IT CAREER HUB

Урок 1.

Введение в тестирование

Что такое тестирование 2
7 принципов тестирования 7
Основные виды тестирования 18

Что такое тестирование

Как бы вы описали, что такое тестирование программного обеспечения (ПО)?

Тестирование программного обеспечения (Software Testing) - проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом.
[IEEE Guide to Software Engineering Body of Knowledge, SWEBOK]

Давайте подробнее разберем каждую часть определения.

Что такое реальное и ожидаемое поведение?
Откуда мы можем узнать как именно должно функционировать ПО?

Реальное поведение (actual behaviour, actual result) - то, как по факту функционирует наше ПО.

Ожидаемое поведение (expected behaviour, expected result) - то, как должно функционировать ПО.

Источником знаний о том, какое поведение системы является ожидаемым, выступают требования (requirements).

Требования могут быть как задокументированные (то, к чему хочется стремиться - чтобы все требования документировались), так и незадокументированные (ввиду недостатка времени/ресурсов документируются не все требования).

Примерами источников задокументированных требований могут являться:

- техническое задание;
- спецификации;
- документы с бизнес-требованиями;
- модели и диаграммы;
- майндмэпы;
- контракты и соглашения и т.д.

Мы бы действительно хотели, чтобы все-все требования были описаны в каком-либо документе, опираясь на который, мы бы однозначно могли понимать ожидаемое поведение разрабатываемой системы. Однако, жизнь вносит свои коррективы.

По причине того, что написание и чтение документов требует временных и ресурсных затрат, зачастую в реальной жизни мы сталкиваемся с ситуацией, когда требования есть, но они не зафиксированы письменно.

Пример.

Представим, что нам необходимо реализовать функциональность авторизации через почту/пароль. Очевидно, что в требованиях не будет расписано, что в случае успешной авторизации пользователь “зайдет” в систему, а в случае неуспешной увидит ошибку. Подобное поведение системы является очевидным и опирается на привычный пользовательский опыт и здравый смысл.

Источниками не задокументированных требований являются здравый смысл, жизненный опыт, ответы на уточняющие вопросы по задокументированным требованиям.

Что значит “осуществляемая на конечном наборе тестов”**?**

Вторая часть определения “осуществляемая на конечном наборе тестов” говорит о том, что протестировать все существующие варианты функционирования системы просто невозможно ввиду сложности разрабатываемого ПО.

Пример.

Представим, что у нас есть текстовое поле, в которое можно вводить только латинские символы длинной не более 5 символов. Рассчитаем количество всех комбинаций символов:

Всего 26 латинских символов. Ограничений по строчным/заглавным не указано, поэтому оба варианта допустимы. Таким образом, на каждой из пяти позиций может находится 52 варианта символов. Всего $52^5 = 380\,204\,032$ вариантов вводимых значений. А это всего одно текстовое поле длины 5.

По этой причине мы вынуждены проводить тестирование на ограниченном наборе случаев. Вернемся к этому тезису при обсуждении принципов тестирования.

Что значит “выбранном определенным образом”?

Заключительная часть определения “выбранном определенным образом” говорит о том, что существуют некоторые подходы, позволяющие ограничить набор тестируемых случаев, не понижая уровень качества. Эти самые подходы и есть техники тест-дизайна. Их мы изучим в рамках урока 7.

Цель тестирования — верификация, валидация и обнаружение ошибок для их исправления.

Таким образом, поиск дефектов - это одна из целей тестирования, но не его суть.

Верификация проверяет документацию, дизайн и код на соответствие требованиям, а валидация оценивает конечный продукт на соответствие ожиданиям клиента.

Различие между верификацией и валидацией:

- **Верификация** отвечает на вопрос: правильно ли создается ПО? Проверка на соответствие требованиям и корректную работу функционала.
- **Валидация**: проверка того, соответствует ли продукт ожиданиям клиента.

Разве ожидания клиента и согласованные требования - это не одно и то же?

К сожалению, зачастую нет. Нужно принять тот факт, что заказчик может плохо разбираться в технических нюансах и то, что было согласовано в требованиях, может расходиться с его реальным представлением системы. Таким образом, валидация на соответствие ожиданиям не менее важна, чем верификация на соответствие требованиям.

Когда мы говорим про гибкие методологии разработки, например, SCRUM, валидация осуществляется в рамках демонстрации очередного результата работы, полученного за определенный промежуток времени. В рамках проведения демонстрации лица, заинтересованные в проекте дают обратную связь относительно части проекта, разработанной за крайний промежуток времени работы над продуктом.

 **Баг (ошибка) — отклонение фактического результата от ожидаемого.**

 **Исторический пример.**

В 1945 году был обнаружен первый "настоящий баг" — мотылек, застрявший в реле компьютера Mark II.

Пример.

В качестве примера бага можно описать случай, когда, согласно требованиям, после авторизации пользователь системы управления обучением должен автоматически перенаправляться на страницу с расписанием, а по факту попадает на страницу личного профиля.

Ожидаемый результат (Expected result): пользователь находится на странице с расписанием.
Фактический результат (Actual result): пользователь находится на странице личного профиля.

Подробнее о багах и баг-репортах мы поговорим на одном из следующих занятий.

7 принципов тестирования

Принцип №1. Тестирование не доказывает отсутствие багов, оно лишь снижает вероятность их наличия.

Данный принцип отражает ограниченность тестирования в выявлении всех возможных дефектов в программном продукте. Даже если все тесты пройдены успешно, это не гарантирует, что в системе отсутствуют ошибки. Этот принцип является ключевым для понимания роли тестирования в процессе разработки программного обеспечения.

