

## Лекция 15 Чек-листы, тест-кейсы, наборы тест-кейсов

### 2.4.1. Чек-лист

Как легко можно понять из предыдущих глав, тестировщику приходится работать с огромным количеством информации, выбирать из множества вариантов решения задач и изобретать новые. В процессе этой деятельности объективно невозможно удержать в голове все мысли, а потому продумывание и разработку тест-кейсов рекомендуется выполнять с использованием так называемых «чек-листов».



**Чек-лист** (checklist<sup>281</sup>) — набор идей [тест-кейсов]. Последнее слово не зря взято в скобки<sup>282</sup>, т.к. в общем случае чек-лист — это просто набор идей: идей по тестированию, идей по разработке, идей по планированию и управлению — **любых** идей.

Чек-лист чаще всего представляет собой обычный и привычный нам список, который может быть:

- Списком, в котором последовательность пунктов не имеет значения (например, список значений некоего поля).
- Списком, в котором последовательность пунктов важна (например, шаги в краткой инструкции).
- Структурированным (многоуровневым) списком (вне зависимости от учёта последовательности пунктов), что позволяет отразить иерархию идей.

Важно понять, что нет и не может быть никаких запретов и ограничений при разработке чек-листов — главное, чтобы они помогали в работе. Иногда чек-листы могут даже выражаться графически (например, с использованием ментальных карт<sup>283</sup> или концепт-карт<sup>284</sup>), хотя традиционно их составляют в виде многоуровневых списков.

Поскольку в разных проектах встречаются однотипные задачи, хорошо продуманные и аккуратно оформленные чек-листы могут использоваться повторно, чем достигается экономия сил и времени.

Для того чтобы чек-лист был действительно полезным инструментом, он должен обладать рядом важных свойств.

**Логичность.** Чек-лист пишется не «просто так», а на основе целей и для того, чтобы помочь в достижении этих целей. К сожалению, одной из самых частых и опасных ошибок при составлении чек-листа является превращение его в свалку мыслей, которые никак не связаны друг с другом.

**Последовательность и структурированность.** Со структурированностью всё достаточно просто — она достигается за счёт оформления чек-листа в виде многоуровневого списка. Что до последовательности, то даже в том случае, когда пункты чек-листа не описывают цепочку действий, человеку всё равно удобнее воспринимать информацию в виде неких небольших групп идей, переход между которыми является понятным и очевидным (например, сначала можно прописать идеи

<sup>281</sup> Понятие «чек-листа» не завязано на тестирование как таковое — это совершенно универсальная техника, которая может применяться в любой без исключения области жизни. В русском языке вне контекста информационных технологий чаще используется понятное и привычное слово «список» (например, «список покупок», «список дел» и т.д.), но в тестировании прижилась калькированная с английского версия — «чек-лист».

<sup>282</sup> Если у вас возник вопрос «почему тут использованы квадратные скобки», ознакомьтесь с синтаксисом «расширенной формы Бэкуса-Наура», который де-факто является стандартом описания выражений в ИТ. См. «Extended Backus-Naur form», Wikipedia. [[https://en.wikipedia.org/wiki/Extended\\_Backus%E2%80%93Naur\\_form](https://en.wikipedia.org/wiki/Extended_Backus%E2%80%93Naur_form)]

<sup>283</sup> «Mind map», Wikipedia. [[http://en.wikipedia.org/wiki/Mind\\_map](http://en.wikipedia.org/wiki/Mind_map)]

<sup>284</sup> «Concept map», Wikipedia. [[http://en.wikipedia.org/wiki/Concept\\_map](http://en.wikipedia.org/wiki/Concept_map)]

простых позитивных тест-кейсов<sup>(77)</sup>, потом идеи простых негативных тест-кейсов, потом постепенно повышать сложность тест-кейсов, но не стоит писать эти идеи вперемешку).

**Полнота и избыточность.** Чек-лист должен представлять собой аккуратную «сухую выжимку» идей, в которых нет дублирования (часто появляется из-за разных формулировок одной и той же идеи), и в то же время ничто важное не упущено.

Правильно создавать и оформлять чек-листы также помогает восприятие их не только как хранилища наборов идей, но и как «требования для составления тест-кейсов». Эта мысль приводит к пересмотру и переосмыслению свойств качественных требований (см. главу «Свойства качественных требований»<sup>(41)</sup>) в применении к чек-листам.



**Задание 2.4.а:** перечитайте главу «Свойства качественных требований»<sup>(41)</sup> и подумайте, какие свойства качественных требований можно также считать и свойствами качественных чек-листов.

Итак, рассмотрим процесс создания чек-листа. В главе «Пример анализа и тестирования требований»<sup>(49)</sup> приведён пример итоговой версии требований<sup>(55)</sup>, который мы и будем использовать.

