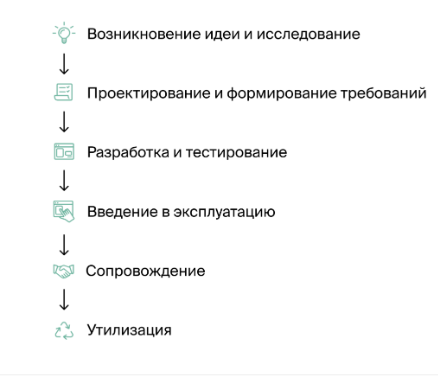
**Жизненный цикл программного обеспечения**

Всего приложение проходит через шесть этапов:



**Возникновение идеи и исследование**

Приложения создают, чтобы решать определённые задачи пользователей. Например, в Zoom и Skype можно удалённо проводить рабочие совещания, а в Яндекс.Навигаторе — прокладывать маршруты с учётом пробок и находить парковку.

Чтобы приложение оказалось полезным, команда изучает потребности и привычки пользователей. Например, менеджер продукта в Яндекс.Движе хочет создать приложение для аренды самокатов в Москве. Но сначала нужно узнать спрос — исследовать рынок и аудиторию.

**Проектирование и формирование требований**

Если исследование подтвердило, что аренда самокатов будет пользоваться спросом, будущее приложение детально описывают:

* Менеджер формулирует, как приложение должно работать.
* Дизайнеры предлагают, как будет выглядеть интерфейс.
* Разработчики продумывают технические детали.

На этом этапе появляются требования и макеты.

**Проектирование и формирование требований**

Если исследование подтвердило, что аренда самокатов будет пользоваться спросом, будущее приложение детально описывают:

* Менеджер формулирует, как приложение должно работать.
* Дизайнеры предлагают, как будет выглядеть интерфейс.
* Разработчики продумывают технические детали.

На этом этапе появляются требования и макеты.

**Требования** — описание того, как приложение должно работать и какой функциональностью обладать.

Визуальное представление того, как должен выглядеть интерфейс — та часть приложения, которую видит пользователь.

**Разработка и тестирование**

Когда требования и макеты готовы, команда распределяет задачи и приступает к работе.

Пока разработчики пишут код, тестировщики составляют тестовую документацию: чек-листы и тест-кейсы. А когда приложение готово — проверяют его и оформляют баг-репорты.

Например, тестировщик заметил, что в Яндекс.Самокате не работает кнопка «Заказать». Это баг — нужно сообщить команде.

**Введение в эксплуатацию**

Когда тестировщики убедились, что всё работает без ошибок, приложение вводят в эксплуатацию, или выпускают в **релиз**. Теперь его увидят пользователи: смогут открыть Яндекс.Самокат и сделать заказ.

**Сопровождение**

После релиза работа не заканчивается: команда разработки исправляет баги, поддерживает и улучшает приложение.

Представь: пользователь написал, что не может заказать самокат на станцию метро «Проспект Вернадского». Команда поддержки сообщает разработчикам, а они исправляют баг и обновляют приложение.

**Утилизация**

Со временем компания может перестать обновлять приложение. Например, сопровождение стало очень дорогим, или приложение больше невозможно поддерживать из-за технических ограничений.

Тогда его могут утилизировать — удалить с сервера или из магазина приложений.

**Жизненный цикл задачи**

Разработки и тестирования тесно переплетены: разработчик реализует новую функциональность → тестировщик её проверяет и составляет баг-репорты → разработчик исправляет баги.

Поэтому разработка и тестирование происходят в рамках одной задачи. Например, менеджер Яндекс.Самоката поставил задачу — сделать форму предзаказа. Когда разработчик дописал код, задача переходит к тестировщику — ему предстоит проверить новую форму.

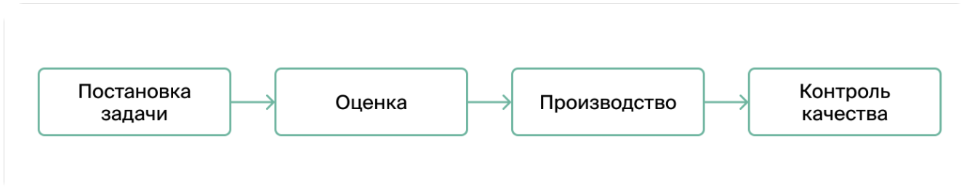
Сделать форму предзаказа самоката.

В ней пользователь вводит следующие данные:

|  |  |
| --- | --- |
| ПОЛЯ | ТРЕБОВАНИЯ |
| Имя | Только буквы русского и английского алфавитов. Длина не менее 2 и не более 25 символов. Если пользователь не соблюдает любое из требований, рамка поля подсвечивается красным, а под ней появляется текст ошибки: «Некорректный ввод». |
| Фамилия | Только буквы русского и английского алфавитов. Длина не менее 2 и не более 25 символов. Если пользователь не соблюдает любое из требований, рамка поля подсвечивается красным, а под ней появляется текст ошибки: «Некорректный ввод». |
| Отчество | Только буквы русского и английского алфавитов. Длина не менее 2 и не более 25 символов. Если |

Задача на разработку и тестирование проходит через четыре этапа:

Задача на разработку и тестирование проходит через четыре этапа:

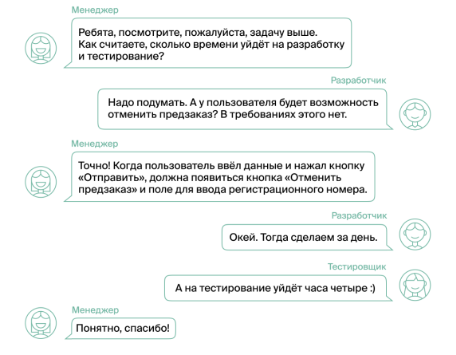


**Постановка задачи**

Сначала менеджер или аналитик формулирует требования к итоговому результату. В примере выше это форма предзаказа. В требованиях указано, какие данные должен ввести пользователь, как выглядит и в какой момент активируется кнопка «Заказать». По этим требованиям в Яндекс.Самокате будут разрабатывать и тестировать форму предзаказа.

**Оценка**

На этапе оценки команда бегло изучает требования, уточняет серые зоны и планирует, сколько времени нужно на задачу. Например:



**Производство**

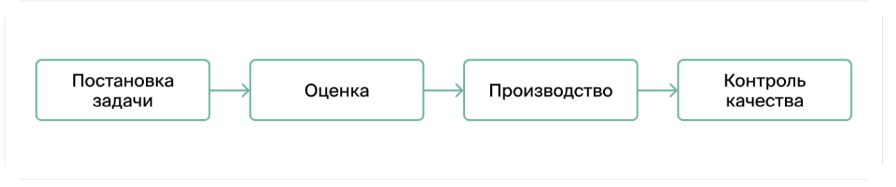
На этапе производства требования и макеты детально анализируют. Потом разработчики пишут по ним код, а тестировщики проектируют проверки.

**Контроль качества**

Когда новая функциональность готова, тестировщики проверяют её по чек-листам и тест-кейсам, которые составили на предыдущем этапе. А разработчики исправляют ошибки, которые обнаружили тестировщики.

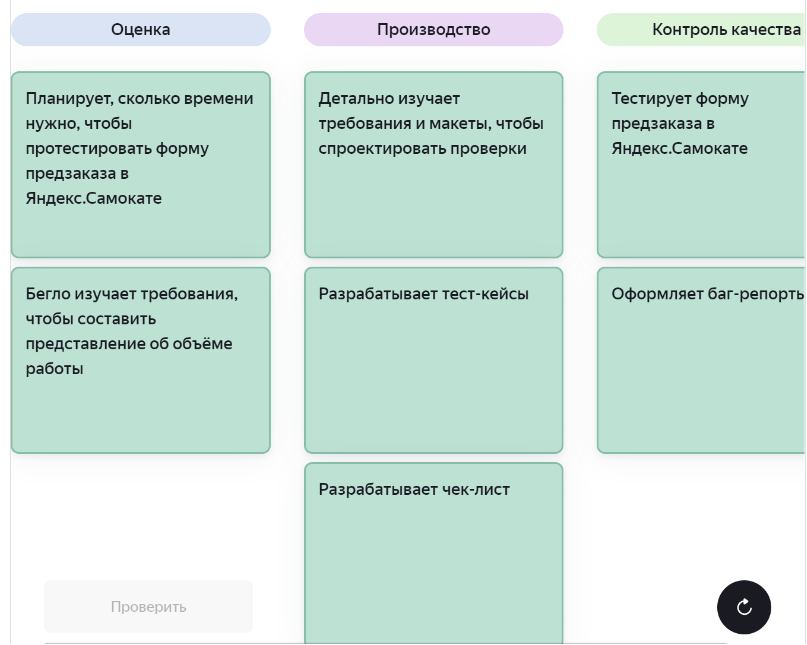
**Как тестировщик работает над задачей**

Задача на разработку и тестирование проходит через четыре этапа:



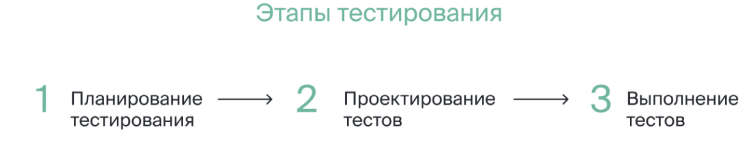
На каждом этапе разработчик и тестировщик выполняют разные действия.

Последовательность действий тестировщика.



**Этапы тестирования**

Как выглядят этапы оценки, производства и контроля качества с точки зрения тестировщика:



**Планирование тестирования**

На этом этапе нужно бегло изучить требования и оценить, сколько времени займёт тестирование.

**Проектирование тестов**

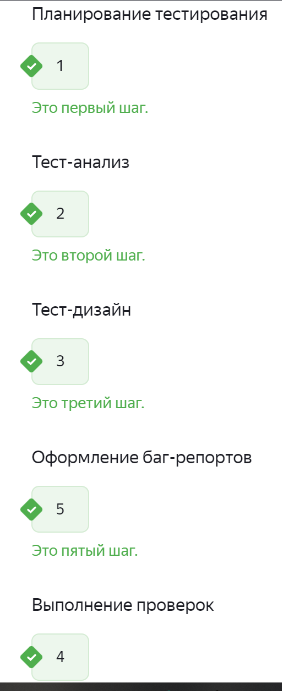
Этап проектирования состоит из двух шагов:

1. Сначала определи, что предстоит проверить: детально анализируются требования и макеты, задаются уточняющие вопросы разработчику, дизайнеру и менеджеру. Это называется **тест-анализ**.
2. Затем проектируются сами проверки: составляются чек-лист и тест-кейсы. Проектирование проверок называется **тест-дизайн**.