- Тестировщики обычно не могут проверить все возможные комбинации данных и сценариев, особенно в сложных и многофункциональных системах. В результате, тесты охватывают только часть возможных ситуаций, которые могут возникнуть при эксплуатации программы.

- Даже при хорошем покрытии тестами, программа может столкнуться с нестандартными условиями эксплуатации, которые сложно учесть во время тестирования. Покрытие тестами - это одна из метрик оценки качества тестирования, представляющая из себя плотность покрытия тестами требований либо исполняемого кода. Например, поведение системы может зависеть от нагрузки, настроек на конкретных устройствах или даже от комбинации действий пользователя.

  

  Пример.

  В веб-приложении может не проявиться ошибка, если пользователь работает в обычном режиме. Однако при повышенной нагрузке на сервер (например, когда многие пользователи одновременно добавляют данные в базу), может произойти сбой. Если тестирование не охватывает такие нагрузки, проблема может остаться невыявленной, хотя обычные тесты показывают корректную работу приложения.

- Тестировщики могут допустить ошибки при составлении тестов, из-за чего некоторые сценарии останутся непроверенными. Или же тестировщики могут неверно интерпретировать требования и не заметить несоответствия.

Пример.

Тестировщик проверяет систему заказов, но пропускает редкий сценарий, когда один и тот же товар заказывают два разных пользователя с одинаковыми скидками одновременно. В реальном использовании системы это может привести к конфликту при обновлении данных о наличии товара на складе. Если этот сценарий не проверен, ошибка останется незамеченной.

Принцип №2 Исчерпывающее тестирование невозможно — необходимо использовать анализ рисков и приоритизацию.

Принцип "исчерпывающее тестирование невозможно" означает, что протестировать все возможные аспекты и сценарии работы системы полностью — задача невыполнимая. Причиной тому является огромное количество комбинаций условий и данных, которые необходимо проверить, чтобы охватить все возможные ситуации. Для большинства приложений количество возможных комбинаций входных данных, действий пользователя и внутренних состояний системы настолько велико, что протестировать их все практически невозможно. Если представить даже простую форму с полями ввода, каждое из которых поддерживает различные значения, общее количество возможных комбинаций очень быстро вырастает до миллионов и миллиардов вариантов.

Пример.

Допустим, есть простая форма заказа с полями для ввода имени, возраста и выбора города. Если:

- поле "Имя" принимает 1000 различных вариантов,
- возраст может быть в диапазоне от 18 до 65 (48 значений),
- город можно выбрать из 50 вариантов,

то количество возможных комбинаций для тестирования составит:

1000×48×50 = 2400000 комбинаций.

Протестировать все их вручную или автоматически становится слишком ресурсоемким и требует огромных затрат времени.

По этой причине в рамках проведения тестирования выбирается лишь часть случаев для проверки, принимают существующие риски не обнаружить баг в тех комбинациях, которые не были протестированы.

Принцип №3. Раннее тестирование помогает снизить затраты.

Ранее тестирование действительно помогало снизить затраты на разработку и поддержку программного обеспечения. Этот подход известен как "shift-left testing" — перенос тестирования на более ранние этапы разработки, начиная с момента планирования и анализа требований. Рассмотрим на примере кнопки с неправильной надписью "Продать" вместо "Купить", почему раннее выявление ошибок экономит время и деньги и как они могут повлиять на пользователей и репутацию компании.

Пример ошибки: кнопка "Продать" вместо "Купить".

Представьте, что на странице товара онлайн-магазина вместо кнопки "Купить" случайно написано "Продать". В результате этой ошибки пользователи могут не завершить покупку, так как текст кнопки вызовет у них сомнение и недоверие к платформе. Вот как это может сказаться на бизнесе и затратах:

Потери от упущенных продаж:

- Пользователь, видя кнопку "Продать" вместо "Купить", может подумать, что на сайте ошибка и отказаться от покупки, что приведёт к потере потенциального дохода. Один ушедший покупатель может быть не столь значимой потерей, но если эта ошибка будет присутствовать длительное время, множество пользователей могут уйти к конкурентам.

Репутационные издержки:

- Ошибки в интерфейсе подрывают доверие пользователей к сайту. Покупатели могут задаться вопросом: "Если они не могут правильно написать кнопку, можно ли доверять их товарам или сервисам?" Такие мелкие ошибки формируют общее негативное впечатление и увеличивают вероятность, что пользователи уйдут к конкурентам, где интерфейс кажется более надёжным.

Затраты на исправление ошибки на поздних этапах:

- Если ошибка обнаруживается после того, как продукт уже вышел в продакшн, её исправление становится сложнее. Чтобы изменить надпись на кнопке, придётся обновить код, провести тестирование, убедиться, что исправление не повлияло на другие функции, и выпустить обновление. Это занимает время и требует ресурсов — разработки, тестирования, CI/CD-процессов. CI/CD, или непрерывная интеграция/непрерывная доставка или развертывание, - это практика разработки программного обеспечения, обеспечиваемая автоматизацией.
- Кроме того, при каждом деплое могут возникнуть непредвиденные сложности, которые увеличат затраты на исправление ошибки. Деплой — это размещение готовой версии программного обеспечения на платформе, доступной для пользователей.

Раннее тестирование, а также внимательное изучение требований, помогает обнаруживать ошибки до того, как они переходят в код. Это позволяет избежать серьезных затрат на исправление, улучшить впечатление пользователей от продукта и повысить его надёжность. Выявление проблемы на этапе анализа требований занимает гораздо меньше времени и практически не требует затрат, а ошибка, выявленная на поздних этапах, ведет к потерям в продажах, репутационным издержкам и расходам на разработку и поддержку.

