолчанию каталогом». Если из контекста (соседних требований, иных документов) ничего не удаётся узнать об этом таинственном «каталоге по умолчанию», нужно задать вопрос. Нельзя просто записать в ожидаемых результатах «отображается диалоговое окно сохранения с указанным по умолчанию каталогом» (как мы проверим, что выбран именно указанный по умолчанию каталог, а не какой-то иной?). И уж тем более нельзя в ожидаемых результатах писать «отображается диалоговое окно сохранения с выбранным каталогом “C:/SavedDocuments”» (откуда взялось это «C:/SavedDocuments», — не ясно, т.е. оно явно выдумано из головы и, скорее всего, выдумано неправильно).

Отсутствие описания приготовления к выполнению тест-кейса. Часто для корректного выполнения тест-кейса необходимо каким-то особым образом настроить окружение. Предположим, что мы проверяем приложение, выполняющее резервное копирование файлов. Если тест выглядит примерно так, то выполняющий его сотрудник будет в замешательстве, т.к. ожидаемый результат кажется просто бредом. Откуда взялось «~200»? Что это вообще такое?

Шаги выполнения	Ожидаемые результаты
1. Нажать на панели «Главная» кнопку «Быстрая дедубликация». 2. Выбрать каталог «C:/MyData».	1. Кнопка «Быстрая дедубликация» переходит в утопленное состояние и меняет цвет с серого на зелёный. 2. На панели «Состояние» в поле «Дубликаты» отображается «~200».

И совсем иначе этот тест-кейс воспринимался бы, если бы в приготовлениях было сказано: «Создать каталог “C:/MyData” с произвольным набором подкаталогов (глубина вложенности не менее пяти). В полученном дереве каталогов разместить 1000 файлов, из которых 200 имеют одинаковые имена и размеры, но НЕ внутреннее содержимое».



Полное дублирование (копирование) одного и того же тест-кейса на разных уровнях дымового тестирования, тестирования критического пути, расширенного тестирования. Многие идеи естественным образом развиваются от уровня к уровню^[73], но они должны именно развиваться, а не дублироваться. Сравните:

	Дымовое тестирование	Тестирование критического пути	Расширенное тестирование
Плохо	Запуск приложения.	Запуск приложения.	Запуск приложения.
Хорошо	Запуск приложения.	Запуск приложения из командной строки. Запуск приложения через ярлык на рабочем столе. Запуск приложения через меню «Пуск».	Запуск приложения из командной строки в активном режиме. Запуск приложения из командной строки в фоновом режиме. Запуск приложения через ярлык на рабочем столе от имени администратора. Запуск приложения через меню «Пуск» из списка часто запускаемых приложений.

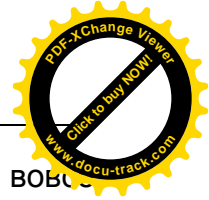
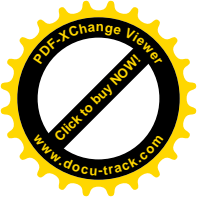
Слишком длинный перечень шагов, не относящихся к сути (цели) тест-кейса. Например, мы хотим проверить корректность одностороннего режима печати из нашего приложения на дуплексном принтере. Сравните:

Плохо	Хорошо
Односторонняя печать 1. Запустить приложение. 2. В меню выбрать «Файл» -> «Открыть». 3. Выбрать любой DOCX-файл, состоящий из нескольких страниц. 4. Нажать кнопку «Открыть». 5. В меню выбрать «Файл» -> «Печать». 6. В списке «Двусторонняя печать» выбрать пункт «Нет». 7. Нажать кнопку «Печать». 8. Закрыть файл. 9. Закрыть приложение.	Односторонняя печать 1. Открыть любой DOCX-файл, содержащий три и более непустых страницы. 2. В диалоговом окне «Настройка печати» в списке «Двусторонняя печать» выбрать «Нет». 3. Произвести печать документа на принтере, поддерживающем двустороннюю печать.

Слева мы видим огромное количество действий, не относящихся непосредственно к тому, что проверяет тест-кейс. Тем более что запуск и закрытие приложения, открытие файла, работа меню и прочее или будут покрыты другими тест-кейсами (со своими соответствующими целями), или на самом деле являются самоочевидными (логично ведь, что нельзя открыть приложением файл, если приложение не запущено) и не нуждаются в описании шагов, которые только создают информационный шум и занимают время на написание и прочтение.

Некорректное наименование элементов интерфейса или их свойств. Иногда из контекста понятно, что автор тест-кейса имел в виду, но иногда это становится реальной проблемой. Например, мы видим тест-кейс с заголовком «Закрытие приложения кнопками "Close" и "Close window"». Уже тут возникает недоумение по поводу того, в чём же различие этих кнопок, да и о чём вообще идёт речь. Ниже (в шагах тест-кейса) автор поясняет: «В рабочей панели внизу экрана нажать "Close window"». Ага! Ясно. Но «Close window» — это НЕ кнопка, это пункт системного контекстного меню приложения в панели задач.