**Выполнение тестов**

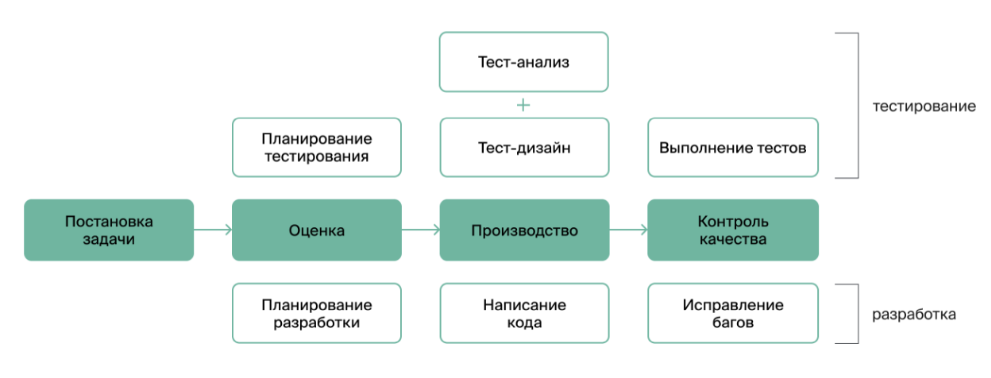
Когда список проверок готов, переходят к тестированию. Если обнаружется баг — оформляется баг-репорт.

**ПОСЛЕДОВАТЕЛЬНОСТЬ ДЕЙСТВИЙ**



Разработчик и тестировщик трудятся в рамках одной задачи, но выполняют разные действия. Тестировщик:

* планирует, сколько времени займёт тестирование;
* проектирует тесты: проводит тест-анализ и тест-дизайн;
* выполняет тесты.



**Тест-анализ** (test analysis) — изучение требований и макетов. Необходимо ответить на вопрос: что именно предстоит тестировать? Так составляется список объектов тестирования.

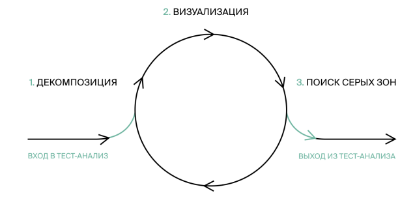
**Объекты тестирования** (test objects) — части приложения, которые нужно проверить.

Например, компания сделала почтовый клиент — приложение, в котором можно обмениваться электронными письмами. В последнем обновлении разработчики добавили фичу: теперь приложение определяет спам и сохраняет его в отдельной папке.

Тестировщик будет проверять не всё приложение целиком, а определённую функциональность — работу со спамом.

Но сначала нужно получить требования: с этого начинается вход в тест-анализ.

Требования можно найти в задаче. Дальше предстоит их декомпозировать и визуализировать, а затем найти серые зоны.



**Декомпозиция**

**Декомпозиция** — разделение целого на части. На этом этапе предстоит разбить крупные объекты тестирования на более мелкие. Так с ними удобнее работать.

Например, функциональность работы со спамом можно разбить на два объекта: определение спама и сохранение в отдельной папке.

**Визуализация**

Когда декомпозируются требования, необходимо их визуализировать.

**Визуализация** — создание наглядной схемы всех объектов тестирования. Так проще воспринимать и структурировать информацию.



**Поиск серых зон**

Когда декомпозируются и визуализируются требования, можно увидеть в них несостыковки, противоречия и пропуски — **серые зоны**. В этом случае необходимо обращаться к менеджеру, он поможет разобраться.

Иногда приходится искать серые зоны повторно. Если удалось заметить неявные требования, необходимо до уточнить их и дополнить схему. Но оказалось, что в ней всё ещё есть слепые зоны. Их тоже нужно уточнить, декомпозировать и визуализировать: так получается полное представление о том, что предстоит тестировать.

Тест-анализ заканчивается, когда удалось декомпозировать и визуализировать все требования, а также исключить серые зоны.

**Этапы тест-анализа:**

* «Анализ требований»;
* «Декомпозиция требований»;
* «Визуализация требований: диаграмма связей»;
* «Визуализация требований: блок-схема»;
* «Серые зоны»;
* «Поиск требований».

**Анализ требований**

**Требования** это описание того, как приложение должно работать и какой функциональностью обладать.

**GUI (Graphical User Interface — «графический пользовательский интерфейс»)** — это привычный для пользователей интерфейс. Информация выводится на компьютере в виде графических элементов: меню, кнопок, форм ввода данных, иконок. Через элементы на экране пользователь отправляет команды компьютеру.

**Требования к функциональности**

В блоке «Требования к функциональности» описано, как приложение должно работать. По этим требованиям и макетам разработчик пишет код, а тестировщик проектирует проверки.

Необходимо изучить требования и выделить в них несколько объектов тестирования.

Объекты тестирования — это функциональность, которая описана в пунктах требований.

Например: «Промо-страница включает нижний блок со ссылками, которые ведут на другие страницы: язык, обратная связь, пользовательское соглашение».

Тестировщику нужно будет проверить, что:

* в нижнем блоке промо-страницы есть ссылки,
* эти ссылки ведут на соответствующие страницы.

**Ограничения для полей ввода в форме регистрации**

Этот блок требований описывает, какой формат данных принимают поля ввода: «Имя», «Фамилия», «Отчество», «Email» и «Номер телефона».

Например: ограничения для поля «Имя»:

* только буквы русского и английского алфавитов,
* длина не менее двух и не более 25 символов.

Нужно протестировать, что поле принимает только такие данные. Например, имя «Мариса» ввести можно, а «М» — нет.

Когда определятся все крупные объекты тестирования, можно приступать к их декомпозиции.

**Декомпозиция требований**

Декомпозиция требований — это подход тест-анализа, при котором тестировщик разбивает крупные объекты тестирования на более мелкие. Так проще проектировать проверки.

**Правило первое**

Обычно одно требование можно разложить на несколько частей.

Например, удалось изучить требования к интернет магазину и выделить объект тестирования — форму регистрации. Форма состоит из нескольких полей ввода: «Имя», «Фамилия», «Отчество», «Email» и «Номер телефона».

**Правило второе**

Требования декомпозируют до атомарного ( Атомарный — это такой, который нельзя поделить на более мелкие части) уровня: так, чтобы поделить их ещё раз было уже нельзя.

Например, поле «Имя» на форме предзаказа можно декомпозировать так:

русские буквы,

английские буквы.

Это и есть атомарный уровень: разбить требование ещё раз не получится.

**Правило третье**

Требования декомпозируют в рамках существующего описания.

Например, если в требованиях к полю «Имя» не написано про заглавные и прописные буквы, то достаточно проверить любые буквы русского и английского алфавитов. Декомпозировать их до заглавных и прописных не нужно.

**Декомпозиция - пример**

Декомпозировать блок с формой регистрации: изучить требования ещё раз и выделить более мелкие объекты тестирования.

Должно получиться так:

* поле «Имя»,
* поле «Фамилия»,
* поле «Отчество»,
* поле «Email»,
* поле «Номер телефона»,
* кнопка «Заказать»,
* информация о количестве подписавшихся пользователей.

Требования к полям «Имя», «Фамилия» и «Отчество» можно разбить так:

* буквы русского алфавита,
* буквы английского алфавита.

Требования к полю «Email»:

* буквы английского алфавита,
* цифры,
* тире,
* точка,
* знак @.

Требования к полю «Номер телефона»:

* цифры,
* знак +,
* «+7» в начале.

**Визуализация требований: диаграмма связей**

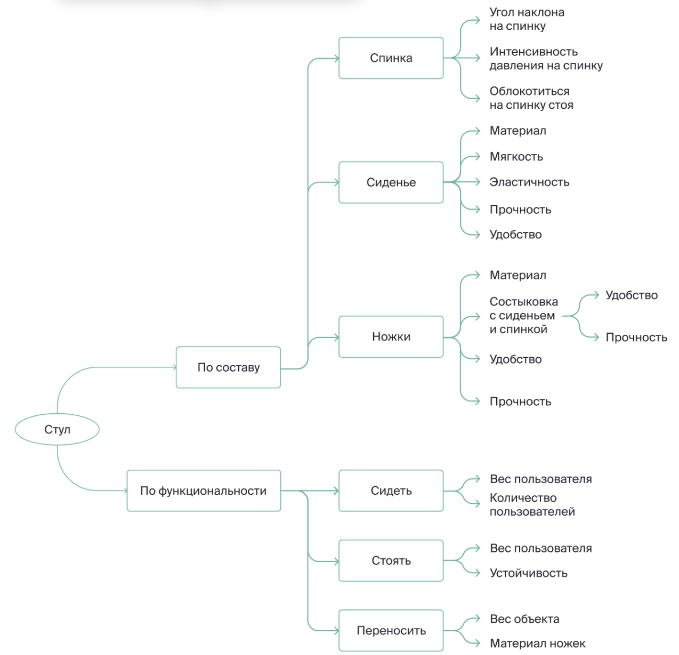
Три правила декомпозиции:

1. Обычно одно требование можно разложить на несколько частей.
2. Требования декомпозируют до атомарного уровня.
3. Требования декомпозируют в рамках существующего описания.

**Что такое диаграмма связей**

**Диаграмма связей или mindmap** (интеллект-карта, ассоциативная карта) — способ визуализации по ассоциативным связям. Позволяет тестировщику отразить требования в удобной форме. Нарисовать такую карту можно после декомпозиции или параллельно с ней.

Например, стул можно декомпозировать по составным частям, материалам и сценариям применения:



Декомпозировать требования лучше поступательно. Например, сначала «разобрать» стул на детали — спинка, ножки и сиденье, а также на разные сценарии — сидеть, стоять и переносить. Следующий уровень — это материалы: дерево, ткань, металл и пластик.