Поскольку мы не можем сразу «протестировать всё приложение» (это слишком большая задача, чтобы решить её одним махом), нам уже сейчас нужно выбрать некую логику построения чек-листов — да, их будет несколько (в итоге их можно будет структурированно объединить в один, но это не обязательно).

Типичными вариантами такой логики является создание отдельных чек-листов для:

- различных уровней функционального тестирования<sup>(74)</sup>;
- отдельных частей (модулей и подмодулей<sup>(120)</sup>) приложения;
- отдельных требований, групп требований, уровней и типов требований<sup>(36)</sup>;
- типичных пользовательских сценариев<sup>(141)</sup>;
- частей или функций приложения, наиболее подверженных рискам.

Этот список можно расширять и дополнять, можно комбинировать его пункты, получая, например, чек-листы для проверки наиболее типичных сценариев, затрагивающих некую часть приложения.

Чтобы проиллюстрировать принципы построения чек-листов, мы воспользуемся логикой разбиения функций приложения по степени их важности на три категории (см. классификацию по убыванию степени важности функций приложения<sup>(74)</sup>):

- Базовые функции, без которых существование приложения теряет смысл (т.е. самые важные — то, ради чего приложение вообще создавалось), или нарушение работы которых создаёт объективные серьёзные проблемы для среды исполнения. (См. «Дымовое тестирование»<sup>(74)</sup>).
- Функции, востребованные большинством пользователей в их повседневной работе. (См. «Тестирование критического пути»<sup>(75)</sup>).
- Остальные функции (разнообразные «мелочи», проблемы с которыми не сильно повлияют на ценность приложения для конечного пользователя). (См. «Расширенное тестирование»<sup>(76)</sup>).

## Функции, без которых существование приложения теряет смысл

Сначала приведём целиком весь чек-лист для дымового тестирования, а потом разберём его подробнее.

- Конфигурирование и запуск.
- Обработка файлов:

		Форматы входных файлов		
		ТХТ	HTML	MD
Кодировки входных файлов	WIN1251	+	+	+
	CP866	+	+	+
	KOI8R	+	+	+

- Остановка.

Да, и всё. Здесь перечислены все ключевые функции приложения.

**Конфигурирование и запуск.** Если приложение невозможно настроить для работы в пользовательской среде, оно бесполезно. Если приложение не запускается, оно бесполезно. Если на стадии запуска возникают проблемы, они могут негативно отразиться на функционировании приложения и потому также заслуживают пристального внимания.

Примечание: в нашем примере мы столкнулись со скорее нетипичным случаем — приложение конфигурируется параметрами командной строки, а потому разделить операции «конфигурирования» и «запуска» не представляется возможным; в реальной жизни для подавляющего большинства приложений эти операции выполняются раздельно.

**Обработка файлов.** Ради этого приложение и разрабатывалось, потому здесь даже на стадии создания чек-листа мы не поленились создать матрицу, отражающую все возможные комбинации допустимых форматов и допустимых кодировок входных файлов, чтобы ничего не забыть и подчеркнуть важность соответствующих проверок.

**Остановка.** С точки зрения пользователя эта функция может не казаться столь уж важной, но остановка (и запуск) любого приложения связаны с большим количеством системных операций, проблемы с которыми могут привести к множеству серьёзных последствий (вплоть до невозможности повторного запуска приложения или нарушения работы операционной системы).

## Функции, востребованные большинством пользователей

Следующим шагом мы будем выполнять проверку того, как приложение ведёт себя в обычной повседневной жизни, пока не затрагивая экзотические ситуации. Очень частым вопросом является допустимость дублирования проверок на разных уровнях функционального тестирования<sup>[74]</sup> — можно ли так делать. Одно временно и «нет», и «да». «Нет» в том смысле, что не допускается (не имеет смысла) повторение тех же проверок, которые только что были выполнены. «Да» в том смысле, что любую проверку можно детализировать и снабдить дополнительными деталями.

- 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 "\").

- Значение LOG\_FILE\_NAME не указано.
- 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:

		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		

- Недоступные входные файлы:
  - Нет прав доступа.
  - Файл открыт и заблокирован.
  - Файл с атрибутом «только для чтения».
- Остановка:
  - Закрытием окна консоли.
- Журнал работы приложения:
  - Автоматическое создание (при отсутствии журнала).
  - Продолжение (дополнение журнала) при повторных запусках.
- Производительность:
  - Элементарный тест с грубой оценкой.

Обратите внимание, что чек-лист может содержать не только «предельно сжатые тезисы», но и вполне развёрнутые комментарии, если это необходимо — лучше пояснить суть идеи подробнее, чем потом гадать, что же имелось в виду.

