1. **что такое тестирование**

Это процесс исследования ПО с целью получения информации о качестве продукта. Так же это проверка соответствия между реальным поведением программы и её ожидаемым поведением.

Тестирование – процесс, связанный с планированием, подготовкой и оценкой программных продуктов и связанных с ними рабочих продуктов, чтобы определить, удовлетворяют ли они определенным требованиям, продемонстрировать, что они соответствуют назначению, и обнаружить дефекты.

2. **что такое qa, qc, testing**

Testing (Тестирование)- основная задача тестирования – выявить и зафиксировать дефекты. Проверка создаваемого программного продукта на соответствие требованиям к этому продукту.

QC (Quality Control, контроль качества)- основная задача контроля качества – предоставить объективную картину того, что происходит с качеством продукта на разных этапах разработки. Работает с готовым продуктом. Начинает работу уже на готовом продукте.

QA (Quality Assurance, обеспечение качества)- основная задача реализовать различные тактики для повышения качества на всех стадиях разработки: подготовка и установка стандартов, анализ качества, выбор инструментов, предотвращение ошибок и постоянное усовершенствование процесса тестирования.

3. **что такое качество-** совокупность определённых характеристик продукта, обусловливающих её пригодность удовлетворять потребности в соответствии с его назначением.

4. **что такое качество по** – качество напрямую связано с требованиями. Но так же соответствие и удовлетворённость всем требованиям, не может гарантировать что продукт понравится конечному пользователю. С этого можно вывести, что качественный продукт- это продукт, который удовлетворяет конечного пользователя, его запросам и нуждам. Это то насколько по удовлетворяет предъявляемым к нему требованиям.

5. **когда надо начинать тестирование** – тестирование нужно начинать как можно быстрее, потому что чем позже баг будет найден, тем дороже обойдется его исправление и тем больше усилий для этого потребуется.

6. **когда надо заканчивать тестирование (эвристика окончания тестирования) –** окончание тестирования может зависеть от таких вещей: 1. По решению менеджера или заказчика, возможно из-за того что продукт выпускать больше не целесообразно. 2. Достижение определенного уровня тестового покрытия. 3. Когда закончилось отведённое время на тестирование. 4. При достижение определенного уровня тестового покрытия.   
Так же есть такие виды эвристик окончания тестирования- Эвристика “Время вышло!”, Эвристика пиньяты, Эвристика ”мертвой лошади”, Эвристика ”Задание выполнено”, Эвристика ”Отмена задания”, Эвристика ”Я зашел в тупик! ”, Эвристика ”освежающей паузы”, Эвристика ”Отсутствие продвижения”, Эвристика “ Больше нет интересных вопросов”, Эвристика уклонения/безразличия.

7. **зачем тестировщику платят деньги**

Тестировщику платят деньги- за улучшения продукта, предоставление актуальной документации по продукту. Нахождение ошибок ценных для бизнеса.

Перечислите типичные возможные обязанности инженера по обеспечению качества?

Команда QA отвечает за мониторинг всего процесса разработки.

Они несут ответственность за отслеживание результатов каждого этапа SDLC и корректировку их в соответствии с ожиданиями.

Они несут ответственность за чтение и понимание необходимых документов.

Анализируют требования к тестированию, а также разрабатывают и выполняют тесты.

Разрабатывают Test case и расставляют приоритеты в тестировании.

Записывают проблемы и инциденты в соответствии с задачами проекта и планами управления инцидентами.

Работают с командой приложения и/или клиентом для решения любых проблем, возникающих в процессе тестирования.

Проводят регрессионное тестирование каждый раз, когда в код вносятся изменения для исправления дефектов.

Должны взаимодействовать с клиентами, чтобы лучше понять требования продукта.

Принимают участие в прохождении процедур тестирования.

8. **что такое Error/Mistake/Defect/Bug/Failure/Fault**

Ошибка- это человеческий фактор (как пример, что то не то в коде). Ошибка (Error) возникает из-за ошибки (Mistake) в написании кода разработчиком.

Дефект (Defect) это скрытый недостаток в ПО, возникший из-за ошибки в написании кода.

Когда дефект (Defect) обнаруживается тестировщиком, он называется багом (Bug).