Нарисовать карту можно в разных инструментах. Например:

* [draw.io](http://draw.io/)
* [Miro](https://miro.com/)
* [XMind](https://www.xmind.net/)

**Визуализация требований: блок-схема**

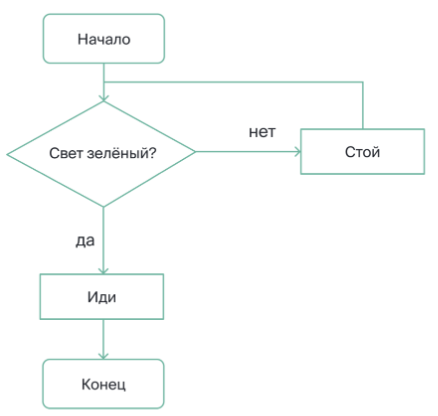
Ещё один способ визуализации — блок-схема. У неё другая цель. Если диаграмма связей отвечает на вопрос «Что есть в приложении?», то блок-схема — на вопрос «Как это работает?».

**Блок-схема** — изображение последовательности шагов, которые нужно выполнить, чтобы решить в приложении определённую задачу.

Например, ты хочешь зарегистрироваться в Кинопоиске и оплатить подписку. Это две разные функциональности — блок-схем тоже будет две.

Ещё на схеме фиксируют, как поведёт себя приложение, если выполнить определённый шаг неправильно: ввести несуществующий адрес электронной почты или не привязать банковскую карту.

Блок-схема, которая описывает взаимодействие пешехода и светофора:



Каждый элемент блок-схемы — это один шаг. Элементы соединены линиями, которые указывают направление.

Каждому элементу соответствует геометрическая фигура.

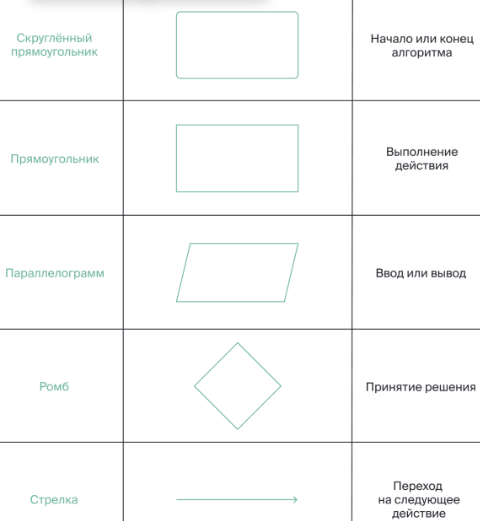
**Обозначения**

Начало и конец схемы — скруглённый прямоугольник.

Действие — обычный прямоугольник. Например: «Стой».

Условие — ромб. Процесс продолжится в ветке, которая соответствует результату. Например:

* Свет зелёный?
* Условие выполнено?
* Значение корректно?



Есть подходящие онлайн-инструменты:

* [draw.io](http://draw.io/)
* [gliffy.com](http://gliffy.com/)
* [lucidchart.com](http://lucidchart.com/)

**Скрытые требования.** Авторы документации не всегда включают требования, которые кажутся им очевидными. Например, текст не должен содержать орфографических ошибок, а вкладка браузера закроется, если нажать сочетание клавиш «Ctrl + W».

**Требований нет.** Бывает, что требований к функциональности в сервисе нет вообще. Например, в онлайн-магазин добавили возможность применять промокод, а приложить требования не подумали.

Искать требования:

* Анализировать продукт и все объекты тестирования. Задавать вопросы: «Какие задачи выполняет элемент?», «Как работает?», «С какими элементами системы связан?»
* Предложить требования, которые формируют функциональность. Например, нужно проверить кнопку заказа в интернет-магазине. Поставь себя на место пользователя: проверь, оформляется ли заказ при нажатии на кнопку, отправляются ли данные заказчика в базу магазина.

А кого спрашивать?

* Разработчика: реализация требований в коде.
* Менеджера: концепция приложения, ожидания пользователя или заказчика, сценарии взаимодействия с приложением.
* Дизайнера: сценарии использования приложения, макеты.

У членов команды может быть неполная информация, поэтому внутренняя коммуникация очень важна.

**Явные требования** фиксируют в документации, по которой разрабатывается сервис. В них — технические характеристики сервиса или продукта: как он должен себя вести в позитивных и негативных сценариях.

В идеальном мире технические характеристики легко найти в документации. Но чаще всего полные и актуальные требования достать очень сложно: это «живые» документы, которые постоянно меняются вместе с приложением.

Документация продукта — потенциальный источник явных требований:

* **Спецификация** описывает поведение интерфейса, компонентов продукта и другие особенности. Иными словами, инструкция к продукту.
* **Техническое задание (ТЗ)** — документ с планом производства продукта: технические характеристики, метрики качества, порядок и условия работ, цель и задачи, сроки, ожидаемые результаты.

**Функциональное и нефункциональное тестирование**

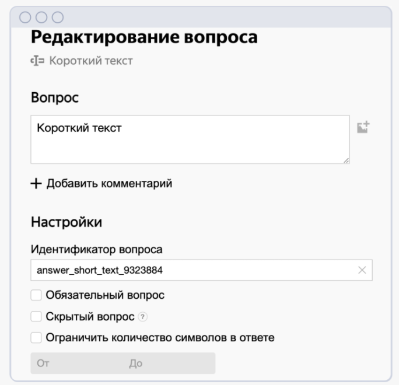
У каждого приложения есть документация, в которой описаны требования.

Например, в документации сказано: «При заполнении поля «Кому» и нажатии кнопки «Отправить» приложение электронной почты должно отправить письмо на указанный адрес».

Это **функциональное требование** — оно описывает, что приложение умеет делать, или его **функциональность**.

Чтобы проверить приложение по функциональным требованиям, проводят **функциональное тестирование**.

Например: требования к функциональности поля «Идентификатор вопроса»:



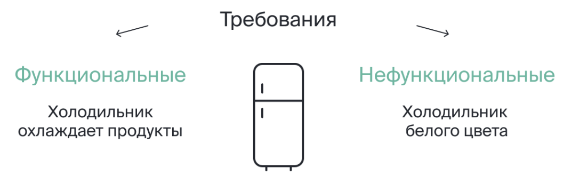
**Поле «Идентификатор вопроса»** по умолчанию заполняется автоматически системным значением: например answer\_short\_text\_5888645. Поле доступно для редактирования. Ограничение — от 1 до 50 символов. Только латинские буквы, цифры и спецсимволы: - (дефис) и \_ (нижнее подчёркивание). При сохранении:

* если поле пустое — проставляется системное значение;
* если в поле введены недопустимые символы — выдаётся сообщение об ошибке: «Введены недопустимые символы»;
* если в поле введены допустимые символы — введённое в поле значение сохраняется.

**Нефункциональное тестирование**

В документации продукта часто есть и **нефункциональные требования**. Это характеристики приложения, которые не влияют на работу напрямую.

Например, по требованиям холодильник должен весить 40 кг. От этого зависят размер и вес деталей, но не функциональность холодильника — охлаждение продуктов.



**Нефункциональное тестирование** помогает проверить приложение по определённой группе критериев, которые не влияют на работу.

Основные нефункциональные проверки — **тестирование безопасности** и **тестирование производительности**.

К ним также относят **тестирование юзабилити** (usability — удобство использования) и **тестирование локализации** — проверку, что интерфейс корректно перевели с одного языка на другой.

**Тестирование безопасности**

Специалист пытается атаковать систему и искать уязвимости:

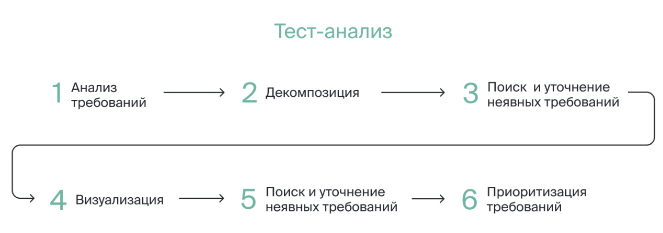
* **XSS (Cross-Site Scripting — межсайтовый скриптинг)**: атака на страницу, когда злоумышленник внедряет вредоносный код и может украсть из форм данные — личную информацию, номер банковской карты.
* **XSRF / CSRF (Сross Site Request Forgery — межсайтовая подделка запроса)** — атака, которая начинается, когда пользователь переходит на сайт по ссылке, а его перенаправляют на фишинговую страницу.
* **Обход авторизации** — попытка получить данные пользователей, которые уже зарегистрированы в системе. Например, специалист по безопасности пытается подобрать пароль к учётной записи. Если ему удаётся, он предлагает повысить требования безопасности к регистрации в системе.
* **DoS или DDoS-атака (Distributed Denial of Service — отказ в обслуживании)** — специалист пытается нагрузить систему разными способами: например, «обстреливает» запросами до отказа системы.

**Тестирование производительности**

Такую проверку проводят, чтобы оценить производительность приложения.

В тестировании производительности различают:

* **Нагрузочное тестирование** — тестировщик проверяет, что приложение соответствует требованиям нагрузки. Если по требованиям система должна обрабатывать до 100 запросов, специалист имитирует такую нагрузку и проверяет, что всё работает без ошибок.
* **Стрессовое** **тестирование** — проверка, что система работает на границах и за пределами допустимой нагрузки. Например, по требованиям система выдерживает 100 запросов. Тестировщик шаг за шагом увеличивает нагрузку выше этой границы, пока сервис не перестанет отвечать на запросы, — и так определяет запас прочности.
* **Тестирование стабильности** — тестировщик проверяет, как работает система при средней нагрузке, но в течение долгого периода. Специалист имитирует среднюю нагрузку на протяжении срока от нескольких часов до нескольких дней. Например, максимум системы — 100 запросов. Тестировщик нагрузит сервис 50-60 запросами на 24 часа. В процессе он следит, как работают память и процессоры серверов.



**Тест-дизайн** — проектирование тестовой документации.

При создании тестовой документации, ты проектируются сценарии тестирования, или **тест-кейсы**. По ним ты будет проверяться приложение.

**Позитивные и негативные проверки**

**Позитивные тест-кейсы** проверяют, что приложение работает без ошибок в двух случаях:

* если использовать его по назначению и не пытаться сломать;
* если вводить корректные данные согласно требованиям.

Например, в требованиях сказано, что приложение принимает в поля «Логин» и «Пароль» только английские буквы и цифры. Авторизация должна пройти успешно, если пользователь введёт логин Ivan и пароль Ivan1234.

