Тестирование ПО. Уровень 1

Разработка тест-кейсов

Определение и структура тест-кейсов. Характеристики хорошего теста. Аксиомы тестирования. Поддерживаемость тест-кейсов. Тест-комплекты. Чек-листы. Подготовка тестовых данных.

Оглавление

[Чек-лист](#_vzsr3n2gsfon)

[Какие поля включает чек-лист](#_q6yvzziop2qc)

[Тест-кейс](#_tyjcwt)

[Что такое тест-кейс?](#_3dy6vkm)

[Цели написания тест-кейсов](#_vyjb5lvrm0qa)

[Атрибуты (или поля) тест-кейса](#_p2n732l10tet)

[Свойства качественных тест-кейсов](#_v92hgqwkcfjz)

[Типичные ошибки](#_1gaqztms1ap3)

[Ошибки оформления и формулировок](#_mld8h0c28s50)

[Логические ошибки](#_wt9aggzemtsz)

[Наборы тест-кейсов](#_x2aeolz8ip57)

[Пример тест-кейса для успешной авторизации](#_lmswjlbbyvrl)

[Разница между тест-кейсом и дефектом](#_wuous73pww18)

[Принципы тестирования](#_ba4sjzss1vos)

[Подходы к тестированию](#_ime04wlxpa7z)

[Глоссарий](#_tydynsve21dp)

[Практическое задание:](#_pv7nlhdipoon)

[Требования к выполненной работе](#_form9hhk9ql9)

[Дополнительные материалы](#_1t3h5sf)

[Используемые источники](#_17dp8vu)

Используемые источники

# Чек-лист

В глоссарии ISTQB нет такого понятия, как чек-лист (check list), но рассматривается такая техника тестирования, как checklist-based testing, или тестирование, основанное на чек-листе.

**Тестирование на основе чек-листа** (checklist-based testing) — методика, основанная на опыте, при которой опытный тестировщик использует список высокоуровневых пунктов, которые необходимо записать, проверить или запомнить, или набор правил или критериев, в соответствии с которыми должен быть проверен продукт.

Исходя из этого, **чек-лист** (check list) — набор/лист проверок, который описан верхнеуровнево, без детализации.

То есть это набор последовательных проверок, которые записаны без излишних деталей. В которых очень важна последовательность и структурированность.

Этот список проверок может быть составлен на разных этапах. К примеру, на этапе тестирования требований. Здесь мы уже можем составлять первичный набор проверок, которые необходимо будет выполнить.

Что важно при составлении чек-листа?

1. Логичность чек-листа. То есть набор проверок в чек-листе составляется для какой-то определенной цели, например, протестировать форму регистрации или авторизации.
2. Последовательность. Это означает, что проверки должны идти логично друг за другом, а не вразнобой. К примеру, при составлении чек-листа для тестирования формы регистрации мы будем последовательно составлять проверки.
3. Структурированность. Составление чек-листа в соответствии с какой-то логичной структурой. Понятно, что для тестирования веб-приложения структура чек-листа будет соответствовать структуре самого приложения.
4. Полнота и неизбыточность. Поскольку чек-лист — набор высокоуровневых проверок, излишняя детализация здесь не нужна. Для деталей у нас есть тест-кейсы. Но и все проверки должны быть учтены и представлены в чек листе.

При построении чек-листов пользуемся логикой разбиения функций приложения по степени их важности на три категории (см. классификацию по убыванию степени важности функций приложения).

Вспомним классификацию по убыванию степени важности функций приложения:

1. Дымовое тестирование.
2. Тестирование критического пути.
3. Расширенное тестирование.

То есть мы составляем чек-лист сначала для базовых функций, без которых существование приложения теряет смысл (т. е. самые важные — то, ради чего приложение вообще создавалось), или тех, нарушение работы которых создает объективные серьезные проблемы для среды исполнения.

Далее составляем чек-лист для функций, востребованных большинством пользователей в их повседневной работе.

И в последнюю очередь составляем чек-лист для остальных функций (разнообразные мелочи, проблемы с которыми не сильно повлияют на ценность приложения для конечного пользователя), менее востребованных большинством пользователей.

К примеру, один из первых чек-листов может составляться для формы регистрации на сайте.



Следующим может быть чек-лист для формы авторизации.

Чек-листы можно хранить, используя специальные инструменты наподобие checkvist, можно создавать чек-листы в таблицах Excel, в Confluence. На каждом проекте может быть свой метод или подход работы с чек-листами.

## Какие поля включает чек-лист

На этапе создания чек-листа:

|  |  |  |
| --- | --- | --- |
| **#** | **Проверка** | **Ожидаемый результат** |
| Страница регистрации | | |
| 1 |  |  |
| 2 |  |  |
| 3 |  |  |
| Страница авторизации | | |
| 1 |  |  |
| 2 |  |  |
| 3 |  |  |
| Главная страница | | |
| 1 |  |  |
| 2 |  |  |
| 3 |  |  |

На этапе выполнения чек-листа (тестированию по чек-листу):

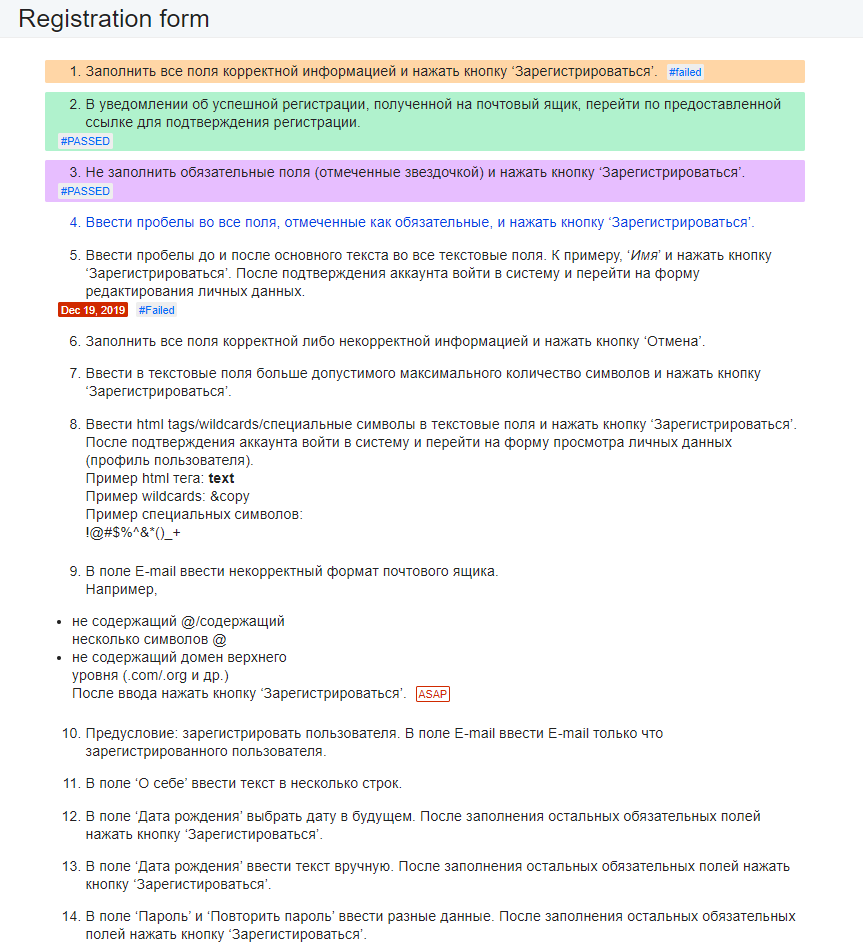
|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Проверка** | **Ожидаемый результат** | **Pass/Failed** |
| Страница регистрации | | | |
| 1 |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |
| Страница авторизации | | | |
| 1 |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |
| Главная страница | | | |
| 1 |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |

В случае, если проект и реализация часто меняется, возможно есть смысл составлять набор проверок без ожидаемого результата. Всё будет зависеть от специфики проекта, и здесь главное — знать, в каком виде вы можете создавать чек-листы и для чего они в целом предназначены.