Если тестировщики упустили дефект и его нашел пользователь, то это сбой (Failure).

Если программа в итоге не выполняет свою функцию, то это отказ (Fault).

9. **что такое баг репорт**

Это описание бага, документ описывающий ситуацию или последовательность действий которые вызвали некорректную работу объекта тестирования. Несоответсвие ожидаемого и актуального результата.

10. **структура баг репорта (основные пункты и все в целом)**

Баг репорт состоит из:

1. Summary- пишется по формуле- Что, где, когда(Что не работает, где проблема, когда, при каких обстоятельствах).
2. Description
3. Priority- очерёдность выполнения зачади. Hight, Medium, Low.
4. Severity- влияние на работу проекта.

Blocker(S1)- тестирование значительной части функциональности вообще недоступно.

Critical(S2)- не работает важная часть одной какой-либо функции либо не работает значительная часть, но имеется workaround.

Major(S3)- не работает важная часть одной какой-либо функции/бизнес-логики, но при выполнении специфических условий, либо есть workaround, позволяющий продолжить ее тестирование либо не работает не очень значительная часть какой-либо функции.

Minor(S4)- часто ошибки GUI, которые не влияют на функциональность, но портят юзабилити или внешний вид.

Trivial(S5)- почти всегда дефекты на GUI - опечатки в тексте, несоответствие шрифта и оттенка и т.п., либо плохо воспроизводимая ошибка, не касающаяся бизнес-логики.

1. Статус баг репорта
2. Кто завёл
3. Id репорта
4. Steps to reproduce
5. Actual/Expected result
6. Atachment

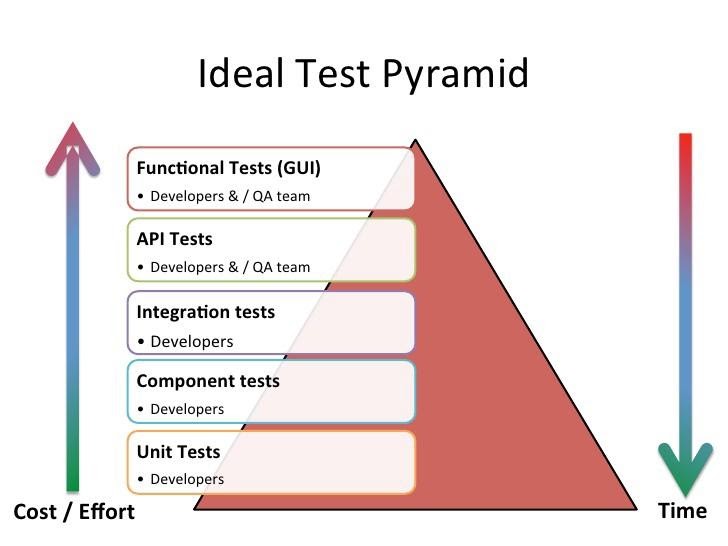
Основные пункты: 1. Summary 2. Steps to reproduce 3. Actual result 4. Expected result

11. **какие уровни уровни тестирования существуют**

Целью уровней тестирования является систематизация тестирования программного обеспечения и простота выявления всех возможных Test case на определенном уровне. Здесь работает хорошая аналогия с пирамидой тестирования.

**Что такое пирамида / уровни тестирования? (Testing Levels)**

«Пирамида тестов» — метафора, которая означает группировку тестов программного обеспечения по разным уровням детализации.



Component/Module/Unit testing(Компонентное, юнит, модульное)- проверяет функциональность модуля или одной операции. Тестировать изолируя друг от друга. Определяет то над чем производятся тесты: над отдельным модулем, группой модулей или системой в целом. Как правило баги исправляются в коде, без баг репорта и проводится за частую автоматизаторами или программистами.

Цель: проверка правильности реализации функциональных / нефункциональных требований в модуле, раннее обнаружение ошибок.

Целью тестирования модуля является не демонстрация правильного функционирования модуля, а демонстрация наличия ошибки в модуле, а также в определении степени готовности системы к переходу на следующий уровень разработки и тестирования.

Объект: модуль / компонент / unit

Но так же в одном из авторитетных источников есть разница.