Также обратите внимание, что многие пункты чек-листа носят весьма высокоуровневый характер, и это нормально. Например, «повреждения в допустимом формате» (см. матрицу с кодировками, форматами и размерами) звучит расплывчато, но этот недочёт будет устранён уже на уровне полноценных тест-кейсов.

## Остальные функции и особые сценарии

Пришло время обратить внимание на разнообразные мелочи и хитрые нюансы, проблемы с которыми едва ли сильно озаботят пользователя, но формально всё же будут считать ошибками.

- 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.



**Задание 2.4.b:** возможно, вам захотелось что-то изменить в приведённых выше чек-листах — это совершенно нормально и справедливо: нет и не может быть некоего «единственно верного идеального чек-листа», и ваши идеи вполне имеют право на существование, потому составьте свой собственный чек-лист или отметьте недостатки, которые вы заметили в приведённом выше чек-листе.

Как мы увидим в следующей главе, создание качественного тест-кейса может потребовать длительной кропотливой и достаточно монотонной работы, которая при этом не требует от опытного тестировщика сильных интеллектуальных усилий, а потому переключение между работой над чек-листами (творческая составляющая) и расписыванием их в тест-кейсы (механическая составляющая) позволяет разнообразить рабочий процесс и снизить утомляемость. Хотя, конечно, написание сложных и притом качественных тест-кейсов может оказаться ничуть не менее творческой работой, чем продумывание чек-листов.

## 2.4.2. Тест-кейс и его жизненный цикл

### 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** (test<sup>285</sup>) — 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 case<sup>286</sup>) — 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<sup>(147)</sup>, 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 прочесть 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 case<sup>287</sup>) — 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<sup>(72)</sup> и системном тестировании<sup>(73)</sup>, а также на уровне дымового тестирования<sup>(74)</sup>. Может служить отправной точкой для проведения исследовательского тестирования<sup>(80)</sup> или для создания низкоуровневых тест-кейсов.

<sup>285</sup> **Test.** A set of one or more test cases. [ISTQB Glossary]

<sup>286</sup> **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]

<sup>287</sup> **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 case<sup>288</sup>) — 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 specification<sup>289</sup>) — документ, описывающий набор тест-кейсов (включая их цели, входные данные, условия и шаги выполнения, ожидаемые результаты) для тестируемого элемента (test item<sup>290</sup>, test object<sup>291</sup>).

**Спецификация теста** (test specification<sup>292</sup>) — документ, состоящий из спецификации тест-дизайна (test design specification<sup>293</sup>), спецификации тест-кейса (test case specification<sup>289</sup>) и/или спецификации тест-процедуры (test procedure specification<sup>294</sup>).

**Тест-сценарий** (test scenario<sup>295</sup>, 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<sup>{141}</sup>.

## 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 coverage<sup>296</sup> 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-

<sup>288</sup> **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]

<sup>289</sup> **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]

<sup>290</sup> **Test item.** The individual element to be tested. There usually is one test object and many test items. [ISTQB Glossary]

<sup>291</sup> **Test object.** The component or system to be tested. [ISTQB Glossary]

<sup>292</sup> **Test specification.** A document that consists of a test design specification, test case specification and/or test procedure specification. [ISTQB Glossary]

<sup>293</sup> **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]

<sup>294</sup> **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]

<sup>295</sup> **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]

<sup>296</sup> **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<sup>[82]</sup> и повторное тестирование<sup>[82]</sup> (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<sup>[47]</sup>).
- Быстро вводить в курс дела нового сотрудника, недавно подключившегося к проекту.

### Жизненный цикл тест-кейса

В отличие от отчёта о дефекте, у которого есть полноценный развитый жизненный цикл<sup>[165]</sup>, для тест-кейса речь скорее идёт о наборе состояний (см. рисунок 2.4.а), в которых он может находиться (**жирным** шрифтом отмечены наиболее важные состояния).