Поэтому раннее тестирование и внимательная работа с требованиями позволяют значительно сэкономить ресурсы и снизить риски для бизнеса.

 **Принцип №4. Кластеризация дефектов — большинство ошибок сосредоточены в небольшом количестве модулей.**

Принцип кластеризации дефектов заключается в том, что большая часть ошибок, как правило, сосредоточена в небольшом количестве модулей или компонентов системы. Этот принцип широко известен и в тестировании программного обеспечения, и в управлении проектами, и отражает правило Парето (80/20), где 80% дефектов обычно сосредоточены в 20% кода.

Почему возникает кластеризация дефектов?

- **Сложные модули:** Некоторые части системы могут быть более сложными с точки зрения логики, алгоритмов или функциональности. В таких местах вероятность возникновения ошибок выше, так как

сложность кода увеличивает риск ошибок при разработке и тестировании.

- **Часто изменяющиеся модули:** Модули, которые часто обновляются и изменяются, могут содержать больше дефектов. С каждой новой модификацией в коде увеличивается риск появления новых ошибок, особенно если тестирование или рефакторинг проводились некачественно.
- **Недостаток тестирования:** Если определенные модули недостаточно тестировались или не тестировались вовсе, они могут содержать больше скрытых ошибок, которые со временем могут накопиться и начать оказывать серьезное влияние на систему.
- **Низкое качество кода:** Плохое проектирование, устаревший или некачественно написанный код также могут приводить к тому, что в одном и том же месте будет обнаруживаться много дефектов.

Принцип кластеризации дефектов подчёркивает, что ошибки в программном обеспечении неравномерно распределены по системе. Большинство дефектов часто сосредоточено в нескольких модулях или компонентах, что делает их целью для прицельного тестирования и улучшения качества. Это знание помогает командам тестировщиков и разработчиков сосредоточить усилия на наиболее проблемных областях, оптимизировать процесс тестирования и снизить затраты на поддержку программного обеспечения.

📎 Принцип №5. Парадокс пестицида — повторяющиеся тесты со временем теряют свою эффективность.

Принцип тестирования, который гласит, что если постоянно проводить одни и те же тесты, то со временем они теряют свою эффективность, потому что обнаруженные ими ошибки исправляются, и тесты перестают выявлять новые дефекты. Этот парадокс получил свое название по аналогии с использованием пестицидов в сельском хозяйстве: если применять один и тот же пестицид на протяжении долгого времени, вредители могут адаптироваться к нему и стать устойчивыми, так что средство теряет свою эффективность.

Когда тесты повторяются из версии в версию, они проверяют уже исправленные ошибки и определенные аспекты системы. В результате они не охватывают новые изменения, которые вносятся в код, и не адаптируются к новым сценариям использования. Повторение одних и тех же тестов даёт уверенность, что старые функции работают как и раньше, но не гарантирует, что новые функции будут работать корректно или что система в целом остается стабильной.

Пример. Предположим, на сайте интернет-магазина есть функционал добавления товаров в корзину. Первоначально тестировщики пишут тесты, которые проверяют основные сценарии: добавление товара в пустую корзину, изменение количества и удаление товара. Эти тесты успешно выявляют начальные ошибки, и разработчики их исправляют.

Спустя несколько месяцев в корзину добавляются новые функции: применение промокодов, работа с бонусными баллами и выбор опций доставки. Однако, если тестировщики продолжают запускать только старые тесты, они не проверяют новые функции. Так как старые тесты не учитывают эти изменения, ошибки в новых функциях могут остаться невыявленными.

Чтобы избежать снижения эффективности тестирования, необходимо постоянно улучшать и обновлять тесты, адаптируя их под новые изменения в системе. Вот несколько подходов для борьбы с парадоксом пестицида:

- **Регулярный анализ и обновление тестов:**
  - Команда тестировщиков должна периодически пересматривать тестовые сценарии, удалять устаревшие и добавлять новые, которые отражают изменения в системе.
  - Например, если в продукт добавили функцию скидок, тестировщики должны создать тесты, которые проверяют все сценарии, связанные с её использованием (применение скидок, исключения и ограничения).

- **Расширение и разнообразие тестов:**
  - Использование различных видов тестирования помогает избежать ситуации, когда тесты слишком однообразны. Например, к функциональным тестам можно добавить тесты на производительность, безопасность, удобство и простоту использования и так далее.
  - Например, если ранее проверялись только функциональные аспекты корзины, то теперь можно добавить нагрузочное тестирование, чтобы увидеть, как она работает при большом количестве товаров.

- **Использование техник тест-дизайна:**
  - Техники тест-дизайна, такие как граничное тестирование, классы эквивалентности, таблицы принятия решений, помогают создавать тесты, которые охватывают новые и необычные случаи, не учитываемые в стандартных сценариях.
  - Например, если корзина изначально проверялась с обычными значениями, можно провести граничное тестирование с максимально и минимально допустимым количеством товаров или ценами.

- **Внедрение автоматизации для регрессионного тестирования:**
  - Автоматизация тестов позволяет легко добавлять и поддерживать новые сценарии, не тратя много ресурсов на повторяющиеся проверки. Она также позволяет запускать большой объем тестов после каждого изменения в коде, что помогает быстрее находить дефекты.
  - Например, если процесс проверки корзины включает 50 тестов, тестировщики могут автоматизировать регрессионные тесты – это набор тестов, направленных на обнаружение дефектов в уже протестированных участках приложения, чтобы они выполнялись автоматически и выявляли любые сбои после обновлений.

