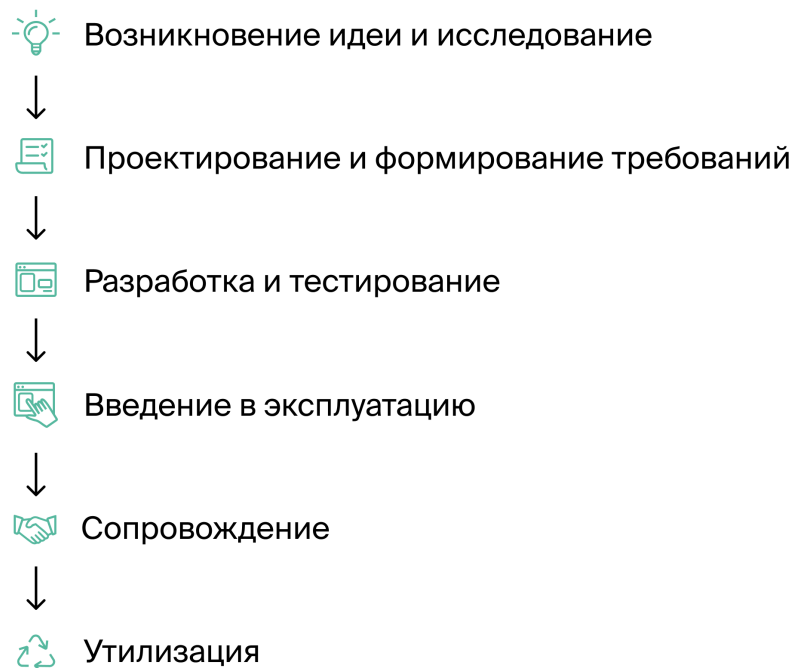


Шпаргалка: Разработка, тестирование и анализ требований

Приложение: от идеи до утилизации

Разработчики создают приложение, а тестировщики его проверяют. Но это только один из этапов **жизненного цикла 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.

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

Процессы 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.

Задача на разработку и тестирование проходит через четыре этапа:

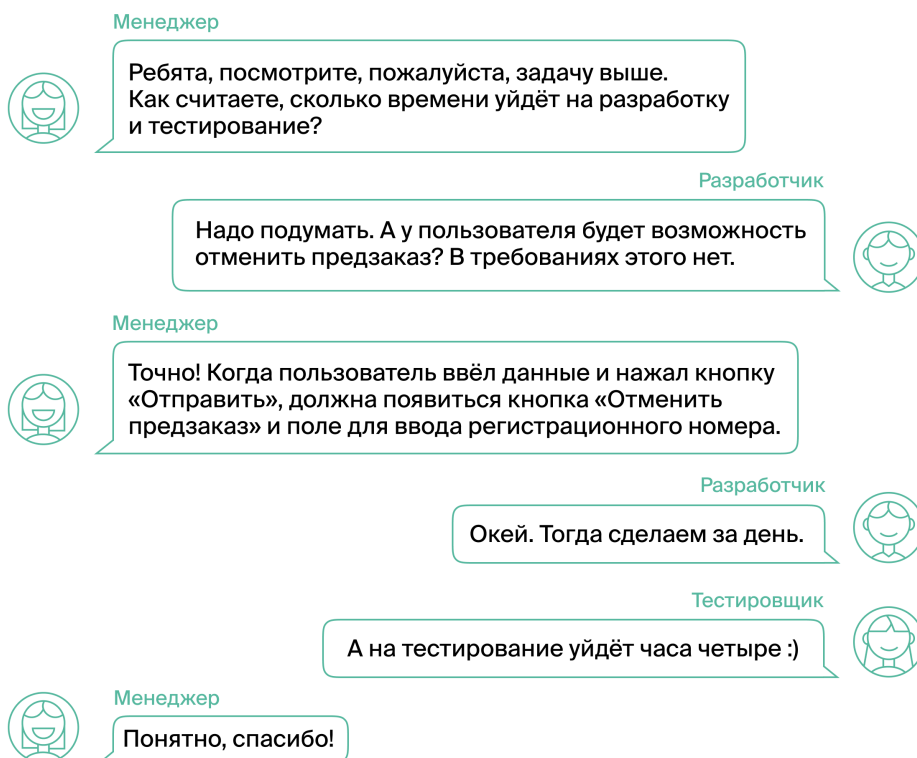


Постановка задачи

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

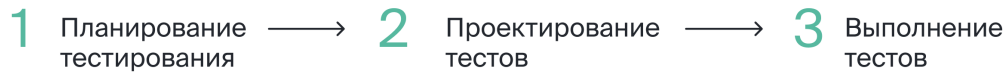
Когда новая функциональность готова, тестировщики проверяют её по чек-листам и тест-кейсам, которые составили на предыдущем этапе. А разработчики исправляют ошибки, которые обнаружили тестировщики.