**Unit testing**- тестирование отдельных программ, модулей, функций для демонстрации того, что программа выполняется согласно спецификации. Проверка в соответствии с design documents. Пишутся и выполняются(обычно) разработчиками. Выполняется первым.

**Component testing**- тестирование каждого объекта или частей программного обеспечения отдельно с или без изоляции других объектов. Проверка в соответствии с test requirements, use case. Проводится тестировщиками. Выполняется после Unit.

Integration testing(Интеграционное)- для проверки связи между компонентами и взаимодействие с различными частями системы. Тестирование двух компонентов(пользователь зарегистрировался и увидел свои данные в личном кабинете). Тип тестирования, при котором программные модули объеденяются логически и тестируются как группа. Целью является выявление багов при взаимодействие между програмными модулями и направлен на проверку обмена данными между этими модулями.

Выделяют 2 подтипа(уровня) интеграционного тестирования:

* Компонентное интеграционное тестирование(Component Integration testing) — проверяется взаимодействие между компонентами системы после проведения компонентного тестирования.
* Системное интеграционное тестирование (System Integration testing) — Проверяется взаимодействие между разными системами после проведения системного тестирования. Цель: проверка правильности реализации взаимодействия между компонентами / модулями / частями системы.

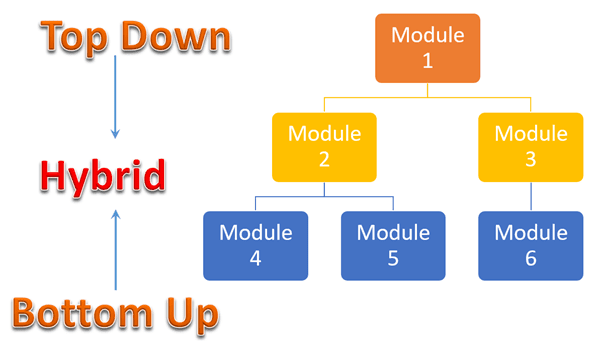
Объект: модули, состоящие из нескольких компонентов; под-системы, API, микросервисы

Подходы к интеграционному тестированию.

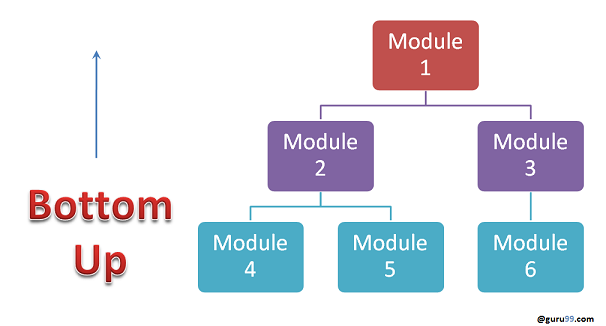
*Большой взрыв (Big bang integration).* Все или практически все разрабатываемые модули собираются вместе в виде законченной схемы или её основной части и затем проводится интеграционное тестирование. Хороший подход для сохранения времени.

***Инкрементальный подход.*** *При таком подходе тестирование выполняется путем объединения двух или более логически связанных модулей. Затем другие связанные модули поэтапно добавляются и тестируются для правильного функционирования. Процесс продолжается до тех пор, пока все модули не будут соединены и успешно протестированы. Включает в себя- Снизу в верх, Сверху в низ и сэндвич-подход.*

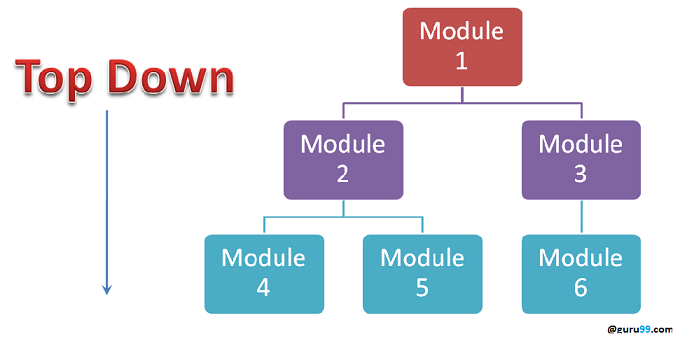
*Сэндвич-подход(Sandwich Approach)- Представляет собой комбинацию подходов сверху вниз и снизу-вверх. Здесь верхние модули тестируются с нижними модулями, а нижние модули интегрируются с верхними модулями и тестируются.*



*Снизу в верх (bottom up integration- Снизу в верх).* Все низкоуровневые модули, процедуры или функции собираются воедино и затем тестируются. После чего собирается следующий уровень модулей для проведения интеграционного тестирования. Считается этот подход полезным, если все или практически все модули разрабатываемого уровня готовы. Помогает определить по результату тестирования- уровень готовности приложения.