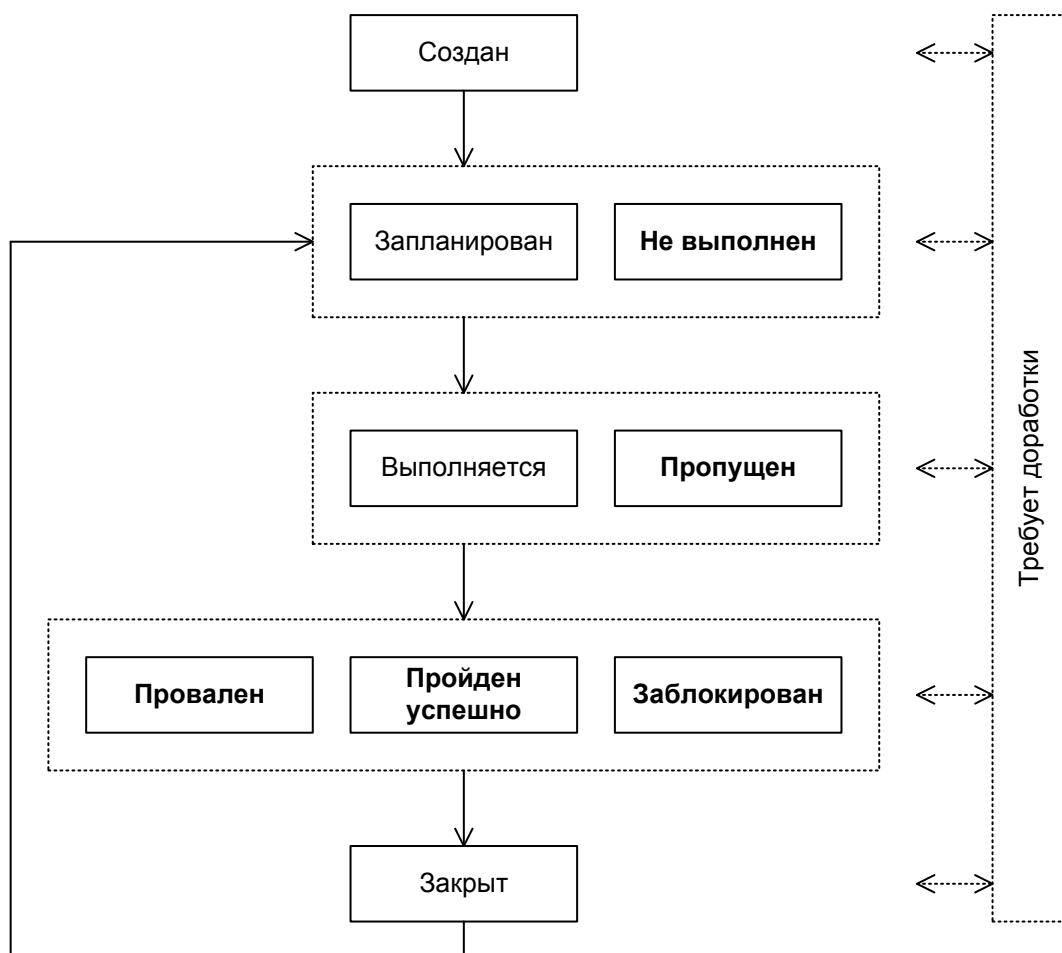


Рисунок 2.4.а — Жизненный цикл (набор состояний) тест-кейса

- **Создан** (new) — типичное начальное состояние практически любого артефакта. Тест-кейс автоматически переходит в это состояние после создания.
- **Запланирован** (planned, ready for testing) — в этом состоянии тест-кейс находится, когда он или явно включён в план ближайшей итерации тестирования, или как минимум готов для выполнения.



- **Не выполнен (not tested)** — в некоторых системах управления тест-кейсами это состояние заменяет собой предыдущее («запланирован»). Нахождение тест-кейса в данном состоянии означает, что он готов к выполнению, но ещё не был выполнен.
- **Выполняется (work in progress)** — если тест-кейс требует длительного времени на выполнение, он может быть переведён в это состояние для подчёркивания того факта, что работа идёт, и скоро можно ожидать её результатов. Если выполнение тест-кейса занимает мало времени, это состояние, как правило, пропускается, а тест-кейс сразу переводится в одно из трёх следующих состояний — «провален», «пройден успешно» или «заблокирован».
- **Пропущен (skipped)** — бывают ситуации, когда выполнение тест-кейса отменяется по соображениям нехватки времени или изменения логики тестирования.
- **Провален (failed)** — данное состояние означает, что в процессе выполнения тест-кейса был обнаружен дефект, заключающийся в том, что ожидаемый результат по как минимум одному шагу тест-кейса не совпадает с фактическим результатом. Если в процессе выполнения тест-кейса был «случайно» обнаружен дефект, никак не связанный с шагами тест-кейса и их ожидаемыми результатами, тест-кейс считается пройденным успешно (при этом, естественно, по обнаруженному дефекту создаётся отчёт о дефекте).
- **Пройден успешно (passed)** — данное состояние означает, что в процессе выполнения тест-кейса не было обнаружено дефектов, связанных с расхождением ожидаемых и фактических результатов его шагов.
- **Заблокирован (blocked)** — данное состояние означает, что по какой-то причине выполнение тест-кейса невозможно (как правило, такой причиной является наличие дефекта, не позволяющего реализовать некий пользовательский сценарий).
- **Закрыт (closed)** — очень редкий случай, т.к. тест-кейс, как правило, оставляют в состояниях «провален / пройден успешно / заблокирован / пропущен». В данное состояние в некоторых системах управления тест-кейсами переводят, чтобы подчеркнуть тот факт, что на данной итерации тестирования все действия с ним завершены.
- **Требуется доработка (not ready)** — как видно из схемы, в это состояние (и из него) тест-кейс может быть переведён в любой момент времени, если в нём будет обнаружена ошибка, если изменятся требования, по которым он был написан, или наступит иная ситуация, не позволяющая считать тест-кейс пригодным для выполнения и перевода в иные состояния.