- **Анализ результатов тестирования:**
  - Полезно анализировать, какие тесты на самом деле обнаруживают ошибки, а какие — нет. Тесты, которые давно не выявляли дефектов, можно пересмотреть и заменить более актуальными сценариями.
    Например, если тест, проверяющий процесс удаления товаров из корзины, уже давно не выявлял дефектов, команда может сократить частоту его выполнения или заменить его новым тестом, который проверяет изменения в функционале корзины.

📎 **Принцип №6. Тестирование зависит от контекста — подходы к тестированию различаются в зависимости от проекта.**

Принцип "Тестирование зависит от контекста" гласит, что подходы, методы и глубина тестирования должны подбираться в зависимости от типа проекта, его целей, специфики и требований. Тестирование не бывает универсальным — один и тот же подход может оказаться эффективным в одном проекте, но не подойти для другого.

В каждом проекте есть уникальные особенности, которые влияют на выбор стратегии тестирования:

- **Тип продукта**: Например, тестирование интернет-магазина, банковского приложения и игровой платформы будут кардинально отличаться, так как у них разные пользователи, задачи и ожидания по функциональности.
- **Цели тестирования**: Для некоторых проектов приоритетом может быть функциональное тестирование, для других — тестирование безопасности или производительности.
- **Жизненный цикл и срок проекта**: Например, краткосрочные проекты с небольшими релизами потребуют иного подхода к тестированию, чем долгосрочные проекты с крупными релизами.

Виды проектов и подходы к тестированию.

Вид проекта	Особенности	Важные аспекты тестирования
Веб-приложения	Веб-приложения требуют тщательного тестирования на различных устройствах, браузерах и операционных системах, так как они доступны множеству пользователей с разными параметрами.	**Кросс-браузерное тестирование**: проверка корректного отображения и работы сайта на разных браузерах. **Тестирование безопасности**: важность безопасности данных пользователей и предотвращение уязвимостей, таких как SQL-инъекции и XSS-атаки. **Тестирование производительности**: веб-приложения должны выдерживать высокую нагрузку, особенно при высокой посещаемости, поэтому тесты на производительность и нагрузку (load testing) играют значительную роль.
Мобильные приложения	Мобильные приложения требуют тестирования на множестве устройств с разными версиями ОС, разрешениями экрана и аппаратными возможностями. Успех мобильного приложения часто зависит от удобства использования, производительности и потребления ресурсов.	**Тестирование на разных устройствах и операционных системах**: поскольку устройства сильно различаются, тесты должны учитывать разные платформы и их ограничения. **Тестирование юзабилити**: пользователи мобильных приложений ожидают удобный интерфейс, быструю загрузку и интуитивную навигацию. **Тестирование производительности и ресурсов**: важно проверять, как приложение использует батарею и память, чтобы не перегружать устройство.
Финансовые и банковские системы	Финансовые системы должны гарантировать высокую надёжность, безопасность и точность расчётов, так как малейшая ошибка может повлечь за собой значительные	**Тестирование безопасности**: это один из важнейших аспектов, так как банковские данные очень чувствительны к утечкам.

	убытки и утрату доверия пользователей. Тестирование здесь требует особой строгости и глубины.	**Тестирование точности расчётов**: любые ошибки в расчетах могут быть критичны. Поддержка точности и работа с разными валютами должны проверяться особенно тщательно. **Стресс-тестирование**: такие системы должны выдерживать большие нагрузки, особенно в пиковые периоды, поэтому нагрузочные тесты — обязательный элемент.
Игровые приложения	Игровые приложения ориентированы на развлечение и должны работать быстро, плавно и быть удобными для пользователя.	**Тестирование производительности**: игра должна показывать высокую частоту кадров (FPS), особенно в динамичных играх. **Тестирование юзабилити и игровой механики**: пользовательский опыт имеет решающее значение, поэтому интерфейс, управление и взаимодействие с игрой должны быть удобными и интуитивными. **Тестирование графики и анимаций**: игры имеют множество визуальных элементов, поэтому необходимо проверять, что графика работает корректно и на слабых устройствах.
Встроенные и медицинские системы	Эти системы часто используются в критически важных условиях, где сбой может повлечь за собой серьезные последствия, в том числе риски для жизни.	**Тестирование надежности и устойчивости**: Эти системы должны оставаться надёжными даже при непредвиденных условиях (перепады напряжения, сбои в сети и т.д.). **Соответствие стандартам**: Встроенные и медицинские системы часто требуют тестирования на соответствие строгим индустриальным стандартам, которые регламентируют надёжность и безопасность. **Функциональное тестирование**: Проверка точности выполнения задач — критична для работы таких систем, особенно в медицине.

Принцип "Тестирование зависит от контекста" подчеркивает важность выбора правильного подхода к тестированию в зависимости от типа проекта, его специфики и требований. В зависимости от контекста меняются приоритеты и методики тестирования: в одном проекте ключевую роль будут играть тесты на безопасность, в другом — на производительность, в третьем — на юзабилити. Применение этого принципа помогает тестировщикам лучше понимать проект, адаптировать подходы к тестированию и обеспечивать высокое качество программного продукта в зависимости от его контекста и потребностей пользователя.

 **Принцип №7. Заблуждение об отсутствии ошибок — важнее управлять критичностью и исправлять ошибки быстро, а не пытаться исключить все баги.**

Принцип "Заблуждение об отсутствии ошибок" указывает на важность управления критичностью багов и быстрого реагирования на них, а не на достижение полного отсутствия ошибок. Идея заключается в том, что ресурсы команды ограничены, и попытка устранить абсолютно все баги может оказаться нецелесообразной и даже экономически невыгодной. В реальной практике важнее классифицировать ошибки по их влиянию на продукт и сосредотачиваться на исправлении критичных проблем, которые могут повлиять на пользователей или бизнес, чем на попытке сделать продукт абсолютно безошибочным.