Ещё примеры:

1. Микроволновка поддерживает функцию «Гриль». Позитивное тестирование — проверить, что режим включается и работает корректно.
2. Калькулятор должен уметь складывать два числа. Позитивное тестирование — проверить, что сумма вычисляется верно.

**Негативные тест-кейсы** проверяют, как поведёт себя приложение, если:

* использовать приложение не так, как задумывали разработчики;
* вводить данные не из требований.

Когда выполняешь негативные проверки, важно убедиться, что приложение продолжает работать без ошибок и реагирует так, как описано в требованиях.

Например, в требованиях указано, что приложение принимает в поля «Логин» и «Пароль» только английские буквы и цифры. Если ввести в поля какие-нибудь другие символы, приложение не ломается, а выдаёт сообщение «Ошибка».

Ещё пример:

На 0 делить нельзя. В требованиях к работе калькулятора этого нет. Негативная проверка — протестировать, как реагирует система, если попытаться поделить на ноль.

**Порядок проверок**

Сначала стоит проводить позитивные проверки. Они важнее: так ты протестируешь, реализованы ли требования.

Например, карандаш сделали по требованиям: он должен состоять из двух элементов — «стержня» и «оболочки». Если одно из этих требований не выполнено, проводить тесты бессмысленно — не реализованы элементы, которые помогают работать основной функциональности.

Пример позитивного тест-кейса: работает ли двигатель автомобиля? В процессе выясняется, что машина не заводится, — тест-кейс не пройден. Нет смысла начинать негативные проверки — сначала двигатель надо починить.

Если двигатель работает, проводят остальные позитивные проверки. Затем приступают к негативным тест-кейсам: например, проверяют, что будет, если раскрутить двигатель выше максимально возможного числа оборотов. В требованиях не сказано, что происходит, если превысить пороговое значение.

**Требования к новому сервису электронной почты**

**Сценарий написания письма**

Поля: Кому, Копия, Тема, Тело письма.

* В поля «Кому» и «Копия» вводят электронный адрес.
* В поле «Тема» указывают тему письма.
* В поле «Тело письма» набирают текст сообщения.
* Поля «Кому» и «Тема» заполнять обязательно.

**Сценарий управления письмом**

Действия: пометить прочитанным, пометить как спам, удалить.

* Пометить прочитанным можно в любой папке.
* Если письмо пометили как спам, оно попадает в папку «Спам». Пометить как спам можно везде, кроме папки «Спам».
* Удалить письмо — попадает в корзину. Удалить письмо можно откуда угодно, кроме корзины.

**Правила тест-дизайна**

В тест-дизайне есть несколько важных правил, которые нужно соблюдать при проектировании тестов.

У тест-дизайна есть критерии завершённости (англ. definition of done):

* тесты должны покрывать все требования;
* тесты не дублируются.

**На одно требование может понадобиться больше одной проверки.** Например, в требовании сказано: «Схема города N масштабируется». Чтобы протестировать эту функциональность, предстоит проверить работу кнопок масштабирования, корректность отображения при масштабировании — целый тест-сьют.

**Проверки нужно проектировать в рамках документации.** Иными словами — до тех пор, пока ты не убедишься, что вся функциональность из требований есть в тест-кейсах. Если в требованиях не описано, как ведёт себя смартфон при температуре -120 градусов по Цельсию, не следует писать тест-кейс, где телефон погружают в жидкий азот.

**Один тест — одна проверка.** Например, если тестируется, как масштабируется схема города N, стоит создать два теста — на кнопку «+» и кнопку «-». Если объединить их в одном тесте, логика проверки сломается: ошибка на одной кнопке будет означать, что сломалась вся функциональность масштабирования — тогда как вторая кнопка может работать.

**Тесты не должны дублировать друг друга.** Например, нелогично проверять работу кнопки «+» в кейсе «Масштабирование схемы города N», а затем ещё раз, когда в интерфейсе тестируют её отображение.

**Исчерпывающее тестирование**

Исчерпывающее тестирование — это тестирование всех возможных вариантов действий.

На сайте авиакомпании можно выбрать тип питания на борту самолёта при покупке авиабилета. Варианты в выпадающем списке:

* обычное — выбрано по умолчанию;
* детское;
* вегетарианское;
* постное;
* кошерное;
* низкокалорийное.

Чтобы проверить возможность выбора опций меню, нужно шесть проверок — столько же, сколько типов питания. Так ты проверишь работу списка на всех возможных входных данных — проведёшь исчерпывающее тестирование.

Исчерпывающее тестирование применяют, когда сервис влияет на жизнь и здоровье пользователя и цена ошибки крайне высока. Например:

* медицина и биомедицина;
* космическая индустрия;
* оборонная промышленность и вооружённые силы;
* государственная служба;
* платёжные сервисы.

Оно проверяет все возможные параметры и их сочетания: от тестировщика не должна укрыться ни одна, даже самая маленькая ошибка.

Пример работы авторизации. Авторизация по почте: адрес и пароль. Данные, которые вводят в каждое поле, могут быть валидны или невалидны.

|  |  |  |  |
| --- | --- | --- | --- |
| N | ПОЧТА | ПАРОЛЬ | РЕЗУЛЬТАТ |
| 1 | валидная | валидный | успешно |
| 2 | валидная | невалидный | неуспешно |
| 3 | невалидная | валидный | неуспешно |
| 4 | невалидная | невалидный | неуспешно |

В поле можно вводить целые числа от 1 до 100 000. При вводе других значений появляется ошибка «Некорректный ввод». Стоит ли проводить исчерпывающее тестирование вручную?

Нет

Исчерпывающее тестирование — полный перебор всех вариантов. Здесь более 100 000 вариантов ввода.

Исчерпывающее тестирование — это крайний случай: нужно перебрать много вариантов значений — на это нужно очень много времени. На практике это невозможно почти всегда.

Вместо исчерпывающего тестирования ставят цель: покрыть тестами всю функциональность с минимальным количеством тестов. Для этого применяют техники тест-дизайна, о которых ты узнаешь в следующих уроках.

**Эквивалентность**

**Что такое эквивалентность**

Эквивалентность — это равноценность объектов. В тестировании эквивалентными считаются значения, которые приложение обрабатывает одинаково.

Если в требованиях к приложению сказано: «В поле "Фамилия" можно ввести от 2 до 15 символов» — значит, приложение одинаково обработает значения от 2 до 15. Значения от 0 до 1 или больше 15 оно обработает по-другому — например, появится сообщение об ошибке.

Такие значения объединяются в классы эквивалентных значений, или **классы эквивалентности (КЭ)**.

Например, у тебя есть мультиварка с программой «Крупа». В этом режиме можно приготовить любые крупы — гречку, рис, пшено. Программа та же: мультиварка приготовит блюдо за 30 минут. Если по этой же программе приготовить овощи, они переварятся. Крупы и овощи — два разных класса эквивалентности. Результат каждого продукта — разный.

**Требования к элементам тестирования**

В поле можно вводить разные символы. Требования к функциональности ввода отражают, какие комбинации символов можно вводить, а что будет провоцировать ошибку.

Форма регистрации магазина - были выделены элементы, которые нужно протестировать. Например, поле «Имя». Известны требования: оно обязательно для заполнения, можно ввести от 2 до 25 символов — русские и английские буквы, пробел, дефис.

Чтобы проверить поле «Имя», вводят разные значения. Какие есть ограничения ввода в требованиях?

По длине текста

Длина допустимого текста — от 2 до 25 символов.

По типу вводимых символов

Разрешённые символы — русские и английские буквы, пробел, дефис.

Поле нельзя оставить пустым

Поле заполнять обязательно.

Каждый элемент нужно анализировать отдельно — требования часто отличаются.

В поле «Электронный адрес» может быть до 200 символов в формате <символы>@<символы>.<символы>. Можно вводить английские буквы, цифры, тире, нижнее подчёркивание. Тире и подчёркивание не могут быть в начале адреса, перед знаком @ и после него. Это поле обязательно для заполнения.

Какие есть ограничения ввода в требованиях к полю «Электронный адрес»?

Допустимая длина текста

Длина электронного адреса должна быть не больше 200 символов.

Разные типы вводимых символов

Электронный адрес может состоять из английских букв, цифр, тире, нижнего подчёркивания.

Поле нельзя оставить пустым

Заполнять поле обязательно.

Соответствие шаблону

Согласно требованиям вводимые данные должны соответствовать шаблону <символы>@<символы>.<символы>.

В поле «Телефон» может быть от 10 до 12 символов, можно вводить только цифры. Номер может начинаться с «+» или сразу с цифры. Знак «+» не считается символом. Это поле заполнять обязательно.

Какие есть ограничения ввода в требованиях к полю «Телефон»?

Допустимая длина текста

Длина текста должна быть от 10 до 12 символов.

Разный шрифт текста

По требованиям шрифт не влияет на поле «Телефон».

Разные типы вводимых символов

В поле можно вводить только цифры.

Поле нельзя оставить пустым

Заполнять поле обязательно.

Соответствие формату

Телефон должен начинаться с «+» или сразу с цифры.

**Классы эквивалентности**

Класс — это диапазон либо набор значений, вызывающий одинаковую реакцию системы.

Диапазон — интервал чисел с границами: например, пароль может быть длиной от 8 до 20 символов. Набор значений — множество значений, каждое прописано в требованиях. Например, в выпадающем списке на сайте можно выбрать один язык из трёх.

В требованиях к приложению для поля «Комментарий» сказано: «Можно ввести только русские буквы, пробел, точку, тире и запятую». Русские буквы, пробел, точка, тире и запятая — это: набор значений.

***Когда определяешь классы эквивалентности, обращай внимание не только на допустимые, но и на запрещённые входные данные.***

По результатам теста, шкала которого от 0 до 100 баллов включительно, студентам выставляются оценки: если набрано больше 80 баллов — отлично. Если меньше 30 — неудовлетворительно. Чтобы получить хорошо, нужно набрать 60 баллов или больше. Остальное — удовлетворительно. Выдели классы эквивалентности и сформулируй диапазоны значений под эти условия.