Ещё раз подчеркнём, что в отличие от жизненного цикла дефекта, который достаточно стандартизирован и формализован, для тест-кейса описанное выше носит общий рекомендательный характер, рассматривается скорее как разрозненный набор состояний (а не строгий жизненный цикл) и может сильно отличаться в разных компаниях (в связи с имеющимися традициями и/или возможностями систем управления тест-кейсами).

### 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-кейсами 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.b.

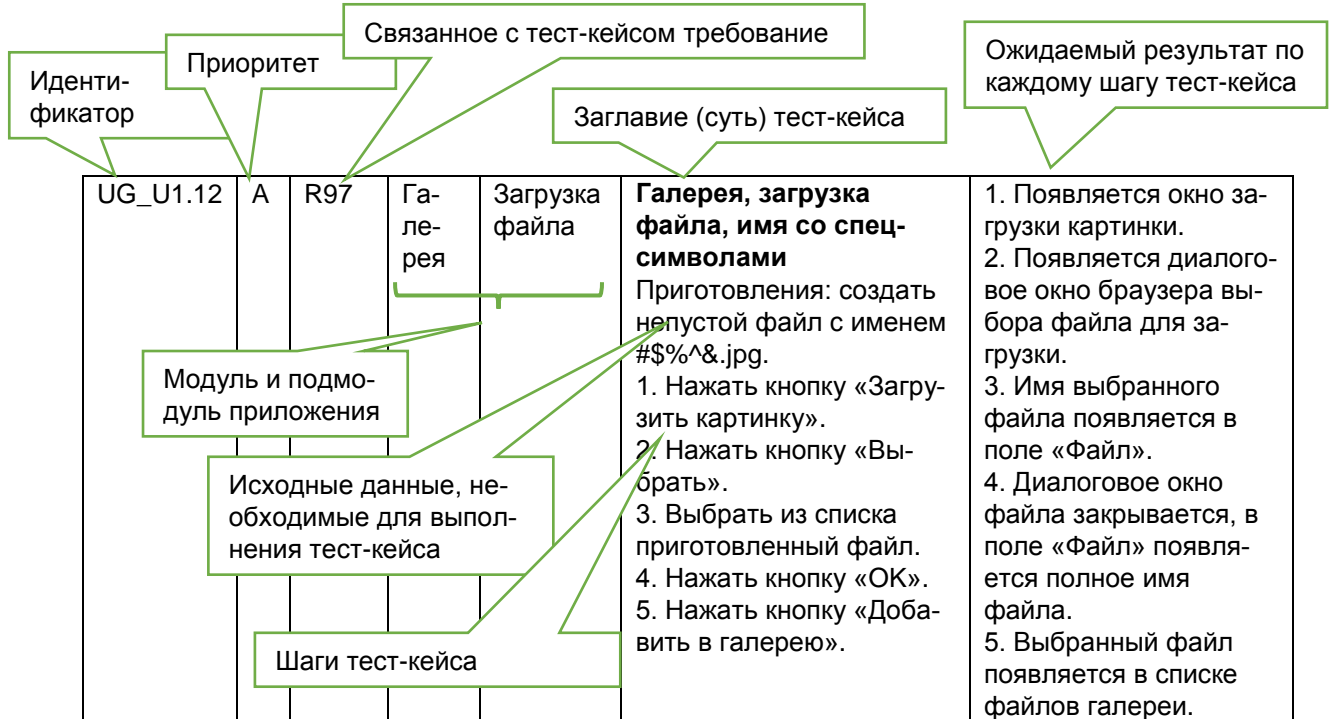


Рисунок 2.4.b — Общий вид тест-кейса

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 (например: UR216\_S12\_DB\_Neg).

**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<sup>(141)</sup> или функции, с которыми связан тест-кейс;
- потенциальной важностью дефекта<sup>(174)</sup>, 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<sup>(138)</sup>.

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<sup>(55)</sup> 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<sup>[138](#)</sup>.

**Заглавие (суть) тест-кейса** (title) призвано упростить и ускорить понимание основной идеи (цели) 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
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.

То, 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».