*Сверху в низ (Top down integration- Сверху в низ).* Вначале тестируются все высокоуровневые модули и постепенно добавляются низкоуровневые. Модули более низкого уровня симулируются заглушками с аналогичной функциональностью, затем по мере готовности они заменяются реальными активными компонентами.



System testing(системное)-

Системное тестирование следует начинать, только если все модули написаны и работают правильно. Тем не менее, это должно произойти до UAT (пользовательского приемочного тестирования).

Основная задача- проверка как функциональных так и не функциональных требований. Выполняется на полной интегрированной системе с целью проверки системы, исходных требований. Как все модули работают между собой. Можно ли пройтись по всему бизнес flow продукта. Выполняется основываясь на тест кейсах. Тестирование 3 или более компонентов или систем, которые в связке позволяют провести бизнес транзакцию. Можно выделить два подхода к системному тестированию: на базе требований (requirements based) and на базе случаев использования (use case based).

Основная задача системного тестирования - в выявлении дефектов, связанных с работой системы в целом, таких как неверное использование ресурсов системы, непредусмотренные комбинации данных пользовательского уровня, несовместимость с окружением, непредусмотренные сценарии использования, отсутствующая или неверная функциональность, неудобство в применении и тому подобное. Системное тестирование производится над проектом в целом с помощью метода "черного ящика". Структура программы не имеет никакого значения, для проверки доступны только входы и выходы, видимые пользователю. Категории тестов системного тестирования:

Полнота решения функциональных задач.

Стрессовое тестирование - на предельных объемах нагрузки входного потока.

Корректность использования ресурсов (утечка памяти, возврат ресурсов).

Оценка производительности.

Эффективность защиты от искажения данных и некорректных действий.

Проверка инсталляции и конфигурации на разных платформах.

Корректность документации

Acceptance testing(приёмочное)- проводится на этапе сдачи готового продукта или готовой части заказчику. “то тип тестирования программного обеспечения, который оценивает операционную готовность программного приложения до его выпуска в производство.

Приёмочные критерии- приоритетный список критериев, которым должен удовлетворять продукт/отдельная фича, перед тем как клиент их примет.

Цель: продукт выглядит и работает так как требовалось изначально. Решается заказчиком- принимаем приложение или нет. Длится до тех пор, пока заказчик не вынесет решение об отправлении приложения на доработку или выдаче. Финальный этап пред релизом.

**Пользовательское приемочное тестирование (UAT – User Acceptance testing).** Пользовательское приемочное тестирование (UAT) - это тип тестирования, выполняемый конечным пользователем или клиентом для проверки / принятия ПО перед его перемещением в production. UAT выполняется на заключительном этапе тестирования после выполнения функциональных, интеграционных и системных испытаний. Основной целью UAT является проверка end-to-end business flow. Он не фокусируется на косметических ошибках, орфографических ошибках или тестировании системы. Приемочное тестирование пользователя выполняется в отдельной среде тестирования с настройкой данных, аналогичных производственным.

**Эксплуатационное приемочное тестирование (OAT - Operational Acceptance testing).** Оценивает операционную готовность программного приложения до его выпуска в производство. Проводится до UAT.

Методы это про ящики

Уровни это модульное, интеграционное, системное, приёмочное.

Типы и виды одно и тоже. Связанные с изменениями- это регрессия, смок, санити, сборки.

Принципы- это 7 принципов

**Принципы тестирования:**

1. Тестирование демонстрирует наличие дефектов, а не их отсутствие

2. Исчерпывающее тестирование невозможно

3. Ранее тестирование сохраняет время и деньги

4. Скопление багов(кластеризация) 80% багов находится в 20% функций

5. Парадокс пестицида. Это когда пре регулярном прогон одних и тех же тестов, ошибки перестают находиться.

