**Классификация тестирования.**

1. **По уровням**

1.1. 1 уровень:

1.1.1. Компонентное тестирование (разработка кода)

Представляет собой тестирование отдельных каждого объекта или частей ПО (например, сайт интернет-магазина: отдельно тестирование корзины, оплаты заказа)

1.1.2. Модульное тестирование/Unit testing

Тестирование отдельных классов/функций для демонстрации того, что программа выполняется согласно спецификации.

Unit tests это не модульное. Это когда разработчики пишут тесты на свой код (оч классно знают свой код). Тестирование модуля и в принципе кода. Юнит тесты на каждую функцию

\_\_\_\_\_

1.2. 2 уровень

1.2.1. Интеграционное тестирование.

Проверка взаимодействия модулей системы. Тестинг части системы, состоящий из 2 и более модулей (например, тестирование того как со страницы корзины на сайте интернет-магазина мы можем произвести оплату посредством платежной системы).

Интеграционное тестирование может состоять из:

1.2.1.1. Тестирование интеграций компонентов

Определяет как отдельные модули одного приложения взаимодействуют между собой.

**CP + Smoke**

1.2.1.2. Системное интеграционное тестирование

Определяет тестирование взаимодействия между **всеми** компонентами системы/ взаимодействие разных систем между собой/ тестирование интерфейсов с помощью которых взаимодействует система.

Интерфейсы

1. Интерфейс програмирования приложения (API)

Набор методов, которые можно использовать для доступа к функциональности другой программы.

Например для проверки платежной системы.

1. Интерфейс командной строки (SLI)

Обычная командная строка в системе Windows с помощью которой выдается инструкция к компьютеру путем ввода текстовой строки.

1. GUI (graphical user interface)

Программные функции в виде графических элементов экрана. (“То, что мы видим в окне браузера, когда открываем страницу в Интернете)

\_\_\_\_

1.3. 3 уровень

1.3.1. Системное тестирование.

Полная проверка приложения. Выполняется на полностью интегрированной системе. С целью проверки соответствия системы исходным требованиям.

Как в целом все модули интернет-магазина например работают между собой на уровне всего приложения целиком.

**E2E**

\_\_\_

1.4. 4 уровень

1.4.1. Приемочное тестирование

Вид тестирования, проводимый на этапе сдачи готового продукта заказчику.

Цель: Определение готовности продукта путем прохода тестовых сценариев на основе спеки и требований к нашему продукту.

Финальный этап тестирования продукта перед его релизом.

Виды Приемочного тестирования:

1.4.1.1. Пользовательское приемочное тестирование (UAT) User Acceptance Testing

Проводится пользователем конечного продукта. Например, при сборе группы конечных пользователей (которые будут тестировать).

При таком тестирование данная группа пользователей отвечает соответствует ли продукт их ожиданиям. Затем они дают заключение.

Команда разработчиков данного продукта фиксит баги и проводит релиз продукта.

1.4.1.2. Эксплуатационное Приемочное тестирование.

Выполняется пользователем или администратором в среде, которая имитирует реальные условия рабочего окружения. Проводится:

* Проверка тестирования резервного копирования
* Проверка аварийности восстановления системы и безопасность.

1.4.1.3. Тестирование на соответствие контракту

Соответствует ли система спеке или нормативному акту. ГОСТ того, как наш продукт должен соответствовать нормативному акту.

GDRP - требование к защите пользовательских данных (Sensitive date)

1.4.1.4. Альфа-тестирование

Эксплуатационное тестирование на стороне разработчика, но не из нашей компании.

1.4.1.5. Бэта-тестирование

Производится кем-то на внешней стороне и без участия разработчика.

Например, небольшой группой пользователей, чтобы получить отзыв до выхода продукта.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**2. По позитивности тестовых сценариев**

2.1. Позитивное тестирование.

Применяются сценарии, которые соответствуют нормальному ожидаемому поведению системы (2+2=4)

2.2. Негативное тестирование.

Применяется сценарий, который соответствует нештатному поведению системы

В калькуляторе: проверить, что будет если поделить на ноль. Должна быть ошибка.

Сначала выполняются Позитивные тестовые сценарии (особенно в режиме ограниченного времени).

