GeekBrains

Основы ручного тестирования

Уровни тестирования

На этом уроке

1. Узнаем, на каких уровнях и как тестируется программное обеспечение.
2. Рассмотрим, что такое тест-кейс и чем он отличается от чек-листа.
3. Научимся тестировать по тест-кейсу.

Оглавление

Уровни тестирования по значимости функциональности

В программном обеспечении не все функции одинаково важны.

1. Одни используются чаще и представляют собой «визитную карточку» программы. Если в них появится поломка, то это заметит очень много людей, которые не смогут выполнить привычные операции.
2. Другие — реже, они удовлетворяют потребности узкого круга пользователей.
3. Некоторые не обращаются к ним совсем. Поломка в этих функциях не приведёт к тяжёлым последствиям, но определённая часть пользователей пострадает.

Для примера рассмотрим функциональность мессенджера. Обычно он умеет:

1. Отправлять текстовые сообщения.
2. Отправлять файлы (фото, видео и т. д.).
3. Отправлять стикеры.
4. Устанавливать напоминания.
5. Импортировать или экспортировать переписку.
6. Менять тему оформления.
7. Устанавливать аватарку.
8. Изменять имя пользователя.

Очевидно, что пользователи каждый день обмениваются сообщениями, а тему оформления или аватарку изменяют гораздо реже. Соответственно, при тестировании больше времени и внимания уделяется обмену сообщениями, и меньше — остальным функциям.

В соответствии со значимостью функциональности определяют три вида тестирования:

1. Smoke (дымовое).
2. Тестирование критического пути.
3. Расширенное тестирование.

Smoke-тестирование

Дымовое тестирование (smoke) — проверка самой важной, ключевой функциональности, неработоспособность которой делает бессмысленной идею использования приложения.

Такое тестирование показывает степень работоспособности ПО и определяет решение о дальнейшем тестировании. Если дымовое тестирование выявляет ошибки, дальнейшее тестирование нецелесообразно. Для продолжения тестирования требуется, чтобы разработчики устранили все дефекты в основной функциональности.

Дымовое тестирование проводят на небольшом наборе простых тестов. Тесты для дымового тестирования автоматизируют в первую очередь.

Пример дымового тестирования формы регистрации:

1. Поля «логин» и «пароль» обязательные.
2. Пароль маскируется.
3. Регистрация считается успешной при валидном логине и пароле.

Тестирование критического пути

Тестирование критического пути исследует функциональность, которую используют типичные пользователи в повседневной деятельности.

Это базовый вид тестирования. Если дымовое тестирование прошло успешно, то на уровне критического пути проверяются все функции, требуемые пользователю для успешной работы с приложением и для достижения поставленных целей. Тестирование критического пути требует больше времени, проводится вручную и автотестами.

Пример тестирования критического пути приложения «Интернет-магазин»:

1. Товар добавляется в корзину.
2. Товар оплачивается картой.
3. Работает отправка сообщения в службу поддержки.

Расширенное тестирование

Расширенное тестирование проверяет всю заявленную в требованиях функциональность, в том числе низкого приоритета и незначительной важности.

Такое тестирование проводится, если есть достаточный запас времени. Оно затрагивает те сценарии использования приложения, которые не проверялись на предыдущих уровнях. Ошибки,

обнаруженные на этом этапе тестирования, не критические и не представляют угрозы для успешного использования приложения.

Пример расширенного тестирования: при тестировании приложения «Интернет-магазин» проверяются правильность заполнения раздела и документация.

В набор для расширенного тестирования попадают тесты критического пути, для которых временно снижен приоритет. Например, функциональность давно не изменялась, поэтому риск возникновения дефектов невысок.

Уровни тестирования по значимости функциональности

Уровни тестирования по степени детализации

Юнит-тесты

Модульное (компонентное, юнит) тестирование — это тестирование наименьших элементов ПО, которые тестируются по отдельности: модули, объекты, классы, функции.