Как тестировщик работает над задачей

Этапы тестирования

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

Этапы тестирования



Планирование тестирования

На этом этапе нужно бегло изучить требования и оценить, сколько времени займёт тестирование. Сначала тебе будет помогать старший тестировщик, но позднее ты научишься составлять план самостоятельно.

Проектирование тестов

Этап проектирования состоит из двух шагов:

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

Выполнение тестов

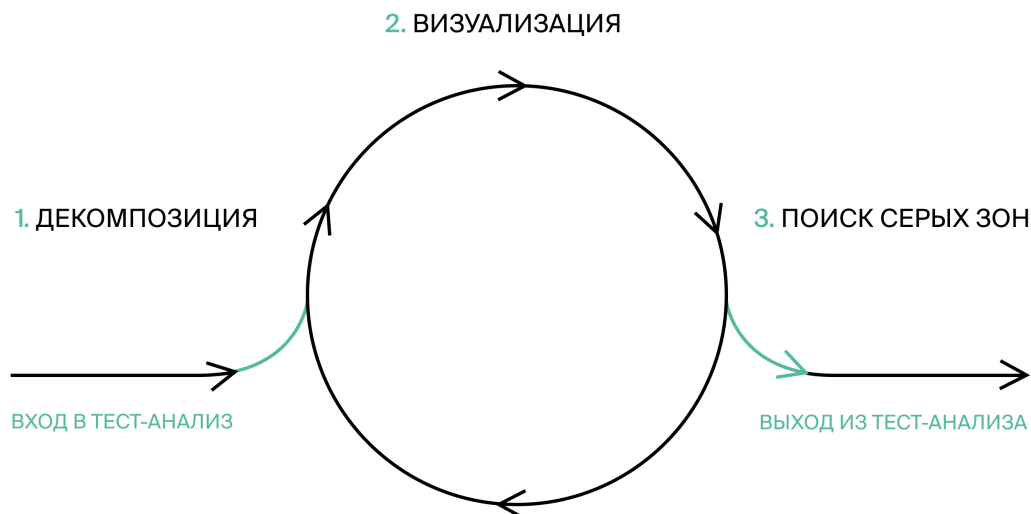
Когда список проверок готов, переходи к тестированию. Если обнаружишь баг — оформи баг-репорт.

Введение в тест-анализ

Тест-анализ (test analysis) — изучение требований и макетов. Когда будешь изучать, постарайся ответить на вопрос: что именно предстоит тестировать? Так ты составишь список объектов тестирования.

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

Тест-анализ



Декомпозиция

Декомпозиция — разделение целого на части. На этом этапе тебе 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 декомпозируешь требования, попробуй их визуализировать.

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

Поиск серых зон

Когда декомпозируешь и визуализируешь требования, сможешь видеть в них несостыковки, противоречия и пропуски — **серые зоны**. В этом случае обращайся к менеджеру, он поможет разобраться.

Иногда приходится искать серые зоны повторно. Например, тебе удалось заметить неявные требования, уточнить их и дополнить схему. Но оказалось, что в ней всё ещё есть слепые зоны. Их тоже нужно уточнить, декомпонировать

и визуализировать: так ты получишь полное представление о том, что предстоит тестировать.

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

Декомпозиция требований

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

Три правила декомпозиции:

Правило первое

Обычно одно требование можно разложить на несколько частей.

Правило второе

Требования декомпозируют до атомарного уровня: так, чтобы поделить их ещё раз было уже нельзя.