6. Тестирование зависит от контекста, зависит от того что мы тестируем

7. Заблуждение об отсутствии ошибок

**Виды тестирования по цели.**

Все виды тестирования программного обеспечения, в зависимости от преследуемых целей, можно условно разделить на следующие группы: функциональное и нефункциональное.

**Функциональные** - проверяет весь функционал продукта. Отвечает на вопросы- Работает ли система? Что тестируем?

Функциональное тестирование предполагает проверку работы функций системы, а нефункциональное – соответственно, общие характеристики нашей программы.

Всё зависит от функционала ПО, что-то может быть его функционалом, а что-то нет, в то время как в другом ПО наоборот. Например антивирус - функционал будет безопасность, а в другом -нефункциональным

Функциональное тестирование направлено на проверку того, какие функции ПО реализованы, и того, насколько верно они реализованы.

**К функциональным видам тестирования относятся:**

Функциональное- тестирование по в целях проверки реализуемости функциональных требований, способность по в определённых условиях решать задачи, нужные пользователям.

Функциональное тестирование: тестирование, основанное на анализе спецификации функциональности компонента или системы. Что делает наша система?

Функциональные возможности: Пригодность(Sutability). Правильность(Accuracy). Способность к взаимодействию(Interoperability). Согласованость(Compliance). Защищённость(Security).

**Security testing-** Это тип тестирования ПО, который выявляет уязвимости, угрозы и риски.

Состоит из двух ключевых вещей. 1 Проверить насколько система может противостоять различным типам атак 2 На сколько эта сама система передаёт данные и то на сколько конфиденциально их хранит. Свойство по, отражающее его способность не допускать неавторизованный доступ, случайный или умышленный, к программам и данным.

Целью тестов безопасности является выявление всех возможных лазеек и слабых мест в ПО, которые могут привести к потере информации, доходов, репутации компании, сотрудников или клиентов. Общая стратегия безопасности основывается на трех основных принципах:

Конфиденциальность - сокрытие определенных ресурсов или информации

Целостность – ресурс может быть изменен только в соответствии с полномочиями пользователя

Доступность - ресурсы должны быть доступны только авторизованному пользователю, внутреннему объекту или устройств

Разработка безопасности по должна начинаться с выбора подхода и технологии, затем прописываются требования, затем они тестируются.

Vulnerability Assessment- нахождений уязвимостей. Обнаружение и сканирование. Фокус на Ширину. Высокое покрытие. Предоставит частичную информацию об уязвимостях.

Penetration testing- Выявление и использование уязвимостей. Симуляция. Глубина. Низкое покрытие. Предоставит полную информацию об уязвимостях

**Interoperability testing(тестирование взаимодействия)-** Тестирование взаимодействия - функциональное тестирование, проверяющее способность приложения/устройства взаимодействовать с одним и более компонентами/системами/устройствами и включающее в себя тестирование **совместимости (compatibility testing) и интеграционное тестирование(это уровень тестирования) (integration testing).**

**Compatibility testing (Тестирование совместимости)**- Тестирование совместимости помогает убедиться в функциональных возможностях и надёжности работы продукта в поддерживаемых браузерах (если речь идет о Web-приложениях) и операционных системах. Также может проверяться работоспособность продукта при использовании различных аппаратных платформ.

Тестирование совместимости, по идее, на различных платформах - там компьютеры, ноутбуки, планшеты, смартфоны и т.д.

**К нефункциональным видам тестирования относятся:**

Нефункциональное описывает тесты для определения характеристик по, которые могут быть измерены разными величинами. **Отвечает на вопрос- как себя ведёт система? Как хорошо работает система**?

Тестирование атрибутов компонента или системы, не относящимся к функциональности, то есть надёжность, эффективность, практичность, сопровождаемость и переносимость.

[**Все виды тестирования производительности**](http://www.protesting.ru/testing/types/loadtesttypes.html)**(Performance testing):**

**Производительность(performance):** степень, с которой система или компонент выполняет заложенные в неё функции в установленных рамках на время обработки и пропускную способность. Цель: Установить реальную производительность программного продукта.