Цель модульного тестирования — выявить ошибки в коде и определить степень готовности приложения или системы к переходу на следующий уровень разработки и тестирования.

Модульное тестирование выполняется разработчиками и опирается на методы анализа кода. Дефекты при модульном тестировании исправляются сразу, без составления отчёта об ошибке. Тестирование автоматизировано.

Тесты, которые создают разработчики для проведения модульного тестирования, называются юнит-тестами. Создание юнит-тестов одновременно с написанием кода или до его написания обязательно на проектах, заинтересованных в качественной, эффективной разработке.

Юнит-тест быстро обнаружит ошибку в коде, а разработчик её сразу исправит. Если на проекте нет юнит-тестов, понадобится больше времени и усилий для обнаружения этой же ошибки в коде при выполнении ручного тестирования.

Интеграционные тесты

Интеграционное тестирование — тестирование части системы, состоящей из двух и более модулей, направленное на проверку взаимодействия этих модулей.

Модули, качественно работающие сами по себе, при взаимодействии вызывают ошибки. Это связано с тем, что модули написаны разными разработчиками или с использованием разных классов и методов. Задача интеграционного тестирования — поиск:

- дефектов, связанных с ошибками реализации и интерпретации интерфейсного взаимодействия между модулями;
- ошибок взаимодействия с другими частями системы — операционной системой, оборудованием.

Пример интеграционного тестирования: тестирование взаимодействия веб-приложения «Интернет-магазин» с платёжным сервисом.

Системные тесты

Системное тестирование — процесс тестирования системы в целом, чтобы проверить, соответствует ли она требованиям. Задача системного тестирования — выявление дефектов, связанных с работой системы в целом:

- неверное использование ресурсов системы;
- непредусмотренные комбинации данных пользовательского уровня;
- несовместимость с окружением;
- непредусмотренные сценарии использования;
- отсутствующая или неверная функциональность;
- неудобство в применении.

В системном тестировании тестовая среда соответствует рабочей среде выполнения. То есть настройки и характеристики тестовых серверов совпадают с настройками и характеристиками

серверов, на которых приложение функционирует. Тестирование проводится на уровне пользовательских интерфейсов.