Правило третье

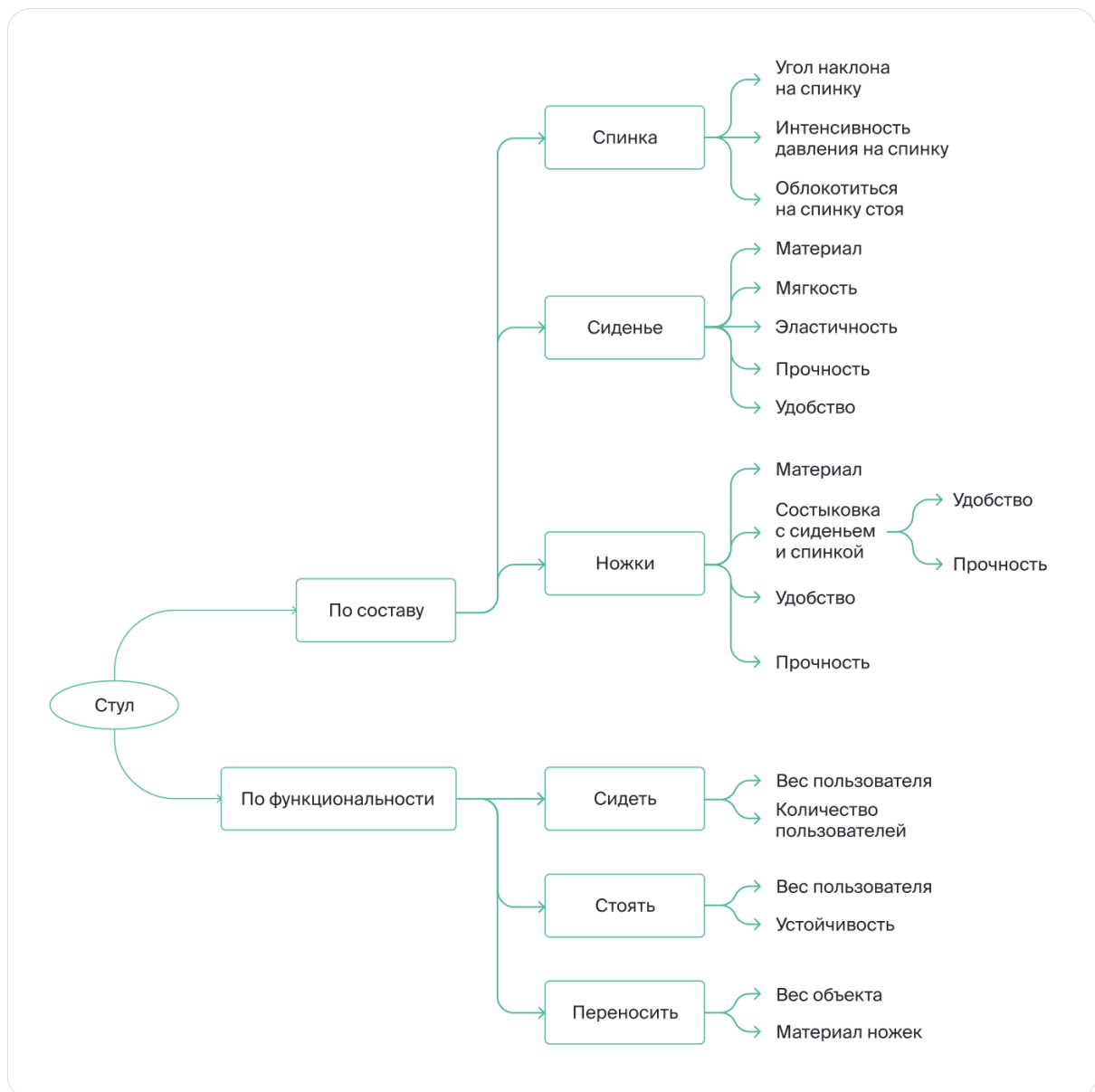
Требования декомпозируют в рамках существующего описания.

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

Визуализация требований: 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
- [Miro](https://miro.com)
- [XMind](https://xmind.com)

Как нарисовать диаграмму связей

Твоя задача — поэтапно разбить требования на части, которые лучше всего отразят функциональность.

Зафиксируй на карте первый уровень требований, а затем декомпозируй каждое до атомарного уровня.

Когда требований много, декомпонировать и визуализировать их лучше параллельно — так удобнее.