Тестирование производительности- это класс тестирования ПО, который фокусируется на производительности системы при определенной нагрузке. Он не ищет напрямую ошибки или дефекты. Он производит аналитику на основе эталонных тестов и предоставляет разработчику всю диагностическую информацию, необходимую для выявления проблем производительности и узких мест. При этом происходит:

* измерение времени выполнения выбранных операций при определенных интенсивностях выполнения этих операций
* определение количества пользователей, одновременно работающих с приложением
* определение границ приемлемой производительности при увеличении нагрузки (при увеличении интенсивности выполнения этих операций)
* исследование производительности на высоких, предельных, стрессовых нагрузках

**Capacity (ёмкости/способностей)-** базовый тест, который обычно выполняется первым. Все последующие тесты на среднее время ответа, регенерацию системы, утилизацию ресурсов нужно выполнять с оглядкой на результаты Capacity. Используемый подход: ступенчато повышаем нагрузку до момента, когда время ответа начнет расти экспоненциально. Экспоненциальный рост времени ответа, как правило, связан с полной утилизацией одного из ресурсов, из-за которого запросы вместо моментальной обработки выстраиваются друг за другом и ждут своей очереди на обработку.

**Scalability Test (Тестирование масштабируемости)** -профиль нагрузки тот же, что и при нагрузочном тестировании. Что получаем в результате? Ответы на следующие вопросы:

Увеличится ли производительность приложения, если добавить дополнительные аппаратные ресурсы?

Увеличится ли производительность пропорционально количеству добавленных аппаратных средств?

**Stress testing(стрессовое тестирование)-** вид тестирования производительности, оценивающий систему или компонент на граничных значениях рабочих нагрузок или за их пределами или к примеру в состоянии ограниченных ресурсов, таких как память или доступ в серверу. Позволяет проверить на сколько продукт в целом работоспособен в условии стресса и оценить способность системы к регенерации. Повышение интенсивности выполнения операций до очень высоких значений. На сервер зашло очень много юзеров и как себя система поведёт когда они уйдут? Объём запросов. Стрессовое тестирование выполняется самым первым, если нет отдельного Capacity тестирования, хотя по факту это все равно будет Capacity, т.к. нагрузка берется «с потолка».

Цель: Проверка того, что система адекватно реагирует на те или иные стрессовые ситуации.

**Load testing(нагрузочное)-** один из подтипов тестирования производительности. Измеряет и притирается, что может наша система. Идёт постепенная нагрузка до максимума. Снимаем показатели: время отклика системы(скорость), загрузки CPU. Вид тестирования производительности, проводимый с целью оценить поведение компонента или ситемы под увеличивающейся нагрузкой, для опеределения максимально допустимого уровня нагрузки. Пример: как быстро открывается страница? Как быстро открывается страница под 1000 юзеров?( это и тестирование производительности и тестирование назгрузочное). Открывается ли страница под 1000 пользователей?(% ошибок).

Цель: Убедиться в том, что система работает соответственно ожидаемым рабочим нагрузочным параметрам (какой предел работоспособности)

**Volume testing(Объёмное тестирование)-** мы берём количество пользователей и обращаем внимание не на количество запросов которые они делают, а на количество вводных данных которые они загружают в нашу систему. Объём загружаемых данных в систему. К примеру сразу 100 юзеров заливают 100 картинок. Хватит ли нам рессурсов?

**Stability testing(тестирование стабильности, тестирование надёжности)-** процесс тестирования, исследующий надежность по. Надёжность- способность по функционировать при заданных условиях на протяжении определённого периода времени. Когда уже измерили нормальное значение пользователей и запускаем с ним на много часовую/денную прогонку. Этот тип тестирования выполняется в самом конце.

**Spike(спайк/шиповое тестирование)-** этот вид тестирования предназначен для определения поведения системы при внезапном увеличении нагрузки (большого количества пользователей) на систему. Например, дни распродаж в интернет-магазине.

**Ramp тестирование-** это метод тестирования, который предлагает ступенчато поднимать нагрузку до тех пор, пока система не выйдет из строя.

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

**Installation testing(Установки)** - В процессе инсталляционного тестирования проверяется корректность установки и деинсталляции программного продукта в среде максимально приближенной к эксплуатационной. Успешность установки, его настройки, обновление, удаление. Кроссплатформенность/кроссбраузерность.