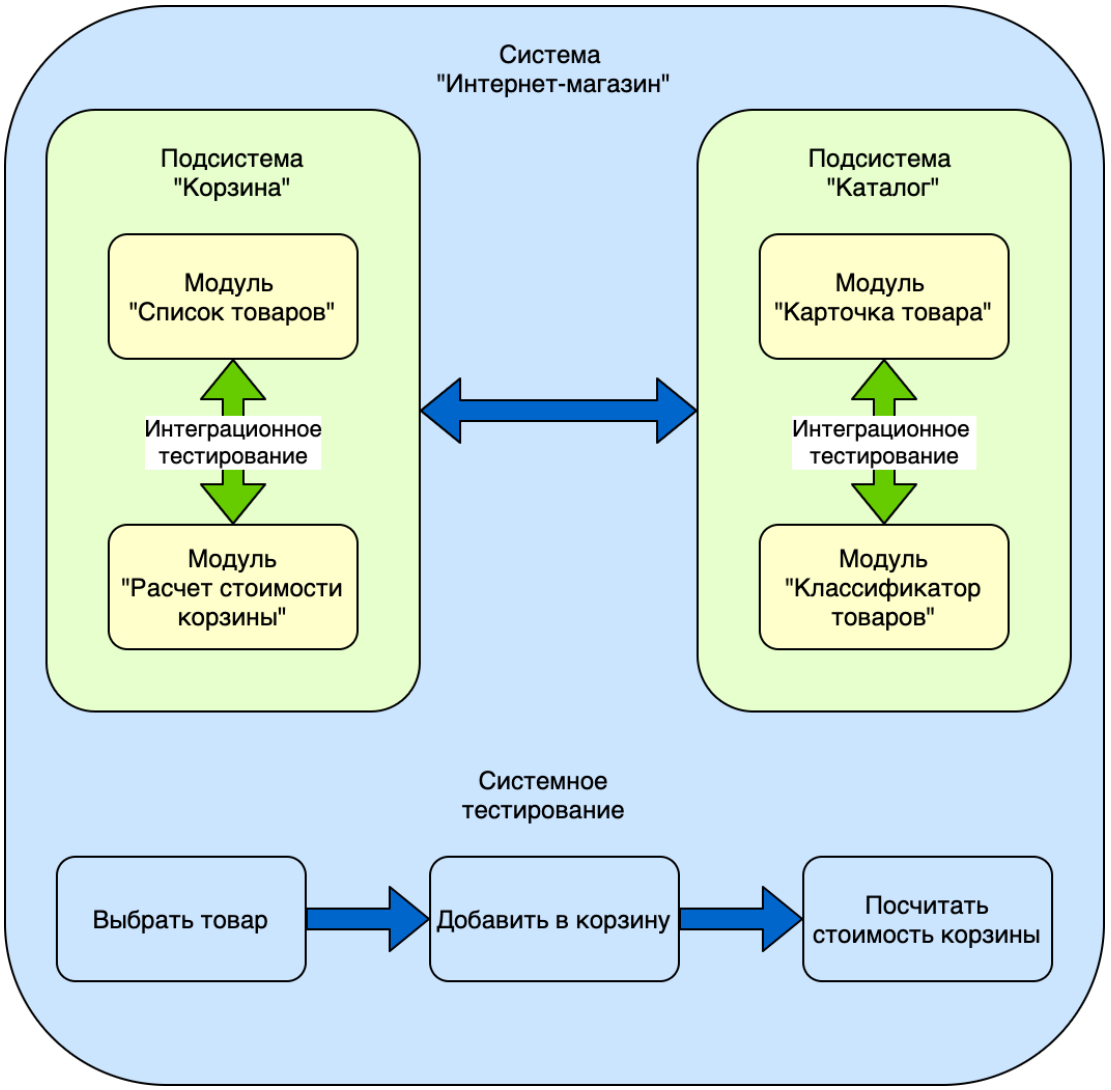
Пример системного тестирования: тестирование процесса заказа товара в приложении «Интернет-магазин» от входа пользователя на сайт до оплаты покупки.

Приёмочное тестирование

Приёмочное тестирование — формальный процесс тестирования, проверяет соответствие системы потребностям, требованиям и бизнес-процессам пользователя. Проводится для вынесения решения о приёме приложения заказчиком.

Такое тестирование проводится на этапе сдачи готового продукта заказчику. Его цель — определение готовности продукта и его соответствия требованиям и ожиданиям заказчика. Приёмочное тестирование — финальный этап, он не считается очень тщательным и подробным. Главным образом проверяется работоспособность основного инструментария. Тестирование проводится на основании предварительно отобранных тест-кейсов:

- самим заказчиком;
- тестировщиками, которые представляют интересы заказчика;
- тестировщиками компании-разработчика — зависит от предпочтений заказчика и предварительных договорённостей.



Уровни тестирования по степени детализации

Тест-кейс

Если тестируемое ПО сложное, тестировщик недостаточно знаком с функциональностью, или требуется подготовка тестовой среды, то чек-листов недостаточно. В этом случае нужен сценарий или инструкция, где пошагово описывается алгоритм проверки. Такой документ называется тестовым случаем, или тест-кейсом.

Тестовый случай (тест-кейс) — это совокупность шагов, конкретных условий и параметров, требуемых для проверки реализации тестируемой функции или её части.

Тест-кейс — это небольшая инструкция по проверке работы функции. В этой инструкции описывается:

- какие шаги нужно выполнить, например, какие кнопки нажать, чекбоксы активировать, данные внести, чтобы проверить работу участка системы;

- какой результат должен быть у каждого шага.