Сначала проверка основных положительных функций и уже потом поиск (проверка чего-то негативного).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**3. По степени важности тестируемых функций**

3.1. Smoke-testing (дымовое тестирование)

Проверка работоспособности основных функций приложения (продукта).

Производится на начальном этапе после нового билда.

* Проверка готовности продукта на расширенное тестирование и проведение общего состояния качества продукта.
* Подтверждает или опровергает, что приложение стартует и выполняет свои основные функции.

Данное тестирование должно быть легким и быстрым, чтобы часто запускать.

\_\_\_

3.2. Тестирование Критического пути

“Действия типичного пользователя в типичной жизни выполняющие типичные задания. Проверяется основная масса требований к продукту:

* выбор шрифта
* возможность набора текста
* вставка картинок

Тестирование критического пути может быть:

* Позитивное
* Негативное

\_\_\_\_

3.3. Расширенный тест

Проверка нестандартных использований ПО:

* Ввод специальных символов в поле логина
* нелогично кликать по кнопкам
* работа на многих вкладках

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Виды тестирования, связанные с изменениями: Smoke, Sanity (при появлении новой фичи, санити отвечает за углубленный тест этой фичи), Re-test, Regression, Приемочное (иногда)

1. **По цели тестирования:**

4.1. Тестирование новой функциональности (New Feature Test)

Проверка качества новой функциональности, поставленной на тестирование

Состоит из Smoke testing, Тест критического пути, Расширенный тест.

\_\_\_

4.2. Регрессионное тестирование (Regression test)

Повлиял ли например New Feature testing на основной модуль

Тестирование ранее проверенной функциональности с целью удостоверения того, что изменения в коде (изменение или добавление) не повлияло на работу старой (прежней) функциональности.

Проверка по тому же чек-лист и/или тест-кейсу после внесенных изменений.

* Обязательно проводится в каждом новом билде
* Проверка исправленных багов и повлияли ли они на другую функциональность (в которой не было багов)
* Покрывает те участки, которые соприкасаются с изменениями в билде.
* Тестирование данное рекомендуется проводить несколько раз.

Регрессионное тестирование АВТОМАТИЗИРУЮТ. Тк предполагает рутинное повторение тех тестов, которые мы уже проводили ранее.

Выборка

1. Выбираются тесты, которые покрывают тестирование безопасности
2. Области которые часто меняются
3. Тесты функций с высокой вероятностью ошибки.

\_\_\_\_

4.3. Re-test

Проверка правильности исправления дефекта

Разработчик кидает пофикшенный баг и тестировщик убеждается в том что данный баг пофиксен.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**5. По степени автоматизации**

5.1. Ручное тестирование

5.2. Автоматизированное тестирование

* Используется специальное ПО
* Автоматизируют часто повторяющиеся кейсы по необходимости для маскимизации тестового покрытия задачи.

Объекты автоматизации:

* Регрессионное тестирование (рутинное повторение тех тестов которые мы проводили ранее с проверкой функционала в котором не имелось багов)

Таким образом автоматизировать нужно когда:

* есть куски функционала, которые уже не меняются. И их проверку нужно автоматизировать.
* Стабильно работает код который уже не изменяется.

**Пишется отдельное приложение, которое тестирует мое приложение. Оно не относится к белому ящику!**

**И по сути такое приложение не знает про код тестируемого продукта, а видит только уже развернутую прилагу. Поэтому автоматизированное тестирование относится к тестированию Серого ящика!**

**Автоматизирование бэка = серый ящик**

**Автоматизирование фронта = черный ящик!**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**6. По знанию кода**

6.1. Тестирование черного ящика (Black Box test)

Неизвестно, как устроена тестируемая система. Данное тестирование основано исключительно на работе с внешними интерфейсом тестируемой системы.

В данном тестировании код совсем неизвестен.