Цель: Основная цель состоит в том, чтобы убедиться, что продукт может быть установлен/деинсталлирован при различных условиях – таких как: новая инсталляция, усовершенствование системы (upgrade), установка по умолчанию, полная установка, установка по выбору.

**Configuration Testing (Конфигурационное тестирование)** — специальный вид тестирования, направленный на проверку работы программного обеспечения при различных конфигурациях системы. С разным железом. С помощью конфигурационных тестов проверяется совместимость продукта с различным программным (software) и аппаратным (hardware) обеспечением.

Конфигурационное тестирование - это, ЕМНИП, тестирование, к примеру, на только компьютерах с различным набором железа/софта и т.п.

Цель: Проверить работоспособность системы при различных конфигурациях.

**Compatibility testing (Тестирование совместимости)-** Тестирование совместимости помогает убедиться в функциональных возможностях и надёжности работы продукта в поддерживаемых браузерах (если речь идет о Web-приложениях) и операционных системах. Также может проверяться работоспособность продукта при использовании различных аппаратных платформ.

Тестирование совместимости, по идее, на различных платформах - там компьютеры, ноутбуки, планшеты, смартфоны и т.д.

**Usability testing(тестирование практичности/удобства использования)-** проводится с целью определения степени понятности, легкости в изучении и использовании. Удобен или интерфейс, навигация, понятность, обучаемость, читабельность, цветовое сопровождение в заданных условиях эксплуатации. Мы как тестировщики не может дать точную оценку, проводим только по тому что было указано в требованиях. Так как для этого нужно делать тесты на целевой аудитории. USABILITY testing показывает, насколько проста в использовании и удобна система программного обеспечения. Так же это UX тестирование.

**Failover and Recovery testing(тестирование на отказ и восстановление)-** коректное реагирование как на отказ так потом и на восстановление. Как сработает бекап. 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.

**Maintainability testing(тестирование сопровождаемости).** Сопровождаемость- это легкость изменения по для исправления дефектов, для соответсвия новым требованиям, с целью облегчения последующего сопровождения или для адаптации к изменившемуся окружению. Тестирование сопровождаемости- процесс тестирования для определения сопровождаемости по.

**Portability testing(Тестирование переносимости)-** процесс тестирования с целью определить переносимость по. Легкость с которой по может быть перенесён из одного аппаратного или програмного окружения в другое.

**User Interface testing (Тестирование графического интерфейса)** — это тип тестирования программного обеспечения, который проверяет графический интерфейс пользователя тестируемого приложения. Тестирование GUI включает проверку экранов с помощью элементов управления, таких как меню, кнопки, значки и все типы панелей — панель инструментов, строка меню, диалоговые окна, окна и т. Д.

Цель тестирования графического интерфейса пользователя (GUI) — обеспечить функциональность интерфейса пользователя. работает согласно спецификации.

Цель: Обнаружение ошибок в интерфейсе и поиск ошибок в функциональности посредством интерфейса.

**Internationalization testing(Тестирование интернационализации)-** процесс разработки приложения, при котором код самого приложения независим от любых языковых и культурных особенностей региона или страны. Суть интернационализации в том, что бы сделать процесс локализации проще, дешевле и быстрее. Реализацию интернационализации обычно начинают на ранних этапах проекта, что бы подготовить продукт к будущей локализации. Во время процесса интернационлазиации определяют, что будет изменяться для будущих локалей и выносят эти данные во внешний файл. Так же во время интернационализации нужно добавить возможность изменять календари, форматы данных, времени, цифр, денежных символов и в целом символов, специфичных для определённых языков.

Цель: Проверить способность продукта быть быстро локализованным под необходимую локаль потенциальных пользователей системы

**Localization testing(Тестирование локализации)-** проверяет, правильно ли локализован продукт. То есть, переведен на другой язык и корректно работает с учетом национальных особенностией страны или региона, в котором будет продаваться и использоваться продукт.

Цель: Проверить, правильно ли локализован продукт.