Атрибуты тест-кейса

У тест-кейса есть обязательные атрибуты. Большая часть заполняется на этапе проектирования тест-кейса, несколько атрибутов заполняются только после прохождения теста.

Атрибуты на этапе проектирования тест-кейсов:

1. Идентификатор тест-кейса — уникальный идентификатор, который присваивается автоматически.
2. Название — краткое описание сценария, который проверяет тест-кейс.
3. Шаги — порядок действий, чтобы проверить работу функции или выполнение сценария.
4. Ожидаемый результат — как должна вести себя система после каждого шага тест-кейса.
5. Приоритет тест-кейса зависит от приоритета функций и сценариев, которые он проверяет и определяет очерёдность выполнения теста на очередном этапе тестирования.
6. Тестовые данные — данные, используемые для проверки, указываются в шагах тест-кейса или выносятся в отдельный файл с указанием ссылки на них.
7. Предусловия — действия, которые нужно выполнить, прежде чем приступать к тест-кейсу, а также настройки приложения и тестовой среды.
8. Постусловия — выполнение тест-кейса, как правило, переводит систему из одного состояния в другое: изменяются настройки, производятся расчёты. Состояние, в которое нужно привести систему после прохождения тест-кейса, указывается в постусловии.

Атрибуты на этапе исполнения тест-кейса

1. Фактический результат — как и ожидаемый, указывается для каждого шага тест-кейса. Если фактический результат совпадает с ожидаемым, то в графе с фактическим ставится статус passed. А если результат отличается от ожидаемого — статус failed. В этом случае тестировщик создаёт отчёт о дефекте. К тест-кейсу добавляется ссылка на дефект, который обнаружился при его исполнении.
2. Статус тест-кейса обозначает результат исполнения этого кейса или причину, по которой он не может исполняться.
    a. passed — все шаги проверены, ПО на 100% соответствует ожиданию.
    b. failed — поведение не соответствует ожиданию, найден новый дефект. known bug — поведение не соответствует ожиданию, найден известный дефект.

c. broken — тест-кейс нуждается в редактировании. Например, в нём есть неизвестные термины, битые ссылки, серьёзные неточности, из-за которых невозможно его выполнить, он непонятен.

d. blocked — выполнение тест-кейса невозможно из-за дефекта, который его блокирует.

e. skipped — тест-кейс пропущен. Например, в кейсе указывается дополнительное условие, которое делает невозможным его прохождение: обращение к нереализуемому компоненту.

Правила работы с тест-кейсами

1. Один тест-кейс — одна проверка или один сценарий.
2. Заголовок точно описывает суть тест-кейса. По заголовку определяется характер проверок, если не открывать шаги теста.
3. Точные названия для элементов приложения: кнопок, чекбоксов, элементов меню. Тест-кейсы используются разными людьми — всем должно быть понятно содержание этих тест-кейсов.
4. Простой технический стиль без объяснений базовых понятий работы ПО. Подробности описываются в требованиях или спецификации.
5. Нет пропущенных шагов: каждое следующее действие вытекает из предыдущего.
6. Нет зависимостей от других тест-кейсов. Тест-кейс обычно имеет отсылки к другим тестам. Плохая практика — жёстко упорядочивать проведение тестов, каждый тест должен выполняться независимо от других.
7. Нет дублей с другими тест-кейсами.
8. Обнаруженная ошибка становится очевидной. Шаги тест-кейса и ожидаемые результаты описываются таким образом, чтобы отклонение от них явно свидетельствовало о дефекте.
9. Гибкость для модификации. В тест-кейсе легко добавить, изменить или убрать шаг, в том числе в середине теста, а также изменить тестовые данные.

Рассмотрим основные ошибки, которые чаще всего допускают при создании тест-кейсов.