* Тестировщик тестирует веб-сайт не зная его реализации и сути. Только на основе информации интерфейса( размер поля, кнопок и тд)
* Источник ожидаемого результата = Спецификация.
* Неинтересно как работает изнутри

**Тупо тестируем GUI (user interface)**

**Автоматизирование бэка = серый ящик**

**Автоматизирование фронта = черный ящик!**

Черным ящиком не только фронт, но и документация и дизайн сайта

6.2. Тестирование белого ящика (White Box testing)

* Известны все детали реализации тестируемой программы
* Внутренняя структура и устройство ПО известны тестировщику
* Сравнение фактического результата (код ПО) с ожидаемым - как должен работать код программы (совпадает ли ожидаемый результат с фактическим)

***Напрямую тестируем код. Тестирование функций с прогоном на своих собственных тестах. Когда разрабы сами тестируют свой же код перед тем как его мержить.***

6.3. Тестирование серого ящика (Grey Box Testing)

**API**

**Автоматизированное тестирование (т.к. пишется отдельное приложение, которое тестирует мое приложение. Оно не относится к белому ящику!**

**И по сути такое приложение не знает про код тестируемого продукта, а видит только уже развернутую прилагу).**

**Автоматизирование бэка = серый ящик**

**Автоматизирование фронта = черный ящик!**

* Известны только некоторые особенности тестируемой системы
* Внутреннее устройство ПО (код) известно частично

Известна внутренняя структура и алгоритм частично, но только для написания тест-кейса.

Само тестирование проводится по технике Черного ящика!

В консоли разработчика можем посмотреть код.

6.4. Сквозное тестирование (E2E)

Я как пользователь захожу в систему что-то там делаю и выхожу из системы (от начала и до конца). И мое приложение при всех этих пользовательских кейсах должно отрабатывать корректно. Например, букинг самоката (мобильное приложение):  
Я открываю приложение, после чего выбираю самокат, откроется его страница, я выбрала самокат, но еще не забукала и выключила приложение. По сути ничем не отличается от СР (мы просто тестируем основные бизнес-сценарии пользователя)

Процесс тестирования при котором происходит подробная симуляция пользовательской среды

* Щелчки мышью
* Заполнение форм
* Нажатие на кнопки

“E2E тестирование – это не просто приемка, пользовательское тестирование, которое будет выполнять заказчик, это выстраивание мостика, с учётом всех возможных ситуаций, по которому пойдет заказчик и поведёт за собой в ногу пользователей.”

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**7.Функциональные виды тестирования:**

7.1. Функциональное тестирование (Functional testing)

Рассматривает заранее указанное поведение и основывается на анализе спецификаций функциональности компонента или системы в целом. Одно из видов тестирования, направленного на проверку соответствия функциональным требованиям ПО к его реальным характеристикам.

* Проверка того, что наша система делает.

Задача:

- Подтверждение того, что наш разработанный продукт обладает всем функционалом, который требует заказчик

* Проводится на всех уровнях

7.2. Тестирование пользовательского интерфейса (GUI Testing)

функциональная проверка интерфейса на соответствие требованиям — размер, шрифт, цвет, consistent behavior.

7.3. Тестирование безопасности (Security and Access Control Testing)

это стратегия тестирования, используемая для проверки безопасности системы, а также для анализа рисков, связанных с обеспечением целостного подхода к защите приложения, атак хакеров, вирусов, несанкционированного доступа к конфиденциальным данным

**“Тестирование безопасности онлайн банка - будет функциональным (отображение данных, возможность перехватить данные).**

***Безопасность - только функциональное!!!!***

7.4.Тестирование взаимодействия (Interoperability Testing)

это функциональное тестирование, проверяющее способность приложения взаимодействовать с одним и более компонентами или системами и включающее в себя тестирование совместимости (compatibility testing) и интеграционное тестирование

**8. Нефункциональные виды тестирования**

Проверка соответствия свойств приложения к его нефункциональным требованиям.

8.1. Тестирования производительности:

Проверка работоспособности системы под разными нагрузками.

Комплекс типов тестирования целью которых является определение работоспособности, стабильности, потребления ресурсов и др. атрибутов качества приложения в условиях различных сценариев и нагрузок.

8.1.1. Нагрузочное тестирование (Performance and Load Testing)

автоматизированное тестирование, имитирующее работу определенного количества бизнес пользователей на каком-либо общем (разделяемом ими) ресурсе.

проверка производительности при норм. условиях.

Требование = норма 1000 пользователей

Создаются условия в меньше чем 1000 пользователей.