Абсолютное устранение всех багов — практически невозможная задача, особенно в сложных и многофункциональных системах. Причин тому несколько:

- **Ограниченные ресурсы**: Команды разработки и тестирования всегда работают с ограниченными ресурсами, такими как время, бюджет и человеческие ресурсы. Попытка исправить все ошибки приводит к перерасходу этих ресурсов, отвлекая их от других приоритетных задач.
- **Уровень критичности багов**: Не все баги одинаково значимы. Многие ошибки могут оставаться в системе, не влияя на пользовательский опыт или бизнес-цели. Исправление мелких ошибок, которые почти не затрагивают пользователей, может не стоить затраченных усилий.
- **Изменяющиеся требования и бизнес-приоритеты**: В процессе разработки и эксплуатации требования могут меняться. Некоторые ошибки могут потерять свою актуальность, если функционал, к которому они относятся, меняется или уходит в приоритет.

Пример.

Представим интернет-магазин, где тестировщики нашли несколько багов перед выпуском новой версии:

- **Критичный баг**: Оформление заказа не работает при оплате банковской картой. Этот баг нужно срочно исправить, так как он напрямую влияет на продажи и на прибыль компании.
- **Высокоприоритетный баг**: Ошибка в работе фильтров товаров по цене. Пользователи могут испытывать неудобства при поиске товаров, но они всё равно могут совершить покупку. Этот баг важен, но его можно исправить немного позже.
- **Среднеприоритетный баг**: Некорректное отображение значков категорий на странице каталога в старом браузере. Так как большинство пользователей используют более современные браузеры, исправление этой ошибки не является критичным.
- **Низкоприоритетный баг**: В подвале сайта есть орфографическая ошибка в информации о компании. Этот баг не влияет на функциональность сайта и может быть исправлен в следующем релизе.

Здесь критичный и высокоприоритетный баги необходимо исправить как можно быстрее, чтобы избежать недовольства пользователей и потерь для бизнеса. Средне- и низкоприоритетные баги могут быть исправлены позже или даже остаться в системе, пока на них не выделят ресурсы.

Принцип "Заблуждение об отсутствии ошибок" подчеркивает, что цель тестирования — не полное устранение всех багов, а поддержание качества продукта через управление критичностью ошибок и оперативное реагирование на важные проблемы. Исправление всех багов невозможно и, зачастую, экономически нецелесообразно. Поэтому важнее определить критичные для бизнеса ошибки и приоритизировать их, распределяя ресурсы рационально и направляя усилия на те аспекты, которые влияют на пользователей и бизнес.

Основные виды тестирования

По отношению к функциональности ПО выделяют 2 вида тестирования: нефункциональное и функциональное.

Нефункциональное тестирование - насколько эффективно, надежно и удобно работает система в различных условиях эксплуатации?

Функциональное тестирование - выполняет ли система свои функции корректно в соответствии с требованиями и спецификациями?

Каждый из этих видов тестирования подразделяется еще на несколько типов. Общую классификацию мы можем видеть в таблицах ниже.

В дальнейшем мы подробнее остановимся на каждом из них.

Нефункциональное тестирование		
Тестирование производительности • Нагрузочное тестирование • Стресс-тестирование • Тестирование стабильности	**Инсталляционное тестирование**	**Usability тестирование**
Тестирование локализации и интернационализации	**Конфигурационное тестирование**	**Тестирование безопасности**
Тестирование на отказ и восстановление	**Тестирование доступности**	**Тестирование документации**

Функциональное тестирование			
По уровню тестирования • Модульное • Интеграционное • Системное • Приемочное	**Связанное с изменениями** • Smoke-тестирование • Регрессионное • Sanity-тестирование	**По знанию системы** • Black box • White box • Grey box	**По критерию запуска программы** • Статическое • Динамическое
Относительно позитивности сценариев • Позитивные тесты • Негативные тесты	**По типу интерфейса** • API • CLI • GUI	**По степени автоматизации** • Ручное • Автоматизи-рованное	**По исполнению сценария** • Сценарное • Исследователь-ское • Ad-hoc

Нефункциональное тестирование - насколько эффективно, надежно и удобно работает система в различных условиях эксплуатации?

1. **Тестирование производительности**: включает нагрузочное тестирование, стресс-тестирование и тестирование стабильности. Насколько эффективно и стабильно система работает под различными условиями нагрузки.

Название и описание	Пример
Нагрузочное тестирование (Load Testing) Нагрузочное тестирование проверяет, как система работает при нормальных и увеличенных уровнях нагрузки. Его цель — убедиться, что система остается производительной и работает без ошибок при ожидаемом количестве пользователей или запросов.	Интернет-магазин ожидает, что во время распродажи его посетит 10,000 пользователей в час, и хочет убедиться, что сайт сможет выдержать такую нагрузку. Во время нагрузочного тестирования симулируется 10,000 одновременных пользователей, которые добавляют товары в корзину, оформляют заказы и просматривают каталоги. Тестирование показывает, насколько быстро загружаются страницы и как стабильно работает процесс покупки. Если время ответа сервера остается в пределах нормы, тест считается успешным.