Для отработки практики составления чек-листов вы можете использовать таблицу Excel, либо воспользоваться [бесплатным ресурсом для создания и хранения чек-листов](https://checkvist.com/).

Как правило, на основании чек-листов создаются тест-кейсы.

Ниже представлен пример чек листа составленный в checkvist:



# Тест-кейс

## Что такое тест-кейс?

В соответствии с ISTQB, **тест-кейс, или тестовый случай** (test case) — набор предварительных условий, входных данных, действий (где применимо), ожидаемых результатов и постусловий, разработанных на тестовых условиях.

Тест-кейс пишется для проверки того или иного свойства или поведения программы. К примеру, может ли пользователь зарегистрироваться, используя валидные данные, или нет? Может пользователь авторизоваться в системе после успешной регистрации или нет? Может ли пользователь успешно произвести покупку, авторизуясь под своими учётными данными или нет? И так далее.

На начальном этапе обучения рассмотрим два вида тест-кейсов:

* Высокоуровневый тест-кейс.
* Низкоуровневый тест-кейс.

**Высокоуровневый тест-кейс** — тест-кейс без конкретных входных данных и ожидаемых результатов.

Высокоуровневый тест-кейс ограничивается общими идеями и операциями. По своей сути он похож с подробно описанным пунктом чек-листа. В условиях ограниченного времени на проекте, может использоваться именно этот вид тест-кейсов. Часто может встречаться в интеграционном или системном тестировании, а также на уровне дымового тестирования. Может служить как идея для проведения исследовательского тестирования или дальнейшего создания низкоуровневых тест-кейсов. Встречается и такое, что шаги тест-кейса объединяют, чтобы уменьшить их количество.

**Низкоуровневый тест-кейс** — тест-кейс с конкретными входными данными и ожидаемыми результатами.

Низкоуровневые тест-кейсы — классический вид тест-кейсов, в котором указываются все предварительные условия, входные данные, и выделяют каждый шаг отдельно. 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.

## Цели написания тест-кейсов

В первую очередь мы пишем тест-кейсы чтобы:

1. Структурировать и систематизировать подход к тестированию. Облегчить и улучшить процесс тестирования на проекте. Без этого крупный проект почти гарантированно обречён на провал.
2. Вычислять метрики тестового покрытия. Принимать меры по его увеличению.
3. Чтобы отслеживать соответствие текущей ситуации плану (сколько понадобится тест-кейсов, сколько уже есть, сколько выполнено из запланированного на данном этапе количества и т. д.).
4. Если нам надо уточнить взаимопонимание между заказчиком, разработчиками и тестировщиками (тест-кейсы зачастую нагляднее показывают поведение приложения, чем это отражено в требованиях). Встречается на проектах, что заказчик смотрит, какие тест-кейсы пишутся командой тестирования. В частности, для приёмочного тестирования.
5. Чтобы хранить информацию для длительного использования и обмена опытом между сотрудниками и командами. А также для её (информации) дополнения, уточнения или переработки.
6. Чтобы проводить регрессионное тестирование, которое, как правило, проводится на основе тест-кейсов.
7. Чтобы повышать качество требований, поскольку во время написания тест-кейсов мы одновременно проверяем требования.
8. Чтобы быстро вводить в курс дела нового сотрудника, недавно подключившегося к проекту.

Тест-кейсы здесь главный источник информации: зная, сколько тест-кейсов написано и какие требования они покрывают, мы можем видеть, что сделано и сколько еще предстоит сделать в плане покрытия требований. Как правило, для этого используют Traceability Matrix, или матрицу прослеживаемости.

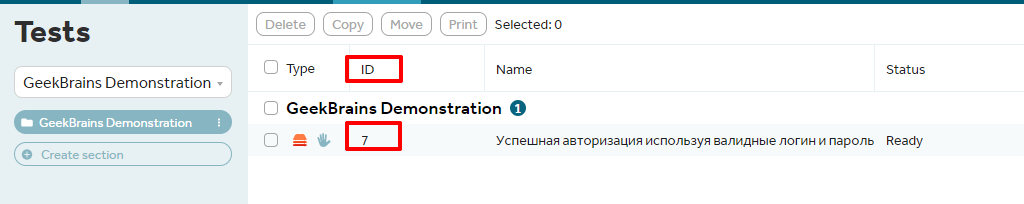
**Матрица прослеживаемости** — двумерная таблица, которая связывает между собой две сущности: требования и тест-кейсы. Таблица позволяет отслеживать ссылки одного объекта на другой, что позволяет определить достигнутый охват и оценить влияние предлагаемых изменений.

## Атрибуты (или поля) 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.

Рассмотрим основные атрибуты тест-кейса на примере программ TestIT и Testlink. Возможно, Testlink кто-то из вас уже установил на свою машину.

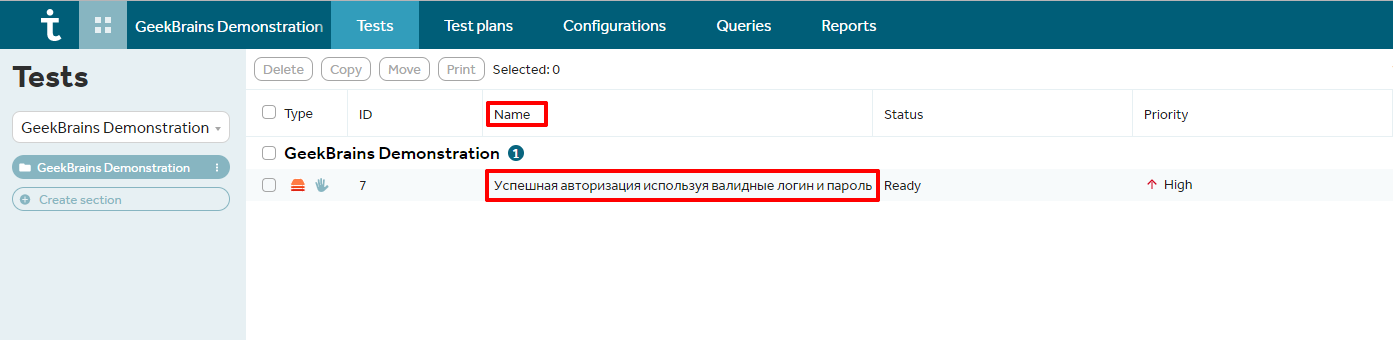
**Идентификатор (identifier, ID)** представляет собой уникальное значение, позволяющее однозначно отличить один тест-кейс от другого и используемое во всевозможных ссылках. В общем случае идентификатор тест-кейса может представлять собой просто уникальный номер, но может быть и куда сложнее: включать префиксы, суффиксы и иные осмысленные компоненты, позволяющие быстро определить цель тест-кейса и часть приложения (например: 20190621-1 или просто ID — 7).