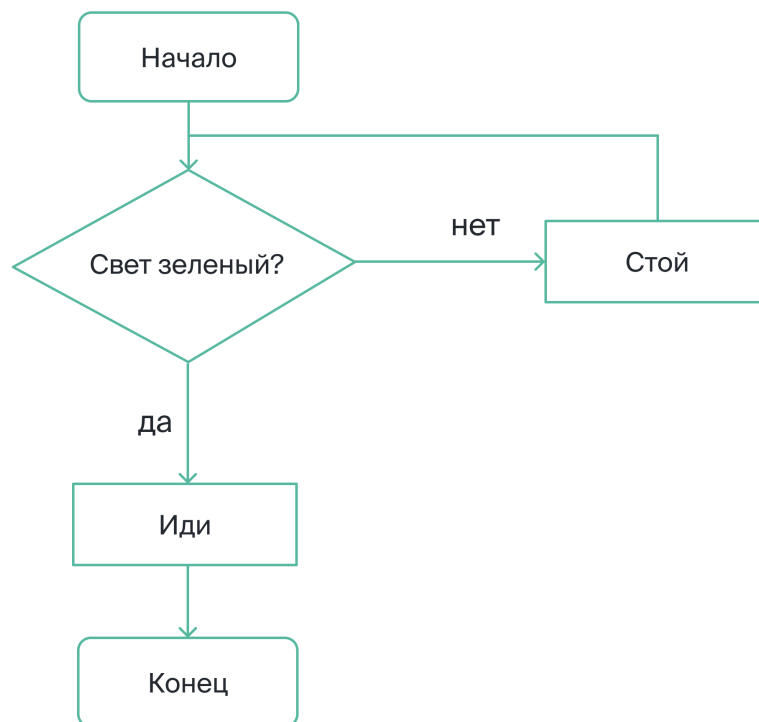
Визуализация требований: блок-схема

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.



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

Каждому элементу соответствует геометрическая фигура.

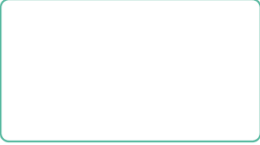


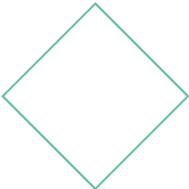

Обозначения

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?

Скруглённый прямоугольник		Начало или конец алгоритма
Прямоугольник		Выполнение действия
Параллелограмм		Ввод или вывод
Ромб		Принятие решения
Стрелка		Переход на следующее действие

Есть подходящие онлайн-инструменты:

- draw.io
- gliffy.com
- lucidchart.com

Но не стесняйся рисовать от руки: можно зачёркивать, переделывать, исправлять. Это менее кропотливо, чем отрисовывать в приложении «на чистовик».

Постарайся запомнить разницу между диаграммой связей и блок-схемой. Диаграмма в структурированном виде показывает, из каких элементов состоит приложение. Блок-схема иллюстрирует, как работает определённая функциональность — от начала до конца.

Серые зоны

Неполные требования — обычная ситуация в жизни тестировщика.

Ты можешь столкнуться с такими проблемами:

Неуточнённые и неполные требования. С первого раза их не заметить. Например, в требованиях к сервису Яндекс.Такси есть требование «Можно применить промокод». При этом не прописано, как промокод ввести и как он влияет на цену при запросе машины.

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.

Как 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.

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.

Поиск требований

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

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

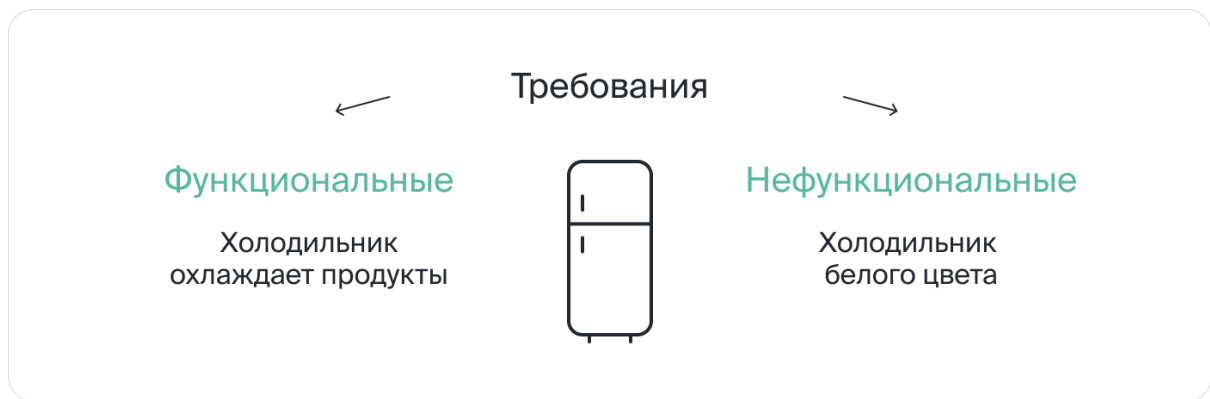
Это **функциональное требование** — оно описывает, что приложение умеет делать, или его **функциональность**.

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

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.



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}}[qa_proektirovanietestov_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 (Cross Site Request Forgery — межсайтовая подделка запроса)** — атака, которая начинается, когда пользователь переходит на сайт по ссылке, а его перенаправляют на {{фишинговую страницу}} [qa_proektirovanietestov_фишинговая страница].
- **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}} [qa_proektirovanietestov_запрос] 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.