От 0 до 29 баллов

Неудовлетворительно

От 30 до 59 баллов

Удовлетворительно

От 60 до 80 баллов

Хорошо

**Этапы выделения классов эквивалентности**

Вернись к примеру поля «Имя». Требование к полю:

|  |  |
| --- | --- |
| ФОРМА ПРЕДЗАКАЗА | ТРЕБОВАНИЯ |
| Имя | Длина: от 2-х символов до 25 символов. Формат: русские буквы, английские буквы, пробел, дефис. Обязательность: да |

Ограничения поля:

* по длине текста;
* по типу вводимых символов, а также наличию пробелов и дефисов в тексте;
* поле нельзя оставить пустым.

Задача — выделить классы эквивалентности длины текста, учитывая ограничения. Порядок действий:

1. Ограничение декомпозируется на диапазон или набор значений? В диапазоне длина значения должна быть в его рамках; в наборе значение нужно выбрать из разрешённого множества.
2. Каковы границы диапазона длины текста? От 2 до 25 символов.
3. Можно ли ввести значения за границами диапазона? Да, меньше 2 и больше 25 символов.
4. Что происходит, если выйти за границы допустимого диапазона? Форма регистрации выдаёт сообщение об ошибке.

Таким образом получается:



**Подбор тестовых значений**

После выделения **классов эквивалентности** (Набор или диапазон значений, которые приложение обработает одинаково.)- следующий этап — подобрать тестовое значение внутри каждого класса.

В тестировании есть гипотеза: «Если система обработала верно одно значение класса, она обработает верно и остальные».

Чтобы подобрать тестовое значение, нужно взять одно значение внутри класса. Значения на границах диапазона нельзя применять в качестве тестовых — это другая группа проверок.

**Граничные значения**

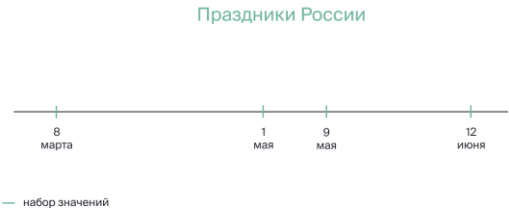
**Граничные значения (ГЗ)** — первая и последняя точки класса эквивалентности — диапазона. Их важно проверять, потому что на границах часто возникают ошибки в приложении.

Если класс — это диапазон значений, обозначают его границы. У наборов граничных значений нет.

Например, в требованиях написано: пароль может быть длиной от 8 до 20 символов. Значит, 8 и 20 — границы класса.



Если класс — это набор, прописывают все значения, которые в него входят. У таких классов границ не бывает. Например, список праздников.



**Как определять границы**

Определить границы можно только у классов, которые представляют собой диапазон, а не набор значений.

В диапазонах допустимы несколько типов данных, например:

* целые числа (1, 2, 3...);
* дробные числа (0.1, 0.2, 0.3...);
* временные интервалы (часы, дни, месяцы).

Границы в требованиях могут прописывать по-разному.

**Явно на интервале чисел** — есть точные значения начала и конца числового интервала. Например, пароль может содержать от 8 до 20 символов.

**Явно на множестве чисел** —в требованиях написано, что операцию можно выполнить при любом целом положительном числе. Например, пополнить счёт мобильного телефона можно не меньше, чем на 1 рубль. У диапазона от 1 до бесконечности есть только минимальная граница — 1: у бесконечности нет границы.

**Неявно на интервале чисел** — в требованиях написаны примерные характеристики интервала, которые нужно уточнить дополнительно. Например, кредит выдают совершеннолетним и не дают пенсионерам.

**Неявно на множестве чисел** — в требованиях написаны примерные характеристики интервала, у которого нет границ или одной из границ. Например, по закону можно получить загранпаспорт в любом возрасте. Минимальная граница — 0 лет, а максимальной нет — нельзя принять за границу определённое число лет.

**Проектирование тестов**

Например, кредит выдают только совершеннолетним и не выдают пенсионерам. Законодательные ограничения: совершеннолетие — с 18 лет, пенсионный возраст — с 65 лет. Чтобы узнать о кредитных возможностях, нужно ввести свой возраст в поле на странице банка.

Чтобы проверить, как работает система на границах:

1. Выделить диапазон значений: от 18 до 64 лет включительно. Это диапазон, когда человек совершеннолетний и не достиг пенсионного возраста.
2. Проверить, что система работает согласно требованиям на границах диапазона, — значения 18 и 64.
3. Проверить, что вне границ диапазона система работает по другой логике, — значения 17 и 65.
4. Проверить, что система работает согласно требованиям внутри границ диапазона, — значения 19 и 63.

Таким образом, в тестирование поля нужно включить шесть проверок — при 18 и 64; 17 и 65; 19 и 63.

Общий **алгоритм проектирования тестов по граничным значениям**:

Например, нужно проверить, что минутная стрелка в часах работает согласно требованиям в диапазоне от 15 до 45 минут. Тип данных — временной интервал. Шаг — минимальная единица, которая увеличивает или уменьшает значение границы, — 1 минута.

1. Определить тип данных и шаг.
2. Проверить, что система работает на границах диапазона: 15 и 45.
3. Сделать шаг за границы диапазона и проверь эти значения: 14 и 46.
4. Сделать шаг внутрь границ диапазона и проверь эти значения: 16 и 44.

**Порядок применения техник КЭ и ГЗ**

Как применять техники классов эквивалентности и граничных значений в правильном порядке.

У оператора можно пополнить счёт мобильного телефона на сумму от 100 рублей 00 копеек до 2999 рублей 99 копеек. Нельзя пополнить счёт на суммы бóльшие или меньшие.

Тип данных: дробное число с точностью до одной сотой, где целая часть — рубли; дробная — копейки. Шаг 0.01.

Классы эквивалентности: а) 0.00 — 99.99; б) 100.00 — 2999.99; в) 3000.00 и больше.



Как проверить пополнение счёта, оптимально применяя техники КЭ и ГЗ?

1. **Проверить значения внутри классов:** 50.00, 1000.00 и 5000.00. Если проверки прошли успешно, переходить к следующему шагу. Если нет — проверять дальше нет смысла: система не работает.
2. **Проверить значения на границах:** 0.00 и 99.99; 100.00 и 2999.99; 3000.00. Если проверки прошли успешно, для этих значений система работает корректно. В последнем классе нет максимальной границы, поэтому её не проверяют.
3. **Проверить значения на один шаг вне границ:** 100.00; 99.99 и 3000.00; 2999.99. Если проверки прошли успешно, удалось подтвердить, что границы верны. В первом классе нельзя задать одно значение вне границ: отрицательную сумму нельзя положить на счёт.
4. **Проверить значения на один шаг внутрь границ:** 0.01 и 99.98; 100.01 и 2999.98; 3000.01. Если проверки прошли успешно, система работает согласно требованиям и внутри всего диапазона: на значения из одного класса система реагирует одинаково.

|  |  |  |  |
| --- | --- | --- | --- |
| ПРОВЕРКА | КЛАСС 1 | КЛАСС 2 | КЛАСС 3 |
| 1 | 50.00 | 1000.00 | 5000.00 |
| 2 | 0.00 и 99.99 | 100.00 и 2999.99 | 3000.00 |
| 3 | 100.00 | 99.99 и 3000.00 | 2999.99 |
| 4 | 0.01 и 99.98 | 100.01 и 2999.98 | 3000.01 |

ПРИМЕР: Поле «Имя» формы регистрации магазина. У каких классов нужно выделить границы?



2-25 символов

У этого класса есть границы.

0-1 символ

У этого класса есть границы.

26 - +∞ символов

У этого класса есть границы.

Границы есть только у тех классов, которые являются диапазонами значений.

Теперь нужно выделить граничные значения. Изучи, как сделать это для класса «Строка длиной 2-25 символов». Пройди по алгоритму:

1. Определить шаг: 1 символ.
2. Проверить, что система работает на границах диапазона: 2 символа и 25 символов.
3. Сделать шаг за границы диапазона: 1 символ и 26 символов.
4. Сделать шаг внутрь границ диапазона: 3 символа и 24 символа.

Результат:



Указать граничные значения класса «Строка длиной 0-1 символов»:

Значения на границах диапазона -0 и 1

Значения на один шаг внутрь диапазона - Не существует

Значения на один шаг за границы диапазона -2

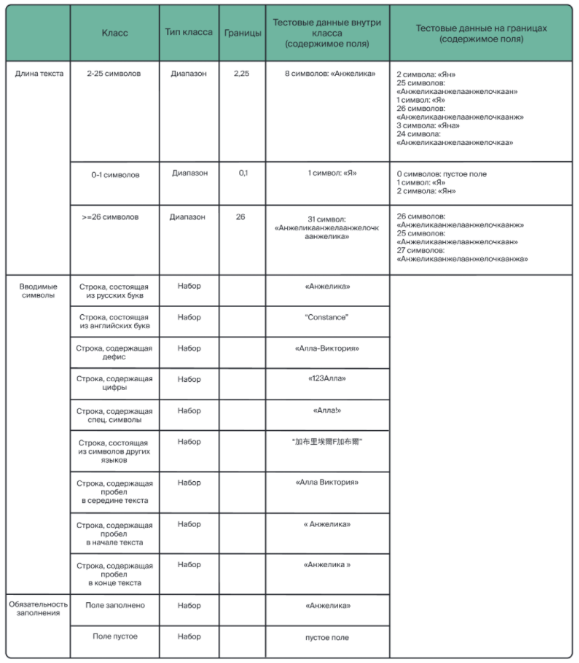
Определить граничные значения класса «Строка длиной 26 — +∞ символов».

Значения на границах диапазона -26

Значения на один шаг внутрь диапазона - 27

Значения на один шаг за границы диапазона - 25

Теперь таблица выглядит [так](https://code.s3.yandex.net/qa/schemes/boundary-values-main-table.png):



**Оптимизация проверок**

В таблице, которая проверяет поле «Имя» по разным классам, есть значения, которые дублируются. Её можно оптимизировать — исключить повторы тестов. Основной принцип оптимизации — сократить дубликаты перед проверкой.

Перед нами два класса эквивалентности из таблицы: «Строка длиной 2-25 символов» и «Строка длиной 0-1 символов» — выделить дубликаты.

Таблица должна выглядеть так:



Значение «2 символа» повторяется дважды. Логичнее оставить тестовое значение в классе «Строка длиной 2-25 символов»: в классе «2-25 символов» эта проверка выполняется раньше, чем в классе «0-1 символов».