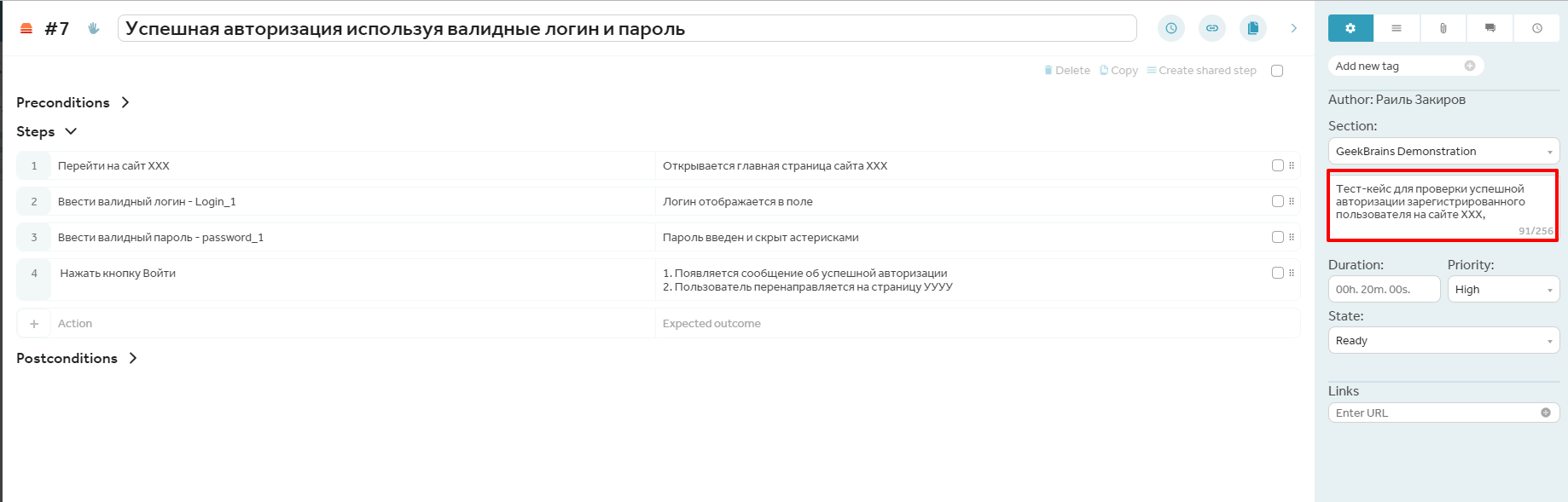
**Заглавие (Name) тест-кейса (title)** призвано упростить и ускорить понимание основной идеи (цели) тест-кейса без обращения к его остальным атрибутам. Именно это поле наиболее информативно при просмотре списка тест-кейсов. Поэтому очень важно его правильно называть. Представьте, перед вами две сотни тест-кейсов, и вам на лету надо сделать какую-то выборку. Заглавия здесь играют очень важную роль.

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

1. Информативность.
2. Относительная уникальность (чтобы не путать разные тест-кейсы).



**Краткое описание (Summary)** — часть тест-кейса, в которой в краткой форме описывается его основная суть. Иногда вместо Summary присутствует поле Description.

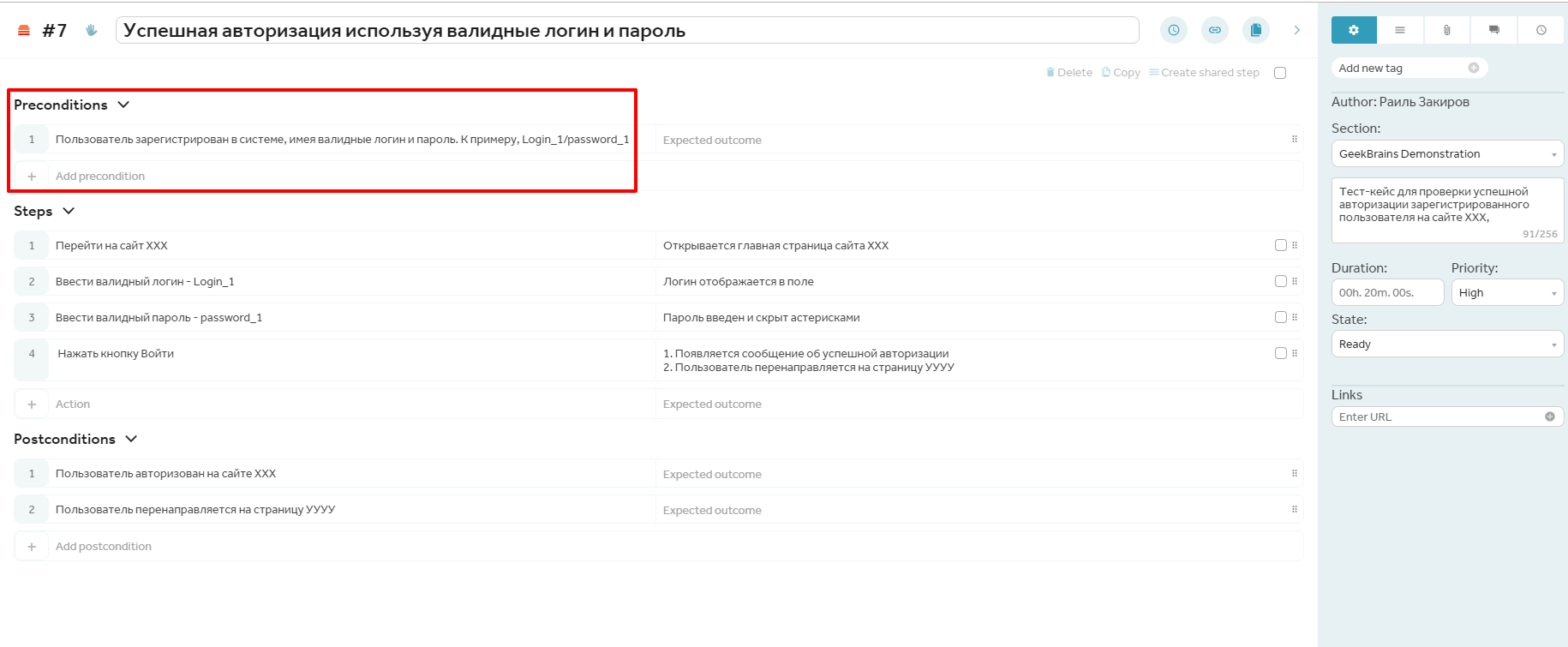


**Исходные данные (Preconditions)**, необходимые условия для начала выполнения тест-кейса (precondition, preparation, initial data, setup), позволяют описать то, что должно быть подготовлено до начала выполнения тест-кейса, например:

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

То, что описывается в этом поле, может готовиться без использования тестируемого приложения, и, таким образом, если здесь возникают проблемы, нельзя писать отчет о дефекте в приложении. Эта мысль очень и очень важна, потому поясним её простым жизненным примером.

К примеру, чтобы протестировать функциональность оплаты товара, он должен быть добавлен в корзину покупателя через базу данных.