Стресс-тестирование (Stress Testing) Стресс-тестирование проверяет, как система работает в условиях экстремальной или пиковой нагрузки, превышающей ожидаемые уровни. Его цель — выявить, где находятся пределы системы, и понять, как она справляется с перегрузками. Это тестирование помогает определить точки отказа и проверить, как система восстанавливается после сбоев.	В социальной сети проводятся стресс-тесты, чтобы проверить ее устойчивость при резком увеличении числа пользователей. В тестировании симулируется нагрузка в 1,5-2 раза больше обычной, например, 50,000 одновременных запросов вместо 20,000. На платформе имитируются сценарии, такие как массовые публикации, отправка сообщений и загрузка видео. Цель — проверить, как система справляется с этим и не приводит ли перегрузка к серьёзным сбоям. Если сайт замедляется, но не «падает» полностью и восстанавливает нормальную работу после снижения нагрузки, тест можно считать успешным.
Тестирование стабильности (Stability Testing) Тестирование стабильности, или тестирование на долговременную нагрузку (Soak Testing), проверяет, насколько стабильно работает система при длительной нагрузке. Цель — убедиться, что система сохраняет производительность, не имеет утечек памяти и не ухудшает время отклика при долгосрочной эксплуатации.	Платформа для видеоконференций должна выдерживать стабильную нагрузку в течение рабочего дня (8-10 часов) при постоянном использовании. В тестировании моделируется сценарий, когда тысячи пользователей находятся в видеозвонках, обмениваются сообщениями и делятся экранами. Тест продолжается в течение нескольких часов без перерывов. Специалисты наблюдают за потреблением памяти, временем отклика и другими метриками. Если система сохраняет стабильность, тест пройден успешно. Если появляются утечки памяти или увеличивается время ответа, это указывает на проблемы, которые следует устранить.

2. Инсталляционное тестирование: проверка процесса установки ПО.

Проверяет, корректно ли устанавливается программное обеспечение (ПО) на целевое устройство или систему и работает ли оно правильно после установки. Цель этого тестирования — убедиться, что установка ПО проходит без ошибок, все компоненты инсталлируются, а система готова к использованию.

Основные задачи инсталляционного тестирования:

- **Проверка процесса установки:** Убедиться, что ПО устанавливается корректно, без ошибок и проблем.
- **Проверка зависимостей:** Проверить, что все необходимые компоненты и зависимости инсталлируются (например, драйверы, библиотеки и настройки).
- **Тестирование сценариев установки:** Проверка различных сценариев установки, таких как полная установка, выборочная установка и обновление существующей версии.
- **Проверка удаления:** Убедиться, что удаление ПО происходит корректно, все файлы и записи реестра удаляются и не остаётся ненужных данных.

3. Usability: тестирование удобства использования.

Вид тестирования, направленный на оценку того, насколько удобно, интуитивно и эффективно пользователи могут взаимодействовать с программным продуктом. Основная цель usability-тестирования — улучшить пользовательский опыт, выявить проблемные места интерфейса и устранить элементы, которые могут вызвать затруднения у пользователей.

Примеры usability-тестирования:

Пример 1. Интернет-магазин.

Usability-тестирование интернет-магазина может включать проверку процесса покупки. Пользователи выполняют задачи, такие как поиск товара, добавление его в корзину и оформление заказа. Тестировщики наблюдают, насколько легко пользователи находят нужные элементы (кнопка "Купить", фильтры по категориям) и могут ли завершить процесс покупки без затруднений. Если пользователи не могут найти нужные кнопки или запутываются в интерфейсе, это указывает на необходимость улучшения.

Пример 2. Мобильное приложение для бронирования.

Usability-тестирование приложения для бронирования гостиниц проверяет, насколько удобно пользователям выбирать даты, вводить данные о поездке и просматривать результаты поиска. Например, если пользователи путаются при

выборе даты или не понимают, как изменить фильтры поиска, это выявляет потенциальные проблемы с удобством. Также можно протестировать, сколько шагов требуется для завершения бронирования и насколько легко пройти процесс от начала до конца.

Пример 3: Корпоративный портал.

В корпоративном портале usability-тестирование помогает оценить, насколько легко сотрудники могут выполнять свои задачи, например, искать документы, оставлять заявки на отпуск или взаимодействовать с коллегами. В ходе тестирования проверяются такие аспекты, как доступность важных функций на главной странице и простота навигации. Если пользователи испытывают сложности при переходе по разделам или поиске информации, это говорит о необходимости улучшения структуры портала.

4. **Локализация и интернационализация: проверка адаптации продукта для разных регионов.**

Интернационализация — это процесс подготовки продукта к поддержке различных языков и культурных особенностей без необходимости переписывать код. Интернационализация предусматривает изначальное проектирование продукта так, чтобы его можно было легко адаптировать для различных регионов.

Локализация — это процесс адаптации интерфейса, контента и функциональности продукта для конкретного региона или языка. Включает перевод текста, адаптацию форматов даты, времени, валюты, изменение изображений и даже корректировку цветовой палитры, чтобы продукт был удобен и понятен для пользователей из конкретного региона.

Основные задачи локализации и интернационализации:

- **Проверка корректности перевода**: Убедиться, что перевод точен и звучит естественно для носителей языка, а не буквально.
- **Адаптация форматов даты, времени и чисел**: Убедиться, что в интерфейсе используются форматы, привычные для данного региона (например, 01.12.2024 в европейском формате и 12/01/2024 в американском).

- **Проверка локализованного контента**: Адаптировать изображения, символы, культурные и религиозные элементы для конкретной страны или региона.
- **Обеспечение поддержки многоязычного интерфейса**: Убедиться, что продукт корректно отображает разные языки, включая языки с другой системой письма (например, арабский или китайский).
- **Проверка удобства навигации в локализованном интерфейсе**: Убедиться, что перевод и адаптация не мешают восприятию и пользователи легко могут ориентироваться.

5. Конфигурационное: тестирование на разных конфигурациях оборудования и ПО.

Тип тестирования, направленный на проверку того, как продукт работает в различных комбинациях оборудования, операционных систем, браузеров и других компонентов программного обеспечения. Его цель — убедиться, что продукт работает стабильно и корректно в разнообразных условиях эксплуатации, обеспечивая хорошее качество пользовательского опыта независимо от конфигурации.