Ещё один отличный пример: «Окно приложения свернётся в окно меньшего



диаметра». Хм. Окно круглое? Или должно стать круглым? А, может, тут и вовсе речь про два разных окна, и одно должно будет оказаться внутри второго? Или, всё же «размер окна уменьшается» (кстати, насколько?), а его геометрическая форма остаётся прямоугольной?

И, наконец, пример, из-за которого вполне можно написать отчёт о дефекте на вполне корректно работающее приложение: «В системном меню выбрать “Фиксация расположения”». Казалось бы, что ж тут плохого? Но потом выясняется, что имелось в виду главное меню приложения, а вовсе не системное меню.

Непонимание принципов работы приложения и вызванная этим некорректность тест-кейсов. Классикой жанра является закрытие приложения: тот факт, что окно приложения «исчезло» (сюрприз: например, оно свернулось в область уведомления панели задач (system tray, taskbar notification area)), или приложение отключило интерфейс пользователя, продолжив функционировать в фоновом режиме, вовсе не является признаком того, что оно завершило работу.

Проверка типичной «системной» функциональности. Если только ваше приложение не написано с использованием каких-то особенных библиотек и технологий и не реализует какое-то нетипичное поведение, нет необходимости проверять системные кнопки, системные меню, сворачивание-разворачивание окна и т.д. Вероятность встретить здесь ошибку стремится к нулю. Если всё же очень хочется, можно вписать эти проверки как уточнения некоторых действий на уровне тестирования критического пути^[74], но создавать для них отдельные тест-кейсы не нужно.

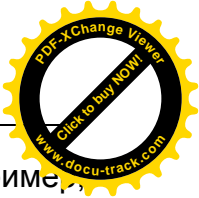
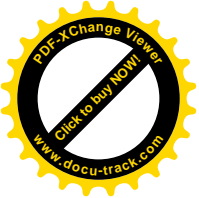
Неверное поведение приложения как ожидаемый результат. Такое не допускается по определению. Не может быть тест-кейса с шагом в стиле «поделить на ноль» с ожидаемым результатом «крах приложения с потерей пользовательских данных». Ожидаемые результаты всегда описывают правильное поведение приложения — даже в самых страшных стрессовых тест-кейсах.

Общая некорректность тест-кейсов. Может быть вызвана множеством причин и выражаться множеством образов, но вот классический пример:

Шаги выполнения	Ожидаемые результаты
... 4. Закрыть приложение нажатием Alt+F4. 5. Выбрать в меню «Текущее состояние».	... 4. Приложение завершает работу. 5. Отображается окно с заголовком «Текущее состояние» и содержимым, соответствующим рисунку 999.99.

Здесь или не указано, что вызов окна «Текущее состояние» происходит где-то в другом приложении, или остаётся загадкой, как вызвать это окно у завершившего работу приложения. Запустить заново? Возможно, но в тест-кейсе этого не сказано.

Неверное разбиение наборов данных на классы эквивалентности. Действительно, иногда классы эквивалентности^[88] могут быть очень неочевидными. Но ошибки встречаются и в довольно простых случаях. Допустим, в требованиях сказано, что размер некоего файла может быть от 10 до 100 КБ (включительно). Разбиение по размеру 0–9 КБ, 10–100 КБ, 101+ КБ **ошибочно**, т.к. килобайт не является неделимой единицей. Такое ошибочное разбиение не учитывает, например, размеры в 9.5 КБ, 100.1 КБ, 100.7 КБ и т.д. Потому здесь стоит применять неравенства: $0 \text{ КБ} \leq \text{размер} < 10 \text{ КБ}$, $10 \text{ КБ} \leq \text{размер} \leq 100 \text{ КБ}$, $100 \text{ КБ} < \text{размер}$. Также можно писать с использованием синтаксиса скобок: $[0, 10) \text{ КБ}$, $[10, 100] \text{ КБ}$, $(100, \infty) \text{ КБ}$, но вариант с неравенствами более привычен большинству людей.



Тест-кейсы, не относящиеся к тестируемому приложению. Например, нам нужно протестировать фотогалерею на сайте. Соответственно, следующие тест-кейсы никак не относятся к фотогалерее (они тестируют браузер, операционную систему пользователя, файловый менеджер и т.д. — но НЕ наше приложение, ни его серверную, ни даже клиентскую часть):

- Файл с сетевого диска.
- Файл со съёмного носителя.
- Файл, заблокированный другим приложением.
- Файл, открытый другим приложением.
- Файл, к которому у пользователя нет прав доступа.
- Вручную указать путь к файлу.
- Файл из глубоко расположенной поддиректории.

Теперь для лучшего закрепления рекомендуется заново прочитать про оформление атрибутов тест-кейсов⁽¹¹⁷⁾, свойства качественных тест-кейсов⁽¹²⁸⁾ и логику построения⁽¹⁴³⁾ качественных тест-кейсов и качественных наборов тест-кейсов.