Некоторые авторы не следуют этой логике и допускают в приготовлениях работу с тестируемым приложением. И здесь нет «правильного варианта» — просто в одной традиции решено одним образом, в другой — другим. Во многом это — ещё и терминологическая проблема: «preparation», «initial data» и «setup» вполне логично выполнять без участия тестируемого приложения, в то время как «precondition» по смыслу ближе к описанию состояния тестируемого приложения. В реальной рабочей обстановке вам достаточно будет прочитать несколько тест-кейсов, уже созданных вашими коллегами, чтобы понять, какой точки зрения на данный вопрос они придерживаются.

**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»), в английском языке не надо использовать частицу «to» (т.е. «запустить приложение» будет «start application», **не** «to start application»);
- соотносите степень детализации шагов и их параметров с целью тест-кейса, его сложностью, уровнем<sup>[74]</sup> 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»<sup>{155}</sup>.

#### 2.4.4. Инструментальные средства управления тестированием

Инструментальных средств управления тестированием (test management tool<sup>[297]</sup>) очень много<sup>298</sup>, 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<sup>[27]</sup>. 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<sup>[119]</sup> 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).

---

<sup>297</sup> 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]

<sup>298</sup> «Test management tools» [<http://www.opensourcetesting.org/testmgt.php>]

**QAComplete<sup>299</sup>**

The screenshot shows the QAComplete test case creation form. It includes fields for Id (Auto Generated), Title, Priority, Folder Name, Status, Assigned To, Avg Run Time (Auto Calculated), Last Run Test Set (Auto Calculated), Last Run Release, Last Run Status (Auto Calculated), Last Run Configuration, Description (with a rich text editor), Owner, Version, Execution Type, Test Type, Latest Notes (with an 'Add New Note' link), Default Host Name, Linked Items (with a 'Link to Items...' link), and File Attachments (with a table of Reset and Browse buttons). Numbered callouts 1 through 19 point to specific elements: 1 (Id), 2 (Title), 3 (Priority), 4 (Folder Name), 5 (Status), 6 (Assigned To), 7 (Last Run Status), 8 (Last Run Configuration), 9 (Avg Run Time), 10 (Last Run Test Set), 11 (Last Run Release), 12 (Description), 13 (Rich text editor), 14 (Owner), 15 (Version), 16 (Test Type), 17 (Latest Notes), 18 (Linked Items), 19 (File Attachments), and 10 (Rich text editor toolbar).

Рисунок 2.4.с — 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

<sup>299</sup> 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 (дымовой).
  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.d — Добавление шагов тест-кейса в QAComplete

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

TestLink<sup>300</sup>

The screenshot shows the 'Create Test Case' interface in TestLink. It features a title field (1), a large summary text area (2), and two side-by-side text areas for 'Steps' (3) and 'Expected Results' (4). At the bottom, there are two list boxes for 'Available Keywords' (5) and 'Assigned Keywords' (6), connected by a set of arrows. Each field has a rich text editor toolbar above it. A 'Create' button is located at the top right and bottom right of the form.

Рисунок 2.4.е — 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.

<sup>300</sup> TestLink [<http://sourceforge.net/projects/testlink/>]

TestRail<sup>301</sup>

**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.f — 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

<sup>301</sup> 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, точность и единообразие формулировок.** 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»<sup>[119]</sup>), 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);
- везде называйте одни и те же вещи одинаково (например, нельзя в одном тест-кейсе некий режим работы приложения назвать «графическое представление», а в другом тот же режим — «визуальное отображение», т.к. многие люди могут подумать, что речь идёт о разных вещах);
- следуйте принятому на проекте стандарту оформления и написания тест-кейсов (иногда такие стандарты могут быть весьма жёсткими: вплоть до регламентации того, названия каких элементов должны быть приведены в двойных кавычках, а каких — в одинарных).

**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
<b>Конвертация из всех поддерживаемых кодировок</b> Приготовление: <ul style="list-style-type: none"> <li>• Создать папки C:/A, C:/B, C:/C, C:/D.</li> <li>• Разместить в папке C:/D файлы 1.html, 2.txt, 3.md из прилагаемого архива.</li> </ul> <ol style="list-style-type: none"> <li>1. Запустить приложение, выполнив команду «php converter.php c:/a c:/b c:/converter.log».</li> <li>2. Скопировать файлы 1.html, 2.txt, 3.md из папки C:/D в папку C:/A.</li> <li>3. Остановить приложение нажатием Ctrl+C.</li> </ol>	<ol style="list-style-type: none"> <li>1. Отображается консольный журнал приложения с сообщением «текущее_время started, source dir c:/a, destination dir c:/b, log file 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:/converter.log».</li> <li>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)».</li> <li>3. В файле C:/C/converter.log появляется запись «текущее_время closed». Приложение завершает работу.</li> </ol>