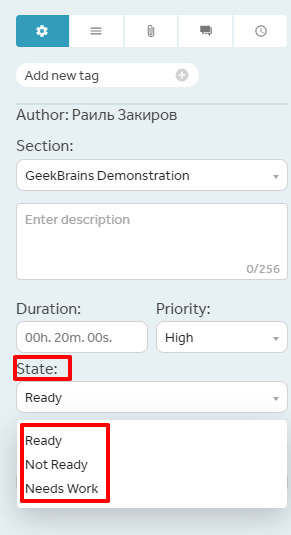


**Статус тест-кейса** показывает его текущее состояние. У тест-кейса может быть несколько статусов. В случае TestLink это:

* Черновик.
* Готов для проверки.
* Проверка в процессе.
* Переработка.
* Устаревший.
* Готов.

Это поле настраиваемое.

В TestIT всего 3 статуса — Ready, Not Ready, Needs Work.



**Приоритет (Priority или Importance)** 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»).

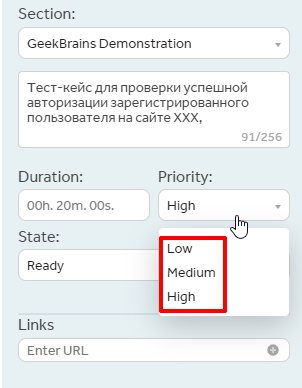
Количество градаций чаще всего лежит в диапазоне от трёх до пяти.

Приоритет тест-кейса может коррелировать с:

* важностью требования, пользовательского сценария или функции, с которыми связан тест-кейс;
* потенциальной важностью дефекта, на поиск которого направлен тест-кейс;
* степенью риска, связанного с проверяемым тест-кейсом требованием, сценарием или функцией.

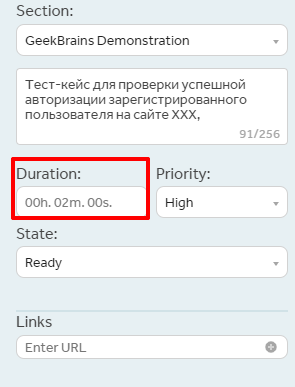
Основная задача атрибута Priority — упрощение распределения внимания и усилий команды. Как правило, более высокоприоритетные тест-кейсы получают больше внимания и выполняются в первую очередь, чем менее приоритетные.

Также упрощается планирование и принятие решения, чем можно пожертвовать в некоей ситуации, когда всё протестировать нет времени; обычно жертвуют низкоприоритетными проверками.



**Тип выполнения — ручной или автоматический**. Если тест-кейс выполняется вручную, то выбирается Manual. Если тест-кейс автоматизирован, выбирается Automated.

**Оценочное время выполнения** (Duration, Estimated execution duration) — поле, в котором указывается оценочное время выполнения/прохождения тест-кейса.



Также к тест-кейсам можно прикрепить **связанное с тест-кейсом требование** (requirement). Показывает основное требование, проверке которого посвящён тест-кейс.

Основное — потому, что один тест-кейс может затрагивать несколько требований. Это поле улучшает такое свойство тест-кейса, как прослеживаемость.

Частые вопросы, связанные с заполнением этого поля, таковы:

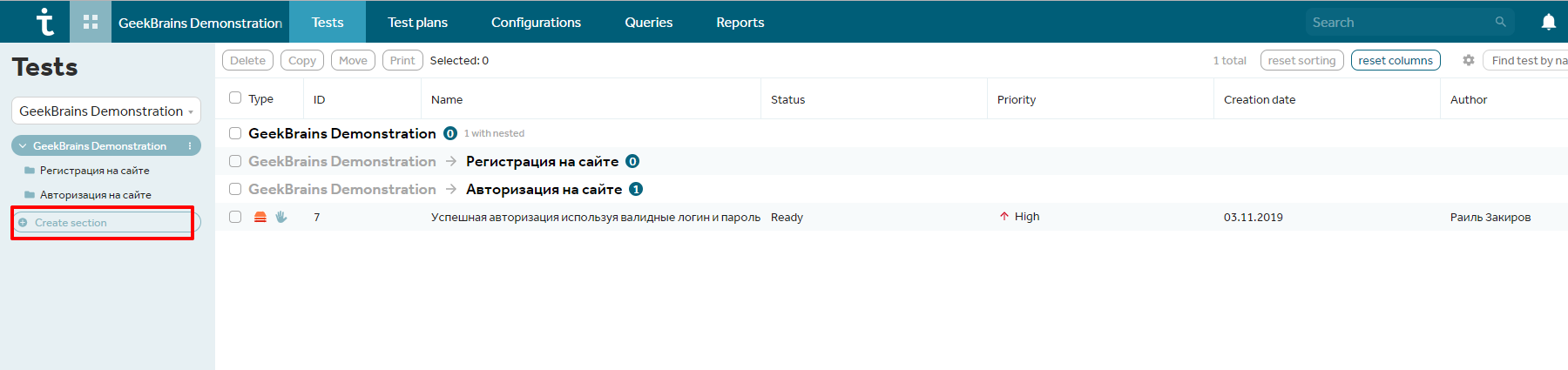
* **Можно ли его оставить пустым?** Да. Если тест-кейс временно разрабатывается вне прямой привязки к требованиям, значение этого поля определить сложно. Хоть такой вариант и не считается хорошим, он достаточно распространён.
* Можно ли в этом поле указывать несколько требований? Да, но чаще всего стараются 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 — следствие 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.

В случае, если у вас идёт разработка только одного определенного модуля, не связанного с другими, для цикла регрессионного тестирования можно отфильтровать тест-кейсы по требуемому модулю.

В TestLink реализация разбивки на модули и подмодули сделана таким образом, что надо создавать отдельные наборы тест-кейсов — test suite или test set.

В TestIT разделение на модули происходит путем добавления Section, как представлено ниже.