Примеры конфигурационного тестирования:

Пример 1. Веб-приложение.

Для веб-приложения конфигурационное тестирование включает проверку работы на различных браузерах и их версиях (например, Chrome 100, Firefox 95, Safari 15), а также на разных операционных системах (Windows, macOS, Linux). Тестировщики проверяют, что приложение корректно отображается и работает в каждом из этих браузеров, что кнопки, формы и другие элементы интерфейса функционируют должным образом.

Пример 2. Мобильное приложение.

Для мобильного приложения конфигурационное тестирование предполагает проверку на различных устройствах с разными версиями операционных систем, например, на Android и iOS. Тестирование проводится на устройствах с разными характеристиками экрана, мощностью процессора и объемом памяти, чтобы

убедиться, что приложение работает стабильно, не вызывает перегрузку устройства и корректно отображает интерфейс на экранах разных размеров.

6. Тестирование безопасности: проверка системы на уязвимости.

Тип тестирования, направленный на выявление уязвимостей системы и обеспечение ее защиты от внешних и внутренних угроз. Основная цель тестирования безопасности — убедиться, что данные пользователей защищены от утечек и несанкционированного доступа, а система устойчива к потенциальным атакам.

Примеры тестирования безопасности:

Пример 1. Веб-приложение.

Для веб-приложения тестирование безопасности может включать проверку на SQL-инъекции, межсайтовый скриптинг (XSS) и уязвимости в процессе аутентификации. Тестировщики проверяют, что злоумышленники не могут внедрить вредоносный код в формы ввода, получить доступ к базе данных через SQL-запросы или обойти защиту, используя уязвимости в системе входа.

Пример 2. Интернет-банкинг.

В системе интернет-банкинга тестирование безопасности может включать проверку механизмов шифрования данных и двухфакторной аутентификации. Тестировщики проверяют, что данные транзакций надежно шифруются, а доступ к аккаунту возможен только с подтверждением через дополнительный код. Также проверяется устойчивость к атакам методом грубой силы (хакерская техника, которая заключается в многократном переборе различных комбинаций паролей или ключей шифрования до тех пор, пока не будет найдена правильная, часто с использованием автоматики) и возможность блокировки при нескольких неудачных попытках входа.

Пример 3. Корпоративная система.

Для корпоративной системы тестирование безопасности включает проверку разделения прав доступа для разных категорий пользователей, например, сотрудников, менеджеров и администраторов. Тестировщики убеждаются, что

пользователи не могут получить доступ к информации, к которой у них нет прав, и что при попытке обойти эти права система регистрирует инцидент и уведомляет администратора.

7. **Тестирование на отказ и восстановление: проверка способности системы восстанавливаться после сбоев.**

Примеры тестирования на отказ и восстановление:

Пример 1. Веб-сервис с базой данных.

Тестирование на отказ и восстановление для веб-сервиса с базой данных может включать симуляцию отключения доступа к базе данных. Тестировщики проверяют, как система реагирует на недоступность базы данных, и тестируют процессы восстановления после восстановления соединения. Также проверяется, что данные, введенные до сбоя, сохраняются корректно.

Пример 2. Онлайн-банкинг.

Для системы онлайн-банкинга тестирование включает проверку реакции на сбои в сети и отключение серверов. Тестировщики могут проверить, что происходит, если транзакция прерывается из-за сетевого сбоя, и как быстро система восстанавливает доступ, а также восстанавливаются ли данные о транзакциях. В случае прерванной операции система должна вернуть пользователя в корректное состояние и уведомить его об ошибке.

Пример 3. Облачное хранилище.

Для облачного хранилища тестирование на отказ и восстановление может включать проверку резервного копирования данных и их восстановления после сбоя. Тестировщики могут симулировать сбой в одном из серверов и проверить, что система переключается на резервный сервер без потери данных. Также проверяется, что после полного восстановления серверов данные остаются в целостности и доступны пользователю.

8. **Тестирование доступности: проверка, насколько продукт доступен для пользователей с ограниченными возможностями.**

Примеры тестирования доступности:

Пример 1. Видеоплатформа.

На видео платформе, такой как YouTube, тестирование доступности может включать проверку наличия субтитров к видеороликам и текстовых описаний для пользователей с нарушениями слуха. Также тестируется возможность управления плеером с помощью клавиатуры, чтобы обеспечить доступность для пользователей с моторными ограничениями, которые не могут пользоваться мышью.

Пример 2. Мобильное банковское приложение.

Для мобильного банковского приложения тестирование доступности может включать проверку адаптации интерфейса для пользователей с нарушениями зрения, таких как поддержка увеличения шрифтов, использование цветовых схем с высокой контрастностью и голосовые подсказки для навигации. Тестировщики также проверяют, что все кнопки и элементы управления доступны для пользователей, которые используют специальные устройства или только клавиатуру для навигации.

9. **Тестирование документации: проверка правильности и полноты документации.**

Это вид тестирования, направленный на проверку правильности, полноты и удобства использования документации, которая сопровождает программное обеспечение. Цель тестирования документации — убедиться, что вся информация в документации актуальна, соответствует функциональности продукта и предоставляет пользователям или разработчикам ясное руководство по использованию и поддержке системы.