Ошибки в тест-кейсах

1. Заголовок тест-кейса отсутствует или сформулирован некорректно.
2. Ссылки ведут на разные или недействительные требования.
3. Используются личные формы глаголов: «нажми», «перейдите», «укажи».

4. Используется будущее или прошедшее время в описании ожидаемых результатов: «страница загрузится», «данные были получены».
5. Пунктуационные, орфографические, синтаксические ошибки.
6. «Выдумывание» особенностей поведения приложения без отсылки к требованиям.
7. Нет описания приготовления к выполнению тест-кейса, если требуются предусловия.
8. Полное дублирование (копирование) одного тест-кейса на уровнях дымового тестирования, тестирования критического пути, расширенного тестирования.

Тест-кейс и чек-лист

И тест-кейс, и чек-лист — это упорядоченная последовательность проверок работы программы. Они используются на любом этапе тестирования: дымовом, регрессионном, системном или приёмочном. Тест-кейс и чек-лист представляют собой документацию к продукту и содержат информацию о его функциональности.

Различия тест-кейса и чек-листа фиксируются в таблице:

	Тест-кейс	Чек-лист
Детализация	Проверки детализированы, расписаны по шагам.	Проверки формулируются в общем виде. Пункт чек-листа = заголовок тест-кейса.
Связность	Тест-кейсы независимы друг от друга и выполняются в произвольном порядке.	Проверки могут быть связаны друг с другом, предполагается их последовательное выполнение.
Понятность	Понятен любому человеку.	Понятен человеку, который знаком с продуктом.
Эффект пестицида	Есть риск возникновения, т. к. каждый раз воспроизводятся одни и те же шаги.	Риск снижается, т. к. каждый выполняет проверку по-своему.

Ценность тест-кейсов и чек-листов

1. Структурируют и систематизирую подход к тестированию: позволяют оценить объём предстоящей работы, распределить время задачи между тестировщиками в команде, не пропустить важных проверок.
2. Основа для метрик тестового покрытия: позволяют оценить, сколько процентов требований протестируются, а на какие требования тесты ещё не появились.
3. Основа для увеличения тестового покрытия: если тест-кейсов недостаточно, то нужно добавить дополнительные.

4. Показатель соответствия ситуации плану:
    - сколько тестов уже выполнено;
    - какие из них прошли успешно;
    - как много осталось проверить;
    - какие функциональности тестировались или нет.
5. Поддерживают взаимопонимание между заказчиком, разработчиками и тестировщиком: написание тест-кейсов часто приводит к дополнительным вопросам по работе приложения, и это закрывает пробелы в знаниях о системе, устраняет недопонимание с заказчиком или разработчиком.
6. Хранят информацию для длительного использования и обмена опытом между сотрудниками и командами. Тест-кейсы и чек-листы используются, чтобы удобно передавать знания о системе другим членам команды, а также обращаться к ним при решении спорных вопросов — при условии, что тест-кейсы написаны качественно и корректно.
7. Основа регрессионного тестирования: тест-кейсы делают повторные проверки регулярными и полноценными.

Наборы тест-кейсов

Тест-кейсы объединяются в наборы по тематике, проверяемому модулю или тестовому циклу, в котором они выполняются.

Набор тест-кейсов — test case suite, test suite, test set — совокупность тест-кейсов, выбранных с общей целью или по общему признаку. Иногда в такой совокупности результаты завершения одного тест-кейса становятся входным состоянием приложения для следующего тест-кейса.

Наборы тест-кейсов бывают последовательными и свободными. В последовательных наборах каждый следующий тест-кейс зависит от результата предыдущего.

Преимущество таких наборов в том, что каждый следующий тест-кейс в наборе, представляющий собой входные данные приложения, получает результат работы предыдущего тест-кейса. Это позволяет сократить количество шагов в отдельных тест-кейсах. Такие последовательные действия точнее имитируют работу пользователей.