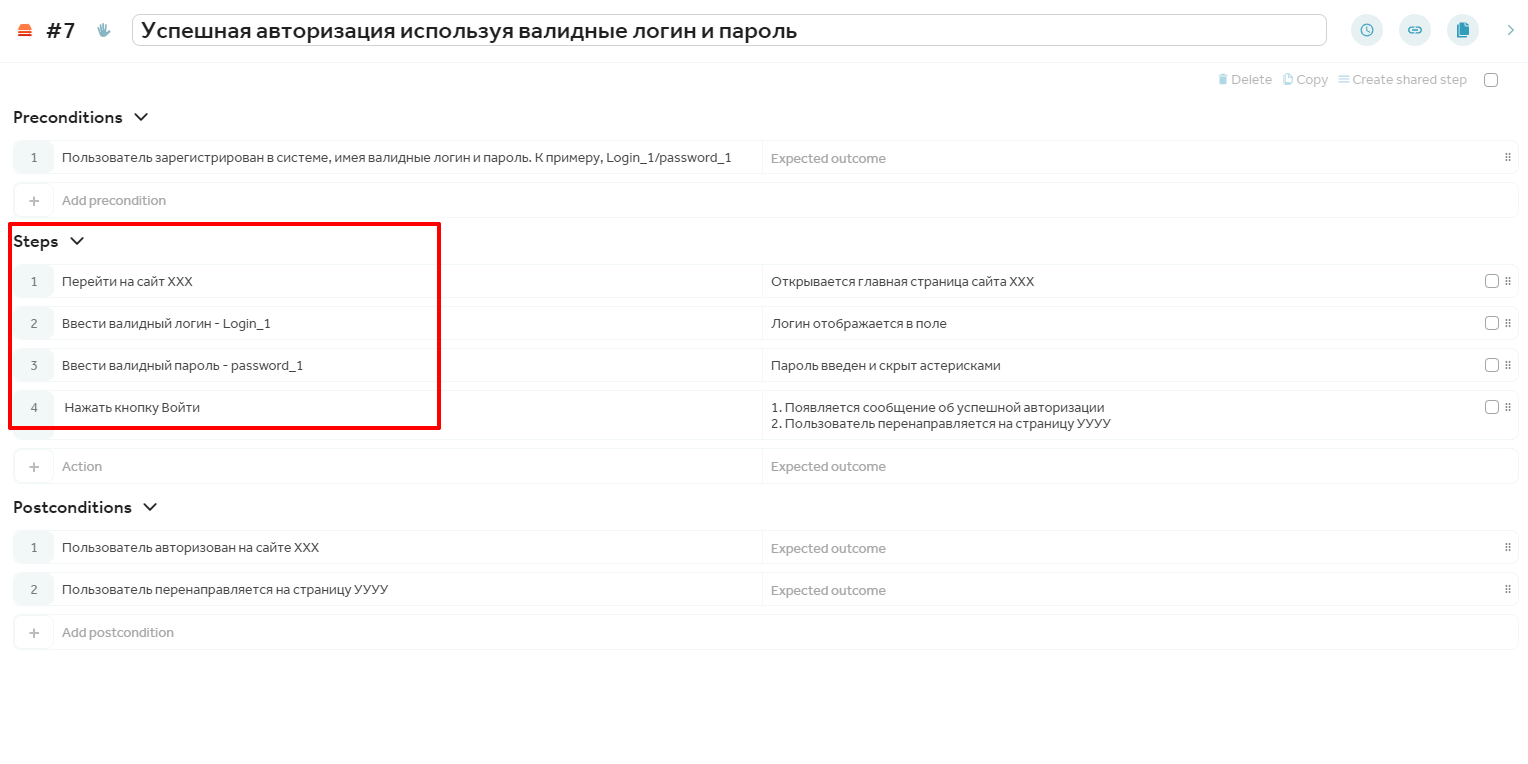


**Шаги тест-кейса (steps)** описывают последовательность действий, которые необходимо реализовать в процессе его выполнения.

Общие рекомендации по написанию шагов таковы:

1. Начинайте с понятного и очевидного места.
2. Избегайте писать очевидные\лишние начальные шаги (запуск приложения, очевидные операции с интерфейсом и т. п.).
3. Если в тест-кейсе всего один шаг, нумеруйте его (иначе возрастает вероятность в будущем случайно «приклеить» описание этого шага к новому тексту).
4. 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»).
5. Соотнесите степень детализации шагов и их параметров с целью тест-кейса, его сложностью, уровнем 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.
6. Ссылайтесь на предыдущие шаги и их диапазоны для сокращения объёма текста (например, «повторить шаги 3–5 со значением…»). В некоторых компаниях так делают, а где-то это считается дурным тоном. Решать вам.
7. Пишите шаги последовательно.
8. Пишите без условных конструкций вида «если… то…».

Очень серьёзная ошибка — ссылаться на шаги из других тест-кейсов и тест-кейсы целиком: если другие тест-кейсы будут изменены или удалены, ваш начнет ссылаться на неверные данные или несуществующие шаги. Если в процессе выполнения другие тест-кейсы или шаги приведут к ошибке, вы не сможете закончить ваш тест-кейс.

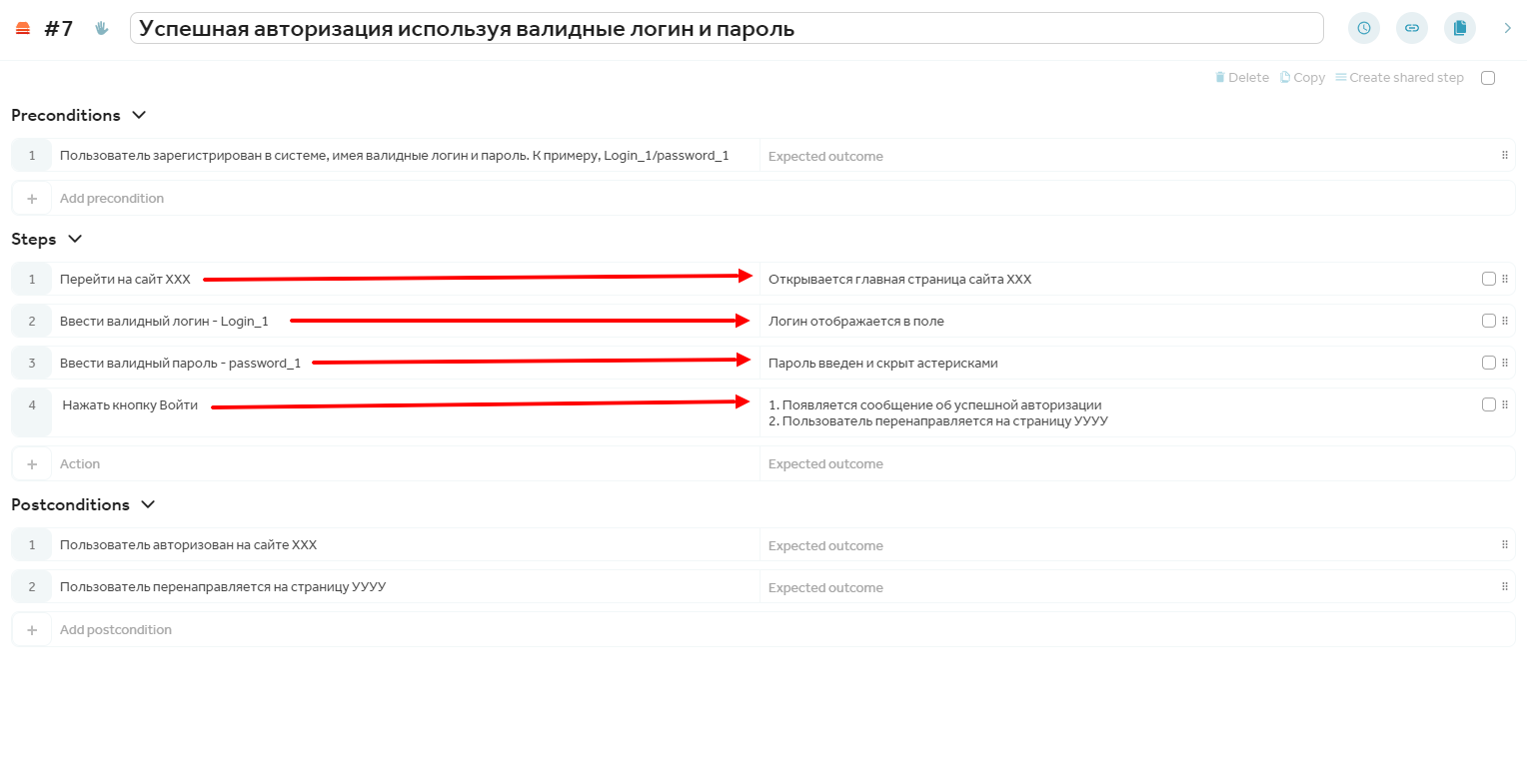


**Ожидаемые результаты (expected results)** по каждому шагу тест-кейса описывают реакцию приложения на действия, описанные в поле «шаги тест-кейса». Номер шага соответствует номеру результата.