Значение «1 символ» дублируется трижды. Оставить значение только в классе «Строка длиной 0-1 символов».

**Пример:**

Вернуться к полному списку проверок. Чтобы протестировать все классы, которые относятся к длине текста, было 15 проверок.



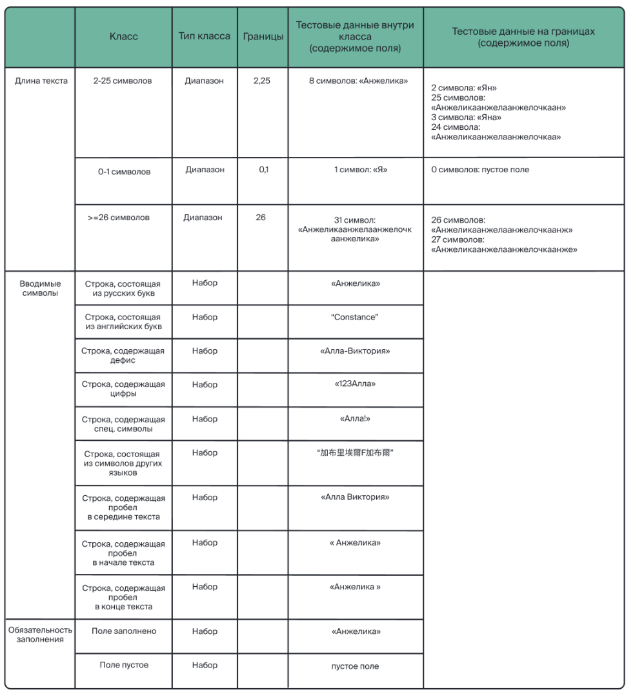
Сколько значений останется после оптимизации проверок длины текста? -10

Значения внутри класса тоже учитывают, когда считают количество тестов.



После оптимизации проверок длины текста таблица выглядит так:



Полная таблица:

В таблице остались другие дубликаты: например, пустое поле. Нам понадобится только один тест. В проверках классов «Поле заполнено», «Строка, состоящая из русских букв» и «Строка длиной 2-25 символов» тоже есть дубликат — Анжелика. Сократи тест до одного.

Получается так:

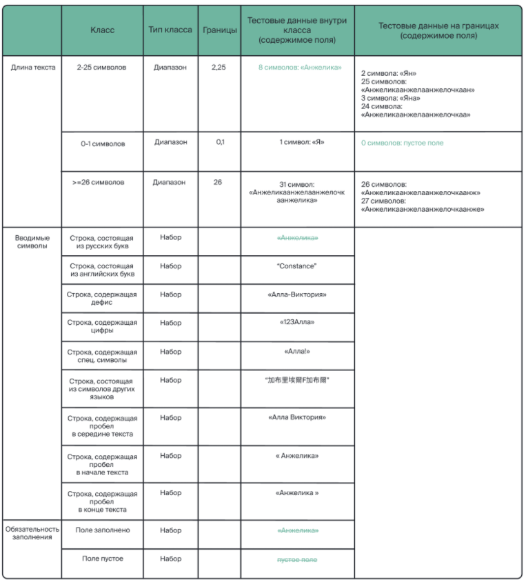
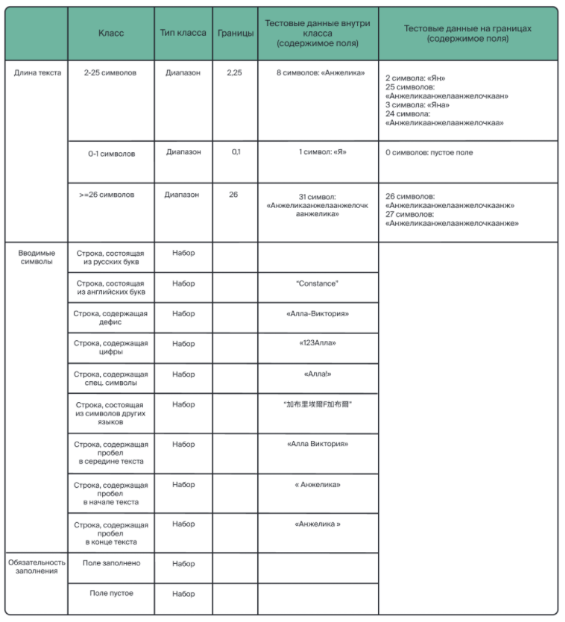


Таблица после полной оптимизации тестов:



**Сложные случаи**

Бывают случаи, когда в документации одно требование пересекается с другим.

Банк разрешает переводить деньги с карты на карту с двумя условиями:

за один раз можно перевести не меньше 50 и не больше 15 000 рублей;

сумма перевода не может превышать сумму, которая есть на счёте.

Два требования пересекаются.

Проверить работу приложения с учётом двух ограничений можно, если разделить интервал на классы в несколько этапов.

По первому условию нужно разделить интервал на три класса с диапазоном значений:

|  |  |  |
| --- | --- | --- |
| N КЛАССА | ДИАПАЗОН, РУБ. | ПОВЕДЕНИЕ СИСТЕМЫ |
| Класс А | 0 - 49 | Перевод невозможен |
| Класс Б | 50 - 15000 | Перевод возможен |
| Класс В | 15001 - ∞ | Перевод невозможен |

Классы спроектированы с допущением, что сумма на счёте не ограничена.

Чтобы соблюсти и второе условие, нужно взять число из диапазона [50; 15000]. Например, на карте осталось 7000 рублей. Класс Б разделится ещё на два с диапазонами значений Б1 и Б2.

|  |  |  |
| --- | --- | --- |
| N КЛАССА | ДИАПАЗОН, РУБ. | ПОВЕДЕНИЕ СИСТЕМЫ |
| Класс Б1 | 50 - 7000 | Перевод возможен |
| Класс Б2 | 7001 - 15000 | Перевод невозможен |

Чтобы проверить, как работает система с учётом сразу двух условий, и оптимизировать количество проверок, нужно соединить обе группы классов. Обратить внимание на столбец «Сумма на счёте» — он содержит условия, которые тоже нужно учитывать при проверке диапазонов.

|  |  |  |
| --- | --- | --- |
| ДИАПАЗОН, РУБ. | СУММА НА СЧЁТЕ | ПОВЕДЕНИЕ СИСТЕМЫ |
| 0 - 49 | не ограничена | Перевод невозможен |
| 50 - 7000 | 7000 | Перевод возможен |
| 7001 - 15000 | 7000 | Перевод невозможен |
| 15001 - ∞ | не ограничена | Перевод невозможен |

Удалось выполнить одно из главных условий тест-дизайна: покрыть требования минимальным количеством проверок.

Например, программа «Кредитный калькулятор». Она рассчитывает максимальную сумму кредита на основании возраста и суммы первоначального взноса.

* Возраст от 18 до 30 лет: кредит не больше 5000$
* Возраст от 31 до 60 лет: кредит не больше 10000$
* Возраст от 61 до 100 лет: кредит не больше 1000$

Если первоначальный взнос менее 10% желаемой суммы, кредит не выдают. В остальных случаях выдают максимальную сумму в зависимости от возраста.

Программа принимает два типа данных:

* возраст человека;
* сумма первоначального взноса.

Выдели классы эквивалентности возраста — диапазоны значений:

* Возраст от 18 до 30 лет
* Возраст от 31 до 60 лет
* Возраст от 61 до 100 лет

На сумму кредита влияет первоначальный взнос. Классы эквивалентности — процент первоначального взноса:

* 0 - 9%
* 10 - 100%

Программа принимает значение суммы первоначального взноса, а не процента. Нужно определить формулу расчёта первоначального взноса:

максимальная сумма кредита в зависимости от возраста \* % первоначального взноса

Чтобы проверить, как работает калькулятор, нужно выделить классы по двум параметрам:

|  |  |
| --- | --- |
| ДИАПАЗОН | ПОВЕДЕНИЕ СИСТЕМЫ |
| Возраст от 18 до 30 лет | 0 - 9% |
|  | 10 - 100% |
| Возраст от 31 до 60 лет | 0 - 9% |
|  | 10 - 100% |
| Возраст от 61 до 100 лет | 0 - 9% |
|  | 10 - 100% |

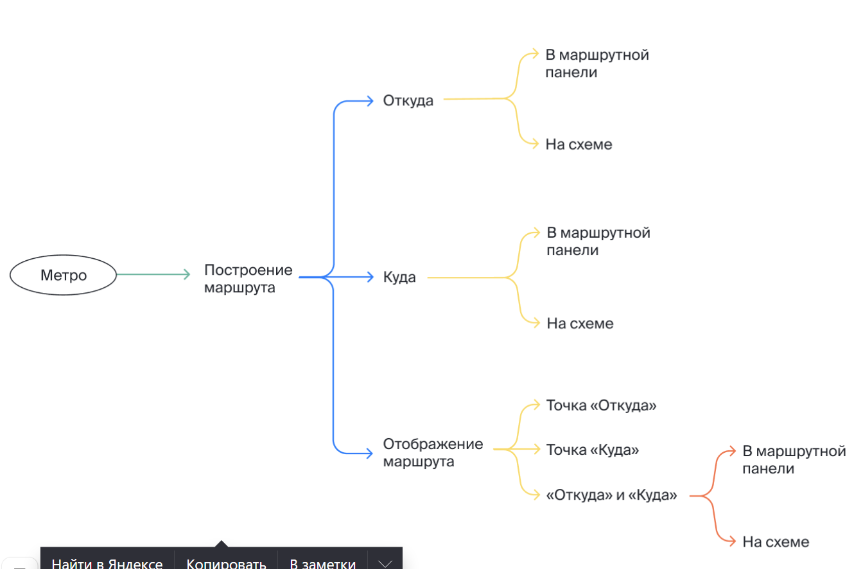
Чтобы рассчитать первоначальный взнос, нужно применить формулу выше:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ДИАПАЗОН | МАКСИМАЛЬНАЯ СУММА | % ПЕРВОНАЧАЛЬНОГО ВЗНОСА | ФОРМУЛА РАСЧЕТА | ОЖИДАЕМЫЙ РЕЗУЛЬТАТ |
| Возраст от 18 до 30 лет | 5000$ | 0 - 9% | макс.сумма кредита \* % первоначального взноса | кредит не выдаётся |
|  |  | 10 - 100% | макс.сумма кредита \* % первоначального взноса | 5000$ |
| Возраст от 31 до 60 лет | 10000$ | 0 - 9% | макс.сумма кредита \* % первоначального взноса | кредит не выдаётся |
|  |  | 10 - 100% | макс.сумма кредита \* % первоначального взноса | 10000$ |
| Возраст от 61 до 100 лет | 1000$ | 0 - 9% | макс.сумма кредита \* % первоначального взноса | кредит не выдаётся |
|  |  | 10 - 100% | макс.сумма кредита \* % первоначального взноса | 1000$ |

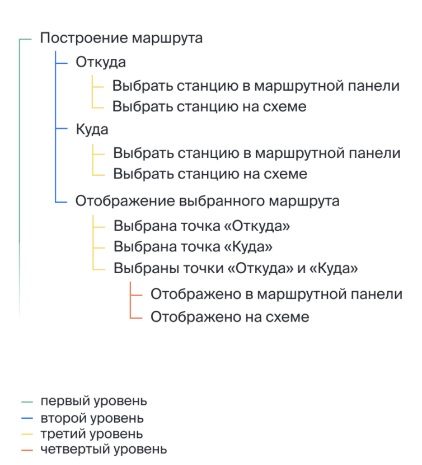
**Чек-лист** — список элементов системы, которые нужно проверить. Каждый пункт чек-листа обязательно содержит элемент.