Сильная связанность тестов — это и слабая сторона последовательных тест-наборов. Если один из тестов завершился неудачно, все последующие выполнить невозможно.

В свободных наборах тест-кейсы выполняются в произвольном порядке. В этом случае неудачное выполнение предыдущего тест-кейса не блокирует работу следующего — и тестировщик может проходить их в любом порядке.

Инструменты для работы с тест-кейсами

Тест-кейсы нужно структурировать, хранить и поддерживать, обеспечивать тестировщикам возможность совместно работать с ними.

Инструменты для работы с тест-кейсами (test-case management tools) — автоматизированные приложения, которые позволяют управлять тест-кейсами. К ним относятся:

1. Jira + Zephyr.
2. Microsoft Team Foundation Server.
3. Test IT.
4. TestLink.
5. Прочие.

Основные функции таких инструментов:

1. Создание тест-кейсов: приложения содержат специальные формы и шаблоны, которые ускоряют процесс разработки тест-кейсов.
2. Выполнение тест-кейсов и запись результатов: тест-кейсы запускаются с использованием приложения, и результат их исполнения записывается в систему.
3. Отслеживание дефектов: возможность прикреплять ссылку на дефект, который обнаружился тест-кейсом. Это позволяет эффективнее отслеживать исправление дефектов.
4. Отслеживание требований и других проектных документов: в приложениях можно прикреплять ссылки на требования, по которым разрабатывается тест-кейс, и отслеживать полноту покрытия требований тест-кейсами.
5. Защита тест-кейсов: тест-кейсы создаются для многократного использования, поэтому они должны защищаться от несанкционированных изменений и удалений. Все действия тестировщиков с тест-кейсами отслеживаются и сохраняются в истории версий.

Контрольные вопросы

1. Зачем нужны тест-кейсы?
2. Каковы атрибуты тест-кейсов?
3. В чём плюсы и минусы тест-кейсов и чек-листов?

Практическое задание

1. Откройте документ, созданный в практическом задании №2. Далее откройте вкладку «Тест-кейсы». Проведите тестирование лендинга по тест-кейсам. Укажите результат тест-кейса. Выделите зеленым цветом шаги, которые пройдены успешно, и красным - на которых тест-кейс упал. Определите уровень тестирования: smoke, критического пути, расширенное (выберите из выпадающего списка).
2. Тест для самопроверки - https://coreapp.ai/app/player/lesson/614344c140a70c676714c754 (сдавать не нужно)

Глоссарий

Дымовое тестирование (smoke) — проверка самой важной, ключевой функциональности, неработоспособность которой делает бессмысленной идею использования приложения.

Инструменты для работы с тест-кейсами — test-case management tools — автоматизированные приложения, которые позволяют управлять тест-кейсами.

Интеграционное тестирование — тестирование части системы, состоящей из двух и более модулей, направленное на проверку взаимодействия этих модулей.

Модульное (компонентное, юнит) тестирование — это тестирование наименьших элементов ПО, тестируются по отдельности: модули, объекты, классы, функции.

Набор тест-кейсов — test case suite, test suite, test set — совокупность тест-кейсов, выбранных с общей целью или по общему признаку. Иногда в такой совокупности результаты завершения одного тест-кейса становятся входным состоянием приложения для следующего.

Расширенное тестирование проверяет всю заявленную в требованиях функциональность, в том числе низкого приоритета и незначительной важности.

Тестирование критического пути исследует функциональность, которую используют типичные пользователи в повседневной деятельности.

Тестовый случай (тест-кейс) — это совокупность шагов, конкретных условий и параметров, необходимых для проверки реализации тестируемой функции или её части.

Дополнительные материалы

1. Статья «Что такое тест-кейс и как его писать».
2. Статья «В тестировании всегда начинаем с простого!».
3. Статья «Элементы интерфейса сайта».

4. Статья «Чек-лист для фильтров на сайтах».

5. Статья «Пишем максимально эффективный тест-кейс».