По написанию ожидаемых результатов можно рекомендуется следующее:

1. Описывайте поведение системы так, чтобы исключить субъективное толкование (например, «приложение работает верно» — плохо, «появляется окно с надписью…» — хорошо). То есть точно указываете, что должно быть на экране, как изменилась кнопка или поле, что появилось в окне/браузере пользователя и так далее.
2. Пишите ожидаемый результат по всем шагам без исключения, если у вас есть хоть малейшие сомнения, 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).
3. Пишите кратко, но не в ущерб информативности.
4. Избегайте условных конструкций вида «если… то…».

Еще одна серьезная ошибка: в ожидаемых результатах ВСЕГДА описывается КОРРЕКТНАЯ работа приложения. Не должно быть ожидаемого результата в виде «приложение вызывает ошибку в операционной системе, и работа аварийно завершается с потерей всех пользовательских данных». 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» — не ошибка приложения, а совершенно нормальная и правильная работа. Ошибкой приложения (в этой же ситуации) было бы отсутствие такого сообщения, и/или повреждение, или потеря записываемых данных.



# Свойства качественных тест-кейсов

**Правильный технический язык, точность и единообразие формулировок.** Это свойство должно применяться и к требованиям, и к тест-кейсам, и к отчётам о дефектах — к любой тестовой документации, применяемой на проекте.

Основные идеи уже были описаны, а из самого общего и важного можно напомнить и добавить:

* писать лаконично, но понятно;
* использовать безличную форму глаголов (например, «открыть» вместо «откройте»);
* указывать точные названия/имена и технически верные названия элементов приложения;
* не объяснять базовые принципы работы с компьютером;
* везде называть одни и те же вещи одинаково, использовать одинаковое название/имана во всей тестовой документации;
* следовать принятому на проекте стандарту оформления и написания тест-кейсов.

**Баланс между специфичностью и общностью.** Тест-кейс считается тем более специфичным, чем более детально в нем расписаны 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.

**Баланс между простотой и сложностью.** Здесь надо руководствоваться общей логикой при написании тест-кейсов и стараться придерживаться идеи, 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;
* они упрощают начальную диагностику ошибки, т. к. сужают круг поиска.

Преимущества сложных тест-кейсов:

* пользователи, как правило, используют сложные сценарии, и потому сложные тесты более полноценно эмулируют работу пользователей;
* программисты редко проверяют такие сложные случаи (и они совершенно не обязаны это делать).

**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. Такое может встретиться, когда на проекте тест-кейсы пишут несколько специалистов. В таком случае необходимо, чтобы был один ревьюер (человек, который делает финальную проверку всех тест-кейсов), как правило, это руководитель команды тестирования или старший специалист по тестированию.

Если обнаруживается несколько тест-кейсов, дублирующих задачи друг друга, лучше всего или удалить все, кроме одного, самого показательного, или перед удалением остальных на их основе доработать 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 очевидно, что произошла ошибка.

**Прослеживаемость.** Из содержащейся в качественном тест-кейсе информации, должно быть понятно, **какую** часть приложения или **какие** функции и требования он проверяет.

**Возможность повторного использования**. Это свойство редко выполняется для низкоуровневых тест-кейсов, но при создании высокоуровневых 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 быть любой тест-кейс с высокой специфичностью.

**Повторяемость.** Тест-кейс должен быть сформулирован таким образом, чтобы при многократном повторении он показывал одинаковые результаты. Это связано с недвусмысленностью формулировки шагов – специалист, выполняющий тест-кейс, должен точно понимать каждый раз, что необходимо выполнить и в рамках каких условий. Также шаги не должны приводить к последствиям, которые могут привести к затруднению дальнейшего тестирования (например, удалить базу данных).

**Ожидаемый результат для каждого шага тест-кейса.** Бывает, встречаются тест-кейсы, в которых ожидаемый результат указывается только после последнего шага. Эта методика написания тест-кейсов снижает их эффективность, поскольку отсутствие результатов для первых шагов не дает чёткого понимания, как должно вести себя приложение или программа при их прохождении.

# Типичные ошибки

## Ошибки оформления и формулировок

1. **Отсутствие заглавия тест-кейса или плохо написанное заглавие**. Это поле — первое, на что смотрит инженер по тестированию открывая набор тест-кейсов или сам тест-кейс. Поэтому это поле обязательно заполнять.
2. **Отсутствие нумерации шагов и/или ожидаемых результатов**. Усложняет прохождение тест-кейсов, поскольку очередность шагов может быть перепутана. Из-за этого становится очень легко перепутать, что к чему относится. Даже выполнение и доработка таких тест-кейсов занимает много времени.
3. **Ссылка на множество требований.** Как правило, тест-кейс пишется для проверки какой-то одной функциональности, но иногда высокоуровневый тест-кейс действительно затрагивает несколько требований. Как правило, в таком случае рекомендуется писать ссылку на максимум 2–3 самых ключевых.
4. **Использование личной формы глаголов.** Если вы пишете требования на русском, то пишите «нажать» вместо «нажмите», «ввести» вместо «введите», «перейти» вместо «перейдите» и т. д.
5. **Использование прошедшего или будущего времени в ожидаемых результатах.** Это не очень серьёзная ошибка, но всё равно «введенное значение отображается в поле» читается лучше, чем «введенное значение отобразилось в поле» или «введенное значение отобразится в поле».
6. **Постоянное использование слов «проверить» (и ему подобных) в чек-листах**. Может получиться, что каждый шаг будет начинаться с «проверить…», «проверить…», «проверить…».
7. **Пунктуационные, орфографические, синтаксические и им подобные ошибки**. Здесь вам очень помогут спеллчекеры, как для русского языка, так и для английского. Встречаются такие ситуации, когда тест-кейсы проверяют со стороны заказчика инженеры по качеству. Если вы будете проверять чьи-то тест-кейсы, а там много грамматических ошибок, как вы отреагируете?

## Логические ошибки

1. **Ссылка на другие тест-кейсы или шаги других тест-кейсов.** За исключением случаев написания строго оговоренного и явно обозначенного набора последовательных тест-кейсов, это запрещено делать. Если ваш тест-кейс будет ссылаться на отсутствующие шаги, пройти его практически невозможно.
2. **Детализация, не соответствующая уровню функционального тестирования.** Например, не нужно на уровне дымового тестирования проверять работоспособность каждой отдельной кнопки или прописывать некий крайне сложный, нетривиальный и редкий сценарий.
3. **Расплывчатые двусмысленные описания действий и ожидаемых результатов.** Стилистика написания тест-кейсов должна быть в таком виде, чтобы любой специалист проекта мог успешно пройти по всем шагам этого тест-кейса. Как правило, не всегда те, кто пишет тест-кейсы, их и проходит.
4. «**Выдумывание» особенностей поведения приложения.** Бывает, что в требованиях есть «белые пятна» в описании. В таком случае, хорошая практика — уточнить формулировку у аналитика, который составлял это требование. «Додумывание» как должно работать приложение в таких случаях, — плохая практика.
5. **Отсутствие описания приготовления к выполнению тест-кейса.** Некоторые специфические тест-кейсы требуют указания и чёткого описания, какие предварительные условия должны быть выполнены перед началом прохождения. Таким примером могут быть:

* определённый статус заявки в системе,
* определённый баланс на счету клиента (отрицательный или очень большой),
* отсутствие какой-либо информации в базе данных.