## Тест-кейс 2:

Шаги	Ожидаемые результаты
<b>Конвертация из всех поддерживаемых кодировок</b> 1. Выполнить конвертацию трёх файлов допустимого размера в трёх разных кодировках всех трёх допустимых форматов.	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<sup>[116]</sup> и высокоуровневых<sup>[115]</sup> 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
<b>Конвертация из всех поддерживаемых кодировок</b> Приготовление: <ul style="list-style-type: none"> <li>• Создать в корне любого диска четыре отдельные папки для входных файлов, выходных файлов, файла журнала и временного хранения тестовых файлов.</li> <li>• Распаковать содержимое прилагаемого архива в папку для временного хранения тестовых файлов.</li> </ul> 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.	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 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
<b>Запуск приложения</b> 1. Запустить приложение.	1. Приложение запускается.

Слишком сложный тест-кейс:

Шаги	Ожидаемые результаты
<b>Повторная конвертация</b> Приготовление: <ul style="list-style-type: none"> <li>• Создать в корне любого диска три отдельные папки для входных файлов, выходных файлов, файла журнала.</li> <li>• Подготовить набор из нескольких файлов максимального поддерживаемого размера поддерживаемых форматов с поддерживаемыми кодировками, а также нескольких файлов допустимого размера, но недопустимого формата.</li> </ul> 1. Запустить приложение, указав в параметрах соответствующие пути из приготовления к тесту (имя файла журнала — произвольное).	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.

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 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.	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.
--	--

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<sup>(132)</sup> из пункта про специфичность и общность.

Пример хорошего сложного тест-кейса может выглядеть так:

Шаги	Ожидаемые результаты
<b>Много копий приложения, конфликт файловых операций</b> Приготовление: <ul style="list-style-type: none"> <li>Создать в корне любого диска три отдельные папки для входных файлов, выходных файлов, файла журнала.</li> <li>Подготовить набор из нескольких файлов максимального поддерживаемого размера поддерживаемых форматов с поддерживаемыми кодировками.</li> </ul> <ol style="list-style-type: none"> <li>Запустить первую копию приложения, указав в параметрах соответствующие пути из приготовления к тесту (имя файла журнала — произвольное).</li> <li>Запустить вторую копию приложения с теми же параметрами (см. шаг 1).</li> <li>Запустить третью копию приложения с теми же параметрами (см. шаг 1).</li> <li>Изменить приоритет процессов второй ("high") и третьей ("low") копий.</li> <li>Скопировать подготовленный набор исходных файлов в папку для входных файлов.</li> </ol>	<ol style="list-style-type: none"> <li>Все три копии приложения запускаются, в файле журнала появляются последовательно три записи о запуске приложения.</li> <li>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"> <li>"source file inaccessible, retrying".</li> <li>"destination file inaccessible, retrying".</li> <li>"log file inaccessible, retrying".</li> </ol> </li> </ol> <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<sup>(74)</sup>, 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
<b>Запуск и остановка приложения</b> 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
<b>Запуск с некорректными параметрами, несуществующие пути</b> 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
<b>Конвертация из всех поддерживаемых кодировок</b> Приготовления: <ul style="list-style-type: none"> <li>Создать в корне любого диска четыре отдельные папки для входных файлов, выходных файлов, файла журнала и временного хранения тестовых файлов.</li> <li>Распаковать содержимое прилагаемого архива в папку для временного хранения тестовых файлов.</li> </ul> <ol style="list-style-type: none"> <li>Запустить приложение, указав в параметрах соответствующие пути из приготовления к тесту (имя файла журнала — произвольное).</li> <li>Скопировать файлы из папки для временного хранения в папку для входных файлов.</li> <li>Остановить приложение.</li> <li>Удалить файл журнала.</li> <li>Повторно запустить приложение с теми же параметрами.</li> <li>Остановить приложение.</li> </ol>	<ol style="list-style-type: none"> <li>Приложение запускается и выводит сообщение о своём запуске в консоль и файл журнала.</li> <li>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.</li> <li>Приложение выводит сообщение о завершении работы в файл журнала и завершает работу.</li> <li>Приложение запускается и выводит сообщение о своём запуске в консоль и заново созданный файл журнала.</li> <li>Приложение выводит сообщение о завершении работы в файл журнала и завершает работу.</li> </ol>

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

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

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 набор<sup>(141)</sup> 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 тестировании<sup>302</sup> с использованием фреймворков наподобие JUnit или TestNG — там существует специальный «механизм фиксации» (fixture), автоматически выполняющий указанные действия перед каждым отдельным тестовым методом (или их совокупности) или после него.