Основные задачи тестирования документации:

- **Проверка соответствия документации функциональности продукта:** Убедиться, что описанные функции и возможности полностью соответствуют реальным возможностям системы, и что документация обновлена в соответствии с последними изменениями продукта.
- **Оценка полноты документации:** Проверить, что вся необходимая информация присутствует — от инструкций по установке и настройке до описания всех ключевых функций и возможных ошибок.
- **Проверка точности и понятности:** Убедиться, что вся информация изложена точно, ясно и доступным языком, чтобы пользователи или технические специалисты могли легко понимать инструкции и следовать им.
- **Проверка структуры и навигации:** Убедиться, что документация имеет логичную структуру и что пользователи могут легко находить нужную информацию, используя оглавление, поисковые функции и перекрестные ссылки.
- **Проверка на наличие опечаток и ошибок:** Проверить, что документация не содержит орфографических, грамматических или терминологических ошибок, которые могут ввести пользователей в заблуждение.

Функциональное тестирование - выполняет ли система свои функции корректно в соответствии с требованиями и спецификациями?

1. **По уровню тестирования:**
   - **Модульное** (component testing): проверка отдельных модулей.
   - **Интеграционное**: тестирование взаимодействия модулей.
   - **Системное**: проверка всей системы в целом.
   - **Приемочное**: проверка на соответствие требованиям перед выпуском продукта.

На примере тестирования автомобиля:

- **Модульное тестирование**: На этапе модульного тестирования автомобиля инженеры проверяют работу отдельных компонентов, таких как двигатель, тормозная система, фары или система кондиционирования. Например, они могут отдельно тестировать двигатель, чтобы убедиться, что он работает корректно и соответствует требованиям по мощности и расходу топлива.

- **Интеграционное тестирование**: На этом этапе проверяется взаимодействие между модулями автомобиля. Например, инженеры тестируют, как работает тормозная система в связке с системой ABS, или проверяют, как взаимодействуют двигатель и трансмиссия, чтобы убедиться, что их работа синхронизирована и сбои одного компонента не влияют на работу другого.
- **Системное тестирование**: На этапе системного тестирования проверяется автомобиль как единая система. Инженеры оценивают, как все модули работают вместе в реальных условиях — например, тестируют автомобиль на дороге, проверяя ускорение, торможение, маневренность и общую безопасность, чтобы убедиться, что машина готова к использованию.
- **Приемочное тестирование**: Это финальный этап, где проверяется соответствие автомобиля всем установленным требованиям и стандартам перед его выпуском на рынок. На этом этапе специалисты и представители заказчика или сертификационных органов оценивают, что автомобиль безопасен, надежен и соответствует всем нормативным требованиям и стандартам, чтобы его можно было передать конечным пользователям.

2. **Связанное с изменениями:**
   - **Smoke-тестирование**: проверка базовой работоспособности системы после изменений.
   - **Регрессионное тестирование**: проверка, не вызвали ли изменения новых ошибок.
   - **Sanity-тестирование**: узконаправленная проверка работы конкретной функции.

На примере тестирования автомобиля:

- **Smoke-тестирование**: Smoke-тестирование автомобиля проводится после сборки или крупных изменений, чтобы убедиться в его базовой работоспособности. Например, проверяется, что двигатель запускается, тормоза работают, световые индикаторы функционируют и автомобиль может двигаться вперед и назад. Это базовая проверка, чтобы убедиться, что автомобиль готов к дальнейшим, более детальным тестам.
- **Регрессионное тестирование**: Регрессионное тестирование автомобиля проводится после внесения изменений, чтобы убедиться, что новые обновления или исправления не привели к появлению новых проблем. Например, если была обновлена электронная система управления двигателем, регрессионное тестирование проверяет не только её, но и

работу всех связанных функций, таких как подача топлива и торможение, чтобы убедиться, что другие системы работают корректно.

- **Sanity-тестирование**: Sanity-тестирование применяется, когда нужно убедиться в работоспособности конкретной функции после внесения точечного изменения. Например, если инженеры заменили сенсор давления в шинах, sanity-тестирование будет включать проверку работы только этого сенсора и отображения данных на панели приборов, чтобы убедиться, что изменение не привело к ошибкам именно в этой функции.

3. **В зависимости от степени доступа к коду (по знанию системы):**
   - **Black box**: тестирование без знания внутреннего устройства системы (нет доступа коду).
   - **White box**: тестирование с анализом внутренней структуры системы (есть доступ к коду).
   - **Grey box**: комбинированный подход (нет полного доступа к коду, но есть возможность читать логи).

4. **По критерию запуска программы**:
   - **Статическое тестирование**: проверка кода без его выполнения, включая ревью и анализ кода.
   - **Динамическое тестирование**: проверка программы в действии с выполнением кода.

5. **Относительно позитивности сценариев**:
   - **Позитивные тесты**: тестирование с корректными входными данными.
   - **Негативные тесты**: тестирование с некорректными данными, чтобы проверить устойчивость программы.

В случае негативного теста мы используем на входе невалидные данные и ожидаем на выходе ошибку. Отображение ошибки в таком случае - ожидаемое поведение системы.

6. **По типу интерфейса**:
    - **API-тестирование**: проверка взаимодействия через программные интерфейсы.
    - **CLI-тестирование**: проверка через командную строку.
    - **GUI-тестирование**: тестирование графического интерфейса пользователя.

7. **По степени автоматизации**:
    - **Ручное тестирование**: выполнение тестов вручную.
    - **Автоматизированное тестирование**: выполнение тестов с помощью программных инструментов.

8. **По исполнению сценария**:
    - **Сценарное тестирование**: тестирование по заранее разработанным сценариям.
    - **Исследовательское тестирование**: выполнение тестов без заранее определенных сценариев, в поиске неожиданных дефектов.
    - **Ad-hoc тестирование**: спонтанное, неформальное тестирование без подготовки.

IT CAREER HUB

Полезные материалы

1. Что такое качество
2. Жизненный цикл тестирования приложений