**Уровни детализации**

Чек-лист удобнее составлять по mindmap. Так можно разложить сервис на проверки по слоям. Например, у тебя нас часть mindmap Яндекс.Метро:



Глядя на mindmap, можно составить такой чек-лист по уровням:



Когда можно детализировать?

1. Когда требований к функциональности не хватает. С появлением новых требований mindmap или чек-листы можно детализировать дальше.
2. Если ты плохо представляешь, как правильно разложить сервис на блоки, сначала стоит создать карту верхнего уровня. Затем полезно уточнить у старшего коллеги, корректно ли сделана карта. После этого можно детализировать дальше.

Глубина детализации чек-листа может быть разной. Зависит от:

* опытности тестировщика, который проходит по чек-листу;
* частоты тестирования функциональности: если её проверяют редко — лучше расписать детальнее, чтобы не забыть;
* подробности требований: если требование общее или его нет совсем, нет смысла детализировать проверку.

Во всём чек-листе уровень детализации должен быть одинаковым.

Когда составляешь чек-лист, обращай внимание на формулировки. От них зависит:

* сможешь ли ты воспроизвести проверки, когда вернёшься к чек-листу через какое-то время;
* поймут ли твой чек-лист коллеги.

Единого шаблона для чек-листов нет: всё зависит от задачи и формата, принятого в компании. Изучи несколько советов, которые помогут сделать чек-лист понятнее и чётче.

**Один пункт — одна проверка**

В одном пункте — одно требование, элемент или ОР. Так проще оценить, соответствует ли ожидаемый результат фактическому.

Например, в такой формулировке зашифрованы сразу несколько проверок:

Маршрут «Откуда-Куда» можно выбрать в маршрутной панели или на схеме, а сами станции «Откуда-Куда» отображаются точками.

Чтобы тестировать было легче, нужно разбить проверку на несколько:

**Работа с маршрутом**

* Выбрать станцию «Откуда» из маршрутной панели.
* Выбрать станцию «Откуда» на схеме.
* Выбрать станцию «Куда» из маршрутной панели.
* Выбрать станцию «Куда» на схеме.
* Проверить, что выбранные станции отображаются точками на схеме.

**Начинать пункт с глагола**

Так формулировку проще понять, потому что она точнее описывает задачу.

Когда пункт чек-листа не содержит глагол — сложнее определить, о чём речь:

**Работа с письмами**

* Входящие
* Исходящие
* Корзина

С глаголом задача сформулирована понятнее:

**Проверка работы с письмами**

* Получить новое письмо
* Отправить письмо
* Удалить письмо

Придерживаться этого правила получится не всегда: не все проверки удаётся сформулировать через глаголы. Например, может появиться пункт «Поле "Вопрос" по умолчанию заполнено значением "Короткий вопрос"». В этом нет ничего страшного. Правило такое: если можешь сформулировать с глаголом — формулируй.

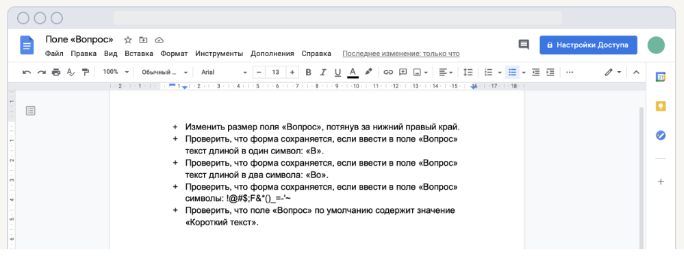
**Не дублировать проверки**

Например, в чек-листе есть проверка ОР: «Станции метро отображаются точками на схеме». Стоит проследить, чтобы она не повторилась в другом месте чек-листа.

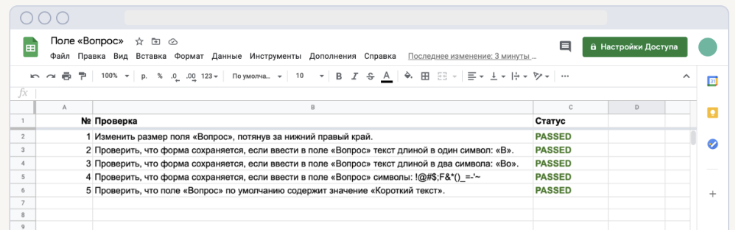
Правила хорошего чек-листа:

* Проверять одним пунктом что-то одно.
* Начинать пункт с глагола.
* Не дублировать проверки.

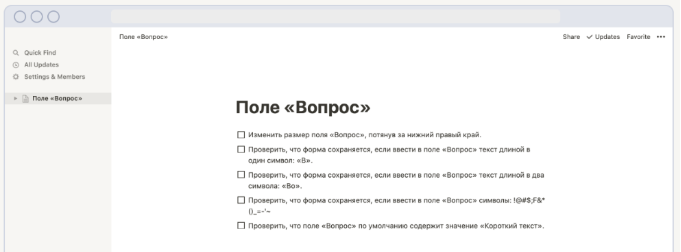
Чек лист пишется в любом текстовом приложении:



В Excel:



В специальных приложениях, которые упрощают работу с чек-листами:



**Собрать тест-кейс**

**Тест-кейс** — это детальное пошаговое описание проверки. Оно показывает определённый путь программы или сценарий проверки.

В отличие от чек-листа каждый тест-кейс — это проверка одного элемента; в тест-кейсе подробно описано, что делать, чтобы достичь ОР.

Тест-кейсы применяют:

* чтобы хранить детальную информацию о продукте: чек-лист, в отличие от тест-кейсов, не детализирован;
* чтобы по ним мог тестировать любой участник команды, даже самый неопытный.

Тест-кейс обязательно содержит **сценарий тестирования**. Для сравнения в чек-листе такой информации нет — тестировщик сам решает, какие шаги совершить, чтобы достичь ОР:

|  |  |  |
| --- | --- | --- |
| ХАРАКТЕРИСТИКА | ЧЕК-ЛИСТ | ТЕСТ-КЕЙС |
| Уровень тестировщика | От мидла до старшего | От новичка до старшего |
| Степень понимания продукта и погружённости | Средний и выше | Любой |
| Сложность создания | Невысокая | Высокая |
| Сложность поддержки | Невысокая | Высокая |

Шаблон тест-кейса создаёт команда. Он может меняться на протяжении жизненного цикла продукта. Обычно в тест-кейсе всегда есть:

1. ID;
2. ожидаемый результат (ОР);
3. заголовок;
4. шаги.

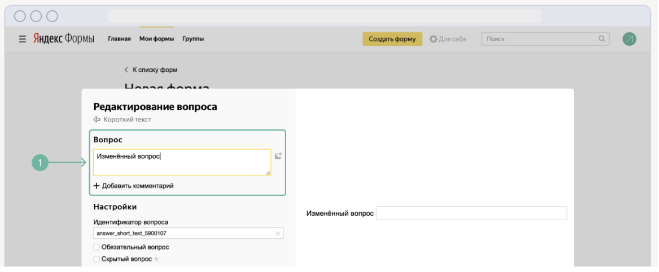
Опциональные элементы:

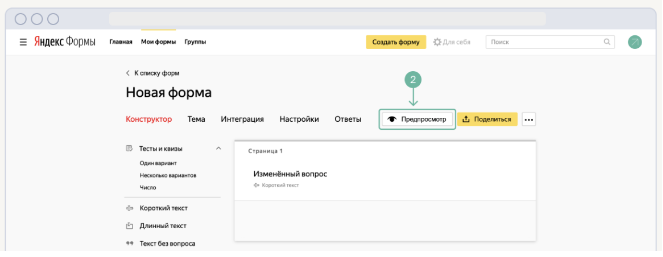
1. предусловие/постусловие — то, что нужно сделать до или после тест-кейса;
2. окружение: ОС, браузер;
3. краткое описание цели — какие требования покрывает тест-кейс;
4. ссылка на требования;
5. теги — обычно присваивают тест-кейсу в зависимости от среза тестирования: например, регрессионное или смоук.

Иногда команда отказывается от любых полей, кроме обязательных ID и ОР.

Например, тебе нужно проверить поле «Вопрос» в Яндекс.Формах.

Требование такое: когда пользователь меняет текст в поле «Вопрос» (1), он должен поменяться и на панели предпросмотра (2).





Пример: каким может быть тест-кейс для этого случая. В нём есть все обязательные элементы.

YNDX-3

**Заголовок:** *[есть заголовок]*

Изменение текста в поле «Вопрос» сразу отображается на панели предварительного просмотра

**Предусловие:** *[опционально включено предусловие: оно нужно, чтобы дальше по шагам можно было создать форму]*

Открыть сайт [forms.yandex.ru](http://forms.yandex.ru/) и авторизоваться под тестовой учётной записью: login/password.

**Шаги:** *[расписаны шаги, как нужно выполнять тест]*

Нажать на кнопку «Создать форму».

На открывшейся странице конструктора форм выбрать тип вопроса «Короткий текст».