**Неизбыточность по отношению к другим тест-кейсам.** В процессе создания множества тест-кейсов очень легко оказаться в ситуации, когда два и более тест-кейса фактически выполняют одни и те же проверки, преследуют одни и те же цели, направлены на поиск одних и тех же проблем. Способ минимизации количества таких тест-кейсов подробно описан в главе «Виды и направления тестирования<sup>(62)</sup>» (см. такие техники тестирования, как использование классов эквивалентности<sup>(89)</sup> и граничных условий<sup>(90)</sup>).

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.

**Демонстративность (способность демонстрировать обнаруженную ошибку очевидным образом).** Ожидаемые результаты должны быть подобраны и сформулированы таким образом, чтобы любое отклонение от них сразу же бросалось в глаза и становилось очевидным, что произошла ошибка. Сравните выдержки из двух тест-кейсов.

Выдержка из недемонстративного тест-кейса:

Шаги	Ожидаемые результаты
<ol style="list-style-type: none"> <li>5. Разместить в файле текст «Пример длинного текста, содержащего символы русского и английского алфавита вперемешку.» в кодировке KOI8-R (в слове «Пример» буквы «р» — английские).</li> <li>6. Сохранить файл под именем «test.txt» и отправить файл на конвертацию.</li> <li>7. Переименовать файл в «test.txt».</li> </ol>	<ol style="list-style-type: none"> <li>6. Приложение игнорирует файл.</li> <li>7. Текст принимает корректный вид в кодировке UTF-8 с учётом английских букв.</li> </ol>

<sup>302</sup> **Unit testing (component testing).** The testing of individual software components. [ISTQB Glossary]

## Выдержка из демонстративного тест-кейса:

Шаги	Ожидаемые результаты
5. Разместить в файле текст «ЁПРП» (Эти символы представляют собой словосочетание «Пример текста.» в кодировке KOI8-R, прочитанной как CP866).	6. Текст принимает вид: «Пример текста.» (кодировка UTF8).
6. Отправить файл на конвертацию.	

В первом случае тест-кейс плох не только расплывчатостью формулировки «корректный вид в кодировке UTF-8 с учётом английских букв», там также очень легко допустить ошибки при выполнении:

- 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.

Второй тест-кейс чётко ориентирован на свою цель по проверке конвертации (не содержит странной проверки с игнорированием файла с неверным расширением) и описан так, что его выполнение не представляет никаких сложностей, а любое отклонение фактического результата от ожидаемого будет сразу же заметно.

**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<sup>(119)</sup> («Ссылка на требование», «Модуль», «Подмодуль»), но и сама логика тест-кейса играет не последнюю роль, т.к. в случае серьёзных 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		<b>Совмещение кодировок</b> 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<sup>(55)</sup>).
- 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	<b>Запуск с некорректными параметрами, несуществующие пути</b> 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.

Некоторые авторы также подчёркивают, что прослеживаемость тест-кейса связана с его неизбыточностью<sup>(137)</sup> по отношению к другим тест-кейсам (намного проще дать ссылку на один уникальный тест-кейс, чем выбирать из нескольких очень похожих).

**Возможность повторного использования.** Это свойство редко выполняется для низкоуровневых тест-кейсов<sup>(116)</sup>, но при создании высокоуровневых тест-кейсов<sup>(115)</sup> можно добиться таких формулировок, при которых:

- тест-кейс будет пригодным к использованию с различными настройками тестируемого приложения и в различных тестовых окружениях;
- 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
<b>Запуск, все параметры некорректны</b> 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.

**Повторяемость.** Тест-кейс должен быть сформулирован таким образом, чтобы при многократном повторении он показывал одинаковые результаты. Это свойство можно разделить на два подпункта:

- во-первых, даже общие формулировки, допускающие разные варианты выполнения тест-кейса, должны очерчивать соответствующие явные границы (например: «ввести какое-нибудь число» — плохо, «ввести целое число в диапазоне от -273 до +500 включительно» — хорошо);
- действия (шаги) тест-кейса по возможности не должны приводить к необратимым (или сложно обратимым) последствиям (например: удалению данных, нарушению конфигурации окружения и т.д.) — не стоит включать в тест-кейс



такие «разрушительные действия», если они не продиктованы явным образом целью тест-кейса; если же цель тест-кейса обязывает нас к выполнению таких действий, в самом тест-кейсе должно быть описание действий по восстановлению исходного состояния приложения (данных, окружения).

**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»<sup>(155)</sup>.

## 2.4.6. Наборы тест-кейсов

### Терминология и общие сведения

	<b>Набор тест-кейсов</b> (test case suite <sup>303</sup> , 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.
	<b>Внимание!</b> Из-за особенностей перевода очень часто вместо «набор тест-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 <sup>(36)</sup> . 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 сценарии<sup>304</sup> (или сценарии использования), представляющие собой цепочки действий, выполняемых пользователем в определённой ситуации для достижения определённой цели.

<sup>303</sup> **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]

<sup>304</sup> 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>]