Если такие условия влияют на прохождение, они обязательно должны быть указаны.

1. **Полное дублирование (копирование) одного и того же тест-кейса на уровнях дымового тестирования, тестирования критического пути, расширенного тестирования.** Многие идеи естественным образом развиваются от уровня к уровню или дополняются, но они должны именно развиваться, а не дублироваться.

# Наборы тест-кейсов

В соответствии с ISTQB, набор тест-кейсов (Test suite) — набор тест-кейсов, которые должны быть выполнены в определенном тестовом цикле.

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, 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.

Надо помнить, что в первую очередь при создании тестов или тест-кейсов мы создаём позитивные тесты. Если позитивные сценарии не срабатывают, часто нет смысла тестировать негативные сценарии.

Рассмотрим такой случай. У нас есть тестовый набор для проверки авторизации пользователя на каком-либо сайте или ресурсе.

Возьмем к примеру, следующие несколько тест-кейсов:

1. Успешная авторизация на сайте при использовании валидных логина и пароля.
2. Не успешная авторизация при использовании валидного логина и невалидного пароля.
3. Не успешная авторизация при использовании невалидного логина (когда у нас опечатка в логине на одну букву, то есть аккаунта не существует) и валидного пароля.
4. Функция CUT неактивна.
5. Функция COPY неактивна.

Предварительное условие для этих тест-кейсов — в системе существуют два логина и два соответствующих им пароля. К примеру Логин\_1/Пароль\_1, Логин\_2/Пароль\_2.

Смысл такой, что если мы не прошли первый тест Happy Path, остальные тест-кейсы нет смысла проверять, весь тестовый набор заблокирован. Пользователь не может залогиниться/авторизоваться на сайте, используя свои настоящие учётные данные.

## Пример тест-кейса для успешной авторизации

**Title**. Успешная авторизация с использованием валидных логина и пароля

**Preconditions**. Пользователь зарегистрирован в системе, имея валидные логин и пароль. К примеру, Login\_1/password\_1.

**Steps:**

1. Перейти на сайт ХХХ — страница авторизации.

**ОР** (ожидаемый результат): открывается главная страница сайта ХХХ.

1. Ввести валидный логин — Логин\_1.

**ОР**: Логин отображается в поле.

1. Ввести валидный пароль — Пароль\_1.

**ОР** — Пароль введен и скрыт астерисками.

4. Нажать кнопку «Войти».

**ОР:**

* Появляется сообщение об успешной авторизации.
* Пользователь перенаправляется на страницу УУУУ.

В примере пропущены другие обязательные поля, но их обязательно надо вносить в систему хранения и управления тест-кейсами, или любое другое место, где хранятся тест-кейсы на проекте.

**Тест-комплекты могут формироваться по следующим признакам:**

● **По типу тестирования**. Например, тест-комплект для дымового тестирования или тестирования критического пути, куда входят наиболее важные для приложения функции.

● **По модулям ПО**. Например, будет создан тест-комплект «Регистрация» в котором находятся все тест-кейсы на регистрацию пользователя в системе, как негативные, так и позитивные. Тест-комплекты могут быть вложены друг в друга для удобства выполнения или разделения по модулям ПО. Например, в тест-комплекте «Управление пользователями» могут находиться тест-комплекты: «Регистрация», «Авторизация», «Назначение прав» и т. д.

# Разница между тест-кейсом и дефектом

На первый взгляд может показаться, что это одно и то же. Но на самом деле это разные виды тестовой документации, у которых первичные цели абсолютно разные.

В первую очередь основная цель отчёта о дефекте — предоставление информации о существующей проблеме/ошибке.

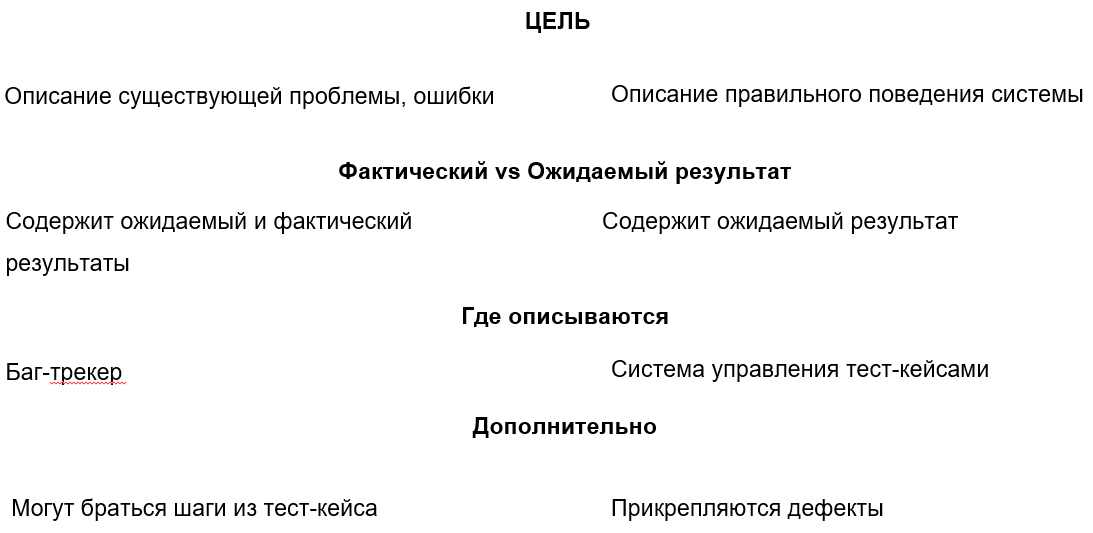
Цель тест-кейса — последовательное описание правильной работы приложения.

Тест-кейс содержит только ожидаемый результат на этапе его написания (или когда он уже написан).

Отчёт о дефекте содержит и ожидаемый (ОР), и фактический результат (ФР).

Также отчёты о дефектах и тест-кейсы, как правило, хранятся в разных системах. Но бывают и исключения. К примеру, в HP ALM (Application Lifecycle Management) позволяет хранить и тест-кейсы и создавать отчёты о дефектах.

Шаги из тест-кейса могут браться для составления отчёта о дефекте.



# Принципы тестирования

Рассмотрим основные принципы тестирования. Часто их называют аксиомами тестирования. Мы будем рассматривать их как принципы тестирования, основываясь на книге «FOUNDATIONS OF SOFTWARE TESTING. ISTQB CERTIFICATION» авторов Dorothy Graham, Erik van Veenendaal, Isabel Evans, Rex Black.

**Тестирование зависит от контекста (Testing is context dependent)**. Тестирование проводится по-разному в разных контекстах. Например, критически важное для безопасности программное обеспечение тестируется не так, как сайт электронной коммерции или как того требует мобильное приложение.

**Исчерпывающее тестирование невозможно (Exhaustive testing is impossible)**. Тестирование всего (всех комбинаций входных данных и предварительных условий) невозможно, за исключением тривиальных случаев. Вместо исчерпывающего тестирования мы используем риски и приоритеты, чтобы сосредоточить усилия на тестировании.

**Кластеризация дефектов (Defect clustering)**. Небольшое количество модулей содержит большинство дефектов, обнаруженных во время тестирования перед выпуском продукта, или показывает большинство операционных сбоев.