8.1.2. стрессовое тестирование (Stress Testing)

позволяет проверить насколько приложение и система в целом работоспособны в условиях стресса и также оценить способность системы к регенерации, т.е. к возвращению к нормальному состоянию после прекращения воздействия стресса. Стрессом в данном контексте может быть повышение интенсивности выполнения операций до очень высоких значений или аварийное изменение конфигурации сервера. Также одной из задач при стрессовом тестировании может быть оценка деградации производительности, таким образом цели стрессового тестирования могут пересекаться с целями тестирования производительности.

“проверка работоспособности системы при экстримальных нагрузках. 1000 пользователей и больше”.

8.1.3. тестирование стабильности или надежности (Stability / Reliability Testing)

Задачей тестирования стабильности (надежности) является проверка работоспособности приложения при длительном (многочасовом) тестировании со средним уровнем нагрузки.

8.1.4. Объемное тестирование (Volume Testing)

Задачей объемного тестирования является получение оценки производительности при увеличении объемов данных в базе данных приложения

Например 1000 пользователй отправляет n-нное количество МБ обработанной информации.

8.2. Тестирование установки (Installation testing)

направленно на проверку успешной инсталляции и настройки, а также обновления или удаления программного обеспечения.

8.3. Тестирование удобства пользования (Usability Testing)

метод тестирования, направленный на установление степени удобства использования, обучаемости, понятности и привлекательности для пользователей разрабатываемого продукта в контексте заданных условий.

Сюда также входит:

User eXperience (UX) — ощущение, испытываемое пользователем во время использования цифрового продукта,

UX = насколько быстро человек дойдет до цели (например, снятие денег со счета, каким путем можно ускорить этот процесс, если он долговатый) в то время как

User interface — это инструмент, позволяющий осуществлять интеракцию «пользователь — веб-ресурс». (UI = комфортный/приятный внешний вид)

8.4. Тестирование на отказ и восстановление (Failover and Recovery Testing)

проверяет тестируемый продукт с точки зрения способности противостоять и успешно восстанавливаться после возможных сбоев, возникших в связи с ошибками программного обеспечения, отказами оборудования или проблемами связи (например, отказ сети). Целью данного вида тестирования является проверка систем восстановления (или дублирующих основной функционал систем), которые, в случае возникновения сбоев, обеспечат сохранность и целостность данных тестируемого продукта.

8.5. Партируемость (Переносимость нашего приложения на разные платформы):

*8.5.1. Конфигурационное тестирование (Configuration Testing)*

специальный вид тестирования, направленный на проверку работы программного обеспечения при различных конфигурациях системы (заявленных платформах, поддерживаемых драйверах, при различных конфигурациях компьютеров и т.д.):

8.5.1.1. Кросс-платформенное тестирование (платформы кросс-браузерного тестирования):

Тестирование продукта на разных типах и версиях ОС (Android, IOS)

8.5.1.2. Кросс-браузерное:

Использование нашего приложения в разных браузерах (Chrome, Opera)

*8.5.2. Тестирование установки*

Успешность установки приложения, его настройки и удаления.

8.6. Тестирование локализации (l10n)

Процесс адаптации нашего продукта к языку и культуре клиента. Перевод языка продукта на англоязычном рынке (например) т.е. насколько качественен перевод нашего приложения.

* Контроль формата даты и времени
* Единицы измерения
* Контроль символов и цветов

***Функциональное*** м.б. если мы работаем с иностранными локалями (сайт) если я переключаю на сайте языки и из правильного файла подтягиваются правильные языки (правильный перевод). Каждое слово в каждом месте д.б. переведено (на сколько вся UI в принципе переведена, все ли элементы).

***Нефункциональное тестирование локализации*** = проверка именно правильности перевода (насколько верен перевод)

**Поменялась валюта на нужную**

**Законодательно соответствует какой-то стране**

**Метры/футы (т.е. единицы измерения)**

8.7. Тестирование Интернационализации

На сколько хорошо наш продукт может в дальнейшем адаптироваться для той или иной локализации.

* чтение справа налево в арабских странах
* в будущем иметь ввиду, что приложение может например выйти на азиатские рынки

8.8. Тестирование пользовательского интерфейса (GUI Testing)

функциональная проверка интерфейса на соответствие требованиям — размер, шрифт, цвет, consistent behavior. (Функциональное или НЕФУНКЦИОНАЛЬНОЕ???)

9. Тестирование доступности

Например доступ для людей с ограниченными возможностями

**CRUD = Create, Read, Update, Delete**