В появившемся окне редактирования изменить текст в поле «Вопрос» на «Изменённый вопрос».

**ОР:** *[есть ожидаемый результат]*

На панели предварительного просмотра текст вопроса поменялся на «Изменённый вопрос».

**Как написать хороший тест-кейс**

Тест-кейсы должны быть понятны твоим коллегам, поэтому уделяй время структуре и формулировкам.

Если сформулировать непонятно и запутанно, команде придётся уточнять, что имел в виду автор тест-кейса. Или коллеги-тестировщики выполнят проверку не так, как задумывалось, — и протестируют не то, что нужно.

Проектировать тест-кейсы — это навык.

**Несколько советов, которые помогут написать логичный и понятный тест-кейс.**

**Один тест-кейс — одна проверка**

В тест-кейсе был только один ожидаемый результат. Это поможет не запутаться, лучше следовать этому правилу.

Например, нужно проверить поле «Идентификатор вопроса»: сохраняется ли значение, которое в него вводит пользователь. Изучи тест-кейс для этого случая:

**ID:**

YNDX-1

**Заголовок:**

Валидация поля «Идентификатор»: проверка сохранения формы с введённым идентификатором

**Предусловие:**

Открыть сайт [https://forms.yandex.ru/](https://forms.yandex.ru/admin/myforms) и авторизоваться под тестовой учётной записью: login/password.

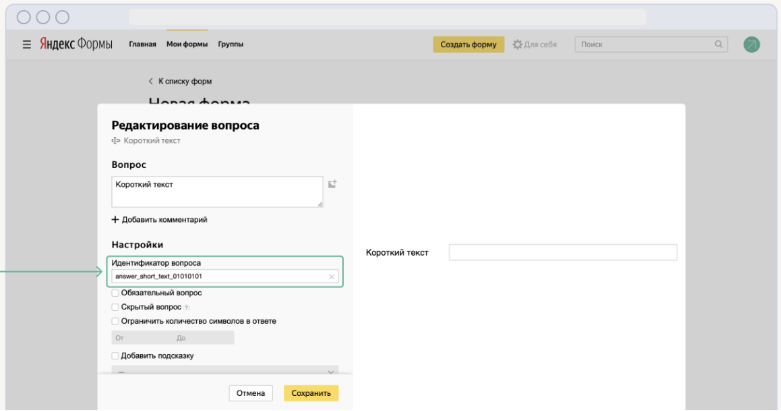
**Шаги:**

1. Нажать на кнопку «Создать форму».
2. На открывшейся странице конструктора форм выбрать тип вопроса «Короткий текст».
3. В появившемся окне редактирования в поле «Идентификатор» ввести 10 символов: 1234567abc.
4. Нажать на кнопку «Сохранить».

**ОР:** *[один тест-кейс — одна проверка]*

Вопрос сохранился с идентификатором 1234567abc.

Окно редактирования вопроса выглядит так:



Если бы в ОР было указано несколько результатов, это затруднило бы проверку.

**Один тест-кейс — одни тестовые данные**

Тестовые данные — данные, которые понадобятся тебе для тестирования. Например, ты проверяешь, что в поле можно ввести текст от 2 до 10 символов. Символы, которые ты вводишь в поле для проверки, — тестовые данные.

В одном тест-кейсе используй одни тестовые данные. Например, если нужно протестировать поле с коротким и длинным именем, составь два разных тест-кейса.

Представь, что ты проверяешь поле «Идентификатор вопроса»: сохраняется ли значение, которое в него вводит пользователь. В шагах тест-кейса есть пункт:

1. В появившемся окне редактирования в поле «Идентификатор» ввести 10 символов: 1234567abc.

Неправильным был бы вариант:

1. В появившемся окне редактирования в поле «Идентификатор» ввести 10 символов: 1234567abc и 10 спецсимволов -!@#$%^&\*().

Здесь сразу два набора тестовых данных. Это ведёт к проблемам:

* менее очевидно, что проверяется в таком кейсе и как его озаглавить;
* непонятно, в каком порядке вводить данные;
* непонятно, нужно ли повторять все шаги для обоих наборов;
* сложнее локализовать проблему, если она возникнет;
* можно забыть проверить какой-то из наборов.

**Уникальный** **ID**

ID тест-кейса не должен повторять другие. Если будет два одинаковых, команда запутается.

Иногда ID для тест-кейсов задаёт система, в которой работает тестировщик. Если такого нет, следуют правилам, принятым в команде. О них тебе расскажут коллеги.

ID может содержать цифры, текст или и то, и другое. Например, YNDX-2.

**Уникальный и полный заголовок**

Хороший заголовок:

* не повторяет заголовки других тест-кейсов — чтобы не запутаться;
* конкретный и отвечает на вопрос «Что я проверяю?» или «Что? Где? Когда?»

Например, удачные заголовки:

* «Отображение подробного маршрута в маршрутной панели» — «Что я проверяю?»
* «Отображение подробного маршрута в маршрутной панели после указания станций "От" и "До"» — «Что, где и когда?»

Если написать просто «Отображение маршрута», непонятно — отображение какого маршрута и где именно.

Ещё удачный пример: «Валидация поля "От": Ввод некорректных значений». Неудачный: «Валидация поля "От"».

**Лаконичные и чёткие шаги**

Когда описываешь шаги:

* не расписывай их чересчур подробно: включай в них только необходимую информацию;
* следи, чтобы каждый шаг отвечал на вопрос «Что нужно сделать?»

Например, нужно проверить, что форма не сохраняется, если пользователь ничего не ввёл в поле «Вопрос». Шаги могут быть такими:

1. Нажать на кнопку «Создать форму».
2. На открывшейся странице конструктора форм выбрать тип вопроса «Короткий текст».
3. В появившемся окне редактирования очистить поле «Вопрос».
4. Нажать на кнопку «Сохранить».

Как было бы слишком подробно:

1. Нажать на кнопку «Создать форму».
2. На открывшейся странице конструктора форм выбрать тип вопроса «Короткий текст».
3. В появившемся окне редактирования очистить поле «Вопрос».
4. Далее идёт кнопка добавления комментария. *[этот пункт лишний. Он не отвечает на вопрос «Что нужно сделать?»]*
5. После этого чек-боксы «Обязательный вопрос» и «Ограничить количество символов в ответе». *[пункт не отвечает на вопрос «Что нужно сделать?»]*
6. Кнопка «Добавить подсказку». *[пункт не отвечает на вопрос «Что нужно сделать?». А ещё для проверки не нужно добавлять подсказку: шаг лишний.]*
7. Проставить в поле «От» значение 1, в поле «До» значение 3, выбрать любую подсказку. *[пункт лишний: для проверки ОР это не нужно.]*
8. Нажать на кнопку «Сохранить».

**Если для проверки нужны особые настройки, указать предусловие**

Предусловие — всё, что нужно сделать до того, как приступать к основным шагам тест-кейса. Например, зайти в приложение под определённым логином или включить какие-то настройки.

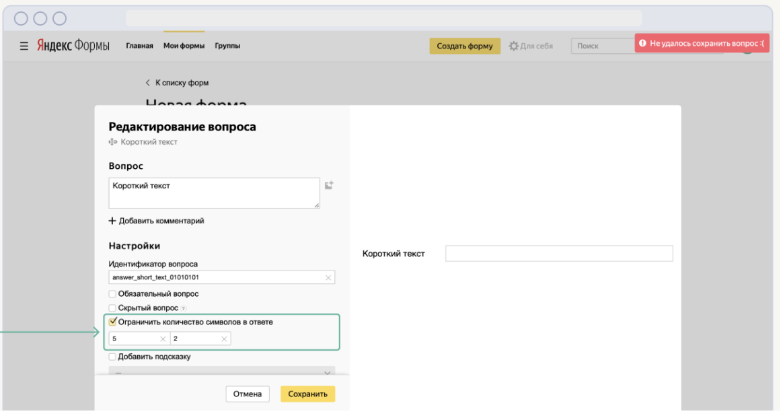
Например, чтобы проверить формы, нужно авторизоваться в Яндекс.Формах. Без этого не получится ни создать форму, ни отредактировать. Поэтому в предусловии тест-кейсов можно указать: «Открыть сайт [https://forms.yandex.ru/](https://forms.yandex.ru/admin/myforms) и авторизоваться под тестовой учётной записью: login/password».

Правила хорошего тест-кейса:

1. Один тест-кейс — одна проверка.
2. Один тест-кейс — одни тестовые данные.
3. Уникальный идентификатор.
4. Полное и уникальное название — отвечает на вопрос «Что нужно проверить?»
5. Лаконичные и чёткие шаги — отвечают на вопрос «Что нужно сделать?». Описано только самое необходимое.
6. Если нужно что-то сделать до тест-кейса, это описано в предусловии.

Пример хорошего чек - кейса:

Например, ты тестируешь чек-бокс «Ограничить количество символов в ответе» по требованиям. Нужно проверить: если минимальное количество символов больше максимального, форма не сохранится.



**ID:** YNDX-4 *[есть ID, он не дублирует предыдущий пример]*

**Заголовок:**

Значение поля «От» не может быть больше значения поля «До» *[название сформулировано как ответ на вопрос «Что я проверяю?»]*

**Предусловие:** *[есть предусловие, без которого не получится выполнить тест-кейс]*

Открыть сайт [https://forms.yandex.ru/](https://forms.yandex.ru/admin/myforms) и авторизоваться под тестовой учётной записью: login/password.

**Шаги:** *[каждый шаг сформулирован как ответ на вопрос «Что нужно сделать?»]*

1. Нажать на кнопку «Создать форму».
2. На открывшейся странице конструктора форм выбрать тип вопроса «Короткий текст».
3. В появившемся окне редактирования включить чек-бокс «Ограничить количество символов в ответе».
4. Заполнить поле «От» значением 5.
5. Заполнить поле «До» значением 2.
6. Нажать на кнопку «Сохранить».

**ОР:** *[один тест-кейс — одна проверка]*

Форма не сохранилась.

**Типичные ошибки в написании тест-кейса**

Начинающие тестировщики могут допускать ошибки при проектировании тест-кейсов. Типичные:

* В тест-кейсе не хватает тестовых данных.
* Дублируются заголовки тест-кейсов.
* В одном тест-кейсе несколько разных проверок.
* Не указан ID тест-кейса.