**Тестирование показывает наличие дефектов (Testing shows presence of defects)**. Тестирование может показать наличие дефектов, но не может доказать, что дефектов нет. Тестирование снижает вероятность того, что в программном обеспечении останутся не обнаруженные дефекты, но даже если дефектов не нашли, это не доказывает, что всё правильно.

**Заблуждение в отсутствие ошибок (Absence of errors fallacy)**. Поиск и устранение дефектов не помогает, если построенная система непригодна для использования и не отвечает потребностям и ожиданиям пользователей. Вероятно, что программное обеспечение, которое на 99% не содержит ошибок, по-прежнему невозможно использовать, если оно не решает проблему юзера. Это может быть, если система тщательно проверена на неправильные требования. Тестирование программного обеспечения — не просто поиск дефектов, но и проверка того, что программное обеспечение отвечает потребностям бизнеса или ожиданиям пользователя. Отсутствие ошибки — ошибка, то есть обнаружение и устранение дефектов не помогает, если сборка системы непригодна для работы и не отвечает потребностям и требованиям пользователя.

**Раннее тестирование (Early testing)**. Тестирование следует начинать как можно раньше в жизненном цикле разработке ПО или системы и оно должно быть сосредоточено на определённых целях.

**Парадокс пестицида (Pesticide paradox)**. Если одни и те же тесты повторять снова и снова, то в конечном итоге один и тот же набор тест-кейсов не найдёт новых ошибок. Чтобы преодолеть этот парадокс, необходимо регулярно пересматривать тест-кейсы, а также добавлять новые и разные тест-кейсы для выполнения проверок в различных частях и модулях приложения или системы, чтобы найти больше дефектов.

# Подходы к тестированию

В тестирование, помимо видов и типов тестирования (что одно и то же), выделяют подходы. Мы рассмотрим подходы к тестированию по степени формализации.

**Тестирование на основе тест-кейсов (scripted testing, test case based testing)**. Формализованный подход, в котором тестирование производится на основе заранее подготовленных тест-кейсов, их наборов и иной документации. Это самый распространённый способ тестирования, который также позволяет достичь максимальной полноты исследования приложения за счёт строгой систематизации процесса, удобства применения метрик и широкого набора выработанных за десятилетия и проверенных на практике рекомендаций.

**Исследовательское тестирование (exploratory testing)**. Частично формализованный подход, в рамках которого тестировщик выполняет работу с приложением по выбранному сценарию, который, в свою очередь, дорабатывается в процессе выполнения, чтобы более полно исследовать приложения. Ключевой фактор успеха при выполнении исследовательского тестирования — работа по сценарию, а не выполнение разрозненных бездумных операций.

**Свободное (интуитивное) тестирование (ad hoc testing)**. Полностью неформализованный подход, в котором не предполагается использования ни тест-кейсов, ни чек-листов, ни сценариев — тестировщик полностью опирается на свой профессионализм и интуицию (experience-based testing) для спонтанного выполнения с приложением действий, которые, как он считает, могут обнаружить ошибку. Этот вид тестирования используется редко и исключительно как дополнение к полностью или частично формализованному тестированию в случаях, когда для исследования некоторого аспекта поведения приложения (пока?) нет тест-кейсов.

# Глоссарий

**Тестирование на основе чек-листа** (checklist-based testing) — методика, основанная на опыте, при которой опытный тестировщик использует список высокоуровневых пунктов, которые необходимо записать, проверить или запомнить, или набор правил или критериев, в соответствии с которыми должен быть проверен продукт.

**Чек-лист** — набор/лист проверок, который описан верхнеуровнево, без детализации.

**Тест-кейс или тестовый случай (test case)** —набор предварительных условий, входных данных, действий (где применимо), ожидаемых результатов и пост условий, разработанных на тестовых условиях.

**Высокоуровневый тест-кейс** (**High-level test case**) — тест-кейс без конкретных входных данных и ожидаемых результатов.

**Низкоуровневый тест-кейс (Low-level test case**) — тест-кейс с конкретными входными данными и ожидаемыми результатами.

**Матрица прослеживаемости (traceability matrix)** — двумерная таблица, которая связывает между собой две сущности, требования и тест-кейсы. Таблица позволяет отслеживать ссылки одного объекта на другой, что помогает определить достигнутый охват и оценить влияние предлагаемых изменений.

**Набор тест кейсов (Test suite)** — набор тест-кейсов, которые должны быть выполнены в определенном тестовом цикле.

**Тестирование на основе тест-кейсов (scripted testing, test case based testing)** — формализованный подход, в котором тестирование производится на основе заранее подготовленных тест-кейсов, наборов тест-кейсов и иной документации.

**Исследовательское тестирование (exploratory testing)** — частично формализованный подход, в рамках которого тестировщик выполняет работу с приложением по выбранному сценарию, который, в свою очередь, дорабатывается в процессе выполнения с целью более полного исследования приложения.

**Свободное (интуитивное) тестирование (ad hoc testing)** — полностью неформализованный подход, в котором не предполагается использования ни тест-кейсов, ни чек-листов, ни сценариев — тестировщик полностью опирается на свой профессионализм и интуицию (experience-based testing) для спонтанного выполнения с приложением действий, которые, как он считает, помогут обнаружить ошибку.

# Практическое задание

**Используя приложенную таблицу, выполните следующие задания:**

1. Прочитать книгу Святослава Куликова (Software Testing — Base Course (Svyatoslav Kulikov)), страницы 112-163.
2. Ответить на вопросы в приложенной форме.
3. Составить тест-кейсы для формы авторизации, используя приведённые в таблице требования.
4. \* Пройдите Challenge 2 на сайте <http://testingchallenges.thetestingmap.org/challenge2.php>.
5. \* Пройдите Challenge 6 на сайте <http://testingchallenges.thetestingmap.org/challenge6.php>.

## Требования к выполненной работе

1. Домашнее задание должно быть выполнено в приложенной форме.
2. Расширение файла должно быть формата Excel (XLS, XLSX).
3. Название файла не менять, и вместо ФИО укажите свои фамилию и имя.

# Дополнительные материалы

1. [Checklists Base](https://checkvist.com/checklists/476089).
2. [What is Adhoc Testing? Types with Example](https://www.guru99.com/adhoc-testing.html).
3. [What is Exploratory Testing? Techniques with Examples](https://www.guru99.com/exploratory-testing.html).
4. [Глоссарий системы TESTLINK](https://training.qatestlab.com/blog/course-materials/testlink-glossary/).
5. [Cheat-sheet](http://tmguru.ru/baza-znanij/upravlenie-testami/cheat-sheet).
6. [TОП-13 ошибок тестировщиков. Часть I. Требования, тест-кейсы](http://software-testing.ru/library/5-testing/66-top-13----i----).
7. [7 принципов тестирования. Часть 3](https://www.luxoft-training.ru/blog/veni_vidi_vickie/541.html).
8. [Usability Checklist](https://teamsuccess.io/UX).

# Используемые источники

Для подготовки методического пособия мы использовали эти ресурсы:

1. Software Testing — Base Course (Svyatoslav Kulikov).
2. Lee Copeland — A Practitioner's Guide to Software Test Design (2004).
3. Dorothy Graham, Erik van Veenendaal, Isabel Evans, Rex Black — Foundations of Software Testing. ISTQB Certification.
4. [7 Software Testing Principles: Learn with Examples](https://www.guru99.com/software-testing-seven-principles.html).