**Тестирование документации (Documentation testing)-** Плохая документация может повлиять на качество продукта. Хорошая документация по продукту играет решающую роль в конечном продукте. Тестирование артефактов, разработанных до, во время и после тестирования продукта, называется тестированием документации. Это нефункциональный тип тестирования программного обеспечения.

**Связанные с изменениями виды тестирования.**Три эти вида тестирования нацелены на то, чтобы избежать потерь времени и усилий, чтобы быстрее определить недостатки ПО и их критичность, а так же то, заслуживает ли оно перехода в фазу более углублённого и тщательного тестирования или же нет.

**Дымовое тестирование (Smoke Testing)-** проверка самых важных частей функционала. В системе произошли изминения, добавились некоторые функции и нужно проверить, что эти функции не сломали наши важные части. Основной флоу продукта функционирует. Направлено в ширь. Для покрытия тестами как можно большего функционала в кратчайшие сроки.

Выполняются каждый раз, когда мы получаем новый билд (версию), проекта (системы) на тестирование, при этом считая её относительно нестабильной.

Цель: Выявление проблем «лежащих на поверхности» – тестируется чаще всего основная бизнес логика программы.

**Тестирование сборки (Build Verification Test)-** Как и дымное тестирование, направленно для предварительной проверки разрабатываемого программного продукта перед запуском полномасштабного тестирования по всем параметрам, проводимого QA-командой. Проводится оно для того, чтобы знать – готов ли релиз для такого этапа разработки ПО, как Тестирование или же он еще нуждается в доработке.

Тестирование конкретной сборки, билда. Проверить что добавленные измeнения, функции и исправленные баги не оказали негативного влияния на уже успешно действующий функционал.

**Санитарное тестирование (Sanity Testing)-** Используется для определения работоспособности определенной части приложения после изменений произведенных в ней или окружающей среде. Конкретная часть функции- тестируем её в глубь. Используется каждый раз, когда мы получаем относительно стабильный билд ПО, чтобы определить работоспособность в деталях. Иными словами, здесь проходит валидация того, что важные части функциональности системы работают согласно требованиям на низком уровне.

**New Feature Test (NFT,NF)-** в данном виде тестирования акцент делается на тестировании новой функциональности, появившейся в конкретном выпуске (build) программного продукта.

**Регрессионное тестирование (Regression Testing)-** выполняется при выходе новой версии по с целью проверки не был ли повреждён старый работающий функционал новыми изменениями. За частую это прогон большей части функционала, но если функционал не затронули изменения, прогер ничего с ними не делал- их можно не добавлять в тесты. Цель регрессионного тестирования подтвердить, что новые изменения кода не должны иметь побочных эффектов для существующих функций.

Цель: Выявление потенциальных проблем, которые могли возникнуть в результате изменений. Проверка исправления найденных ранее дефектов- регрессия. Регрессионное тестирование проводится для подтверждения того, что недавнее изменение программы или кода не оказало неблагоприятного воздействия на существующие функции.

**Тест критического пути (critical path test)-** с помощью данного вида тестирования покрываются все сценарии стандартного использования приложения(повседневного). Данный тип также характеризует тестирование по глубине его проведения. В критикал пас более расширено, то есть ещё смотрим что будет если убрать товар из корзины, увеличить количество, то есть то, что делает пользователь.

Для данного вида тестирования пишутся наиболее подробные и глубокие тест-кейсы, чтобы покрыть всю возможную функциональность приложения.

Тест критического пути может быть как позитивным, так и негативным.

**Расширенные тесты(Extended tests)-** выполняются для изучения всей функциональности, указанной в требованиях. Проверяется даже функциональность с низким приоритетом. При этом в этом тестировании нужно понимать, какой функционал наиболее ценный, а какой менее важный. При условии, что у вас достаточно времени или других ресурсов, тесты на этом уровне можно использовать для требований с низким приоритетом. Проверяется нестандартное использование программного продукта, границы переполнения массивов данных, ввод специальных символов, нелогичное кликанье по кнопкам, открыть одно окно и закрыть предыдущее, и так далее.

**Re-test(повторное тестирование)** - это тип тестирования, выполняемый для проверки того, что конкретный дефект устранен после исправления кода. Повторное тестирование проводится для подтверждения того, что тест-кейсы, которые не прошли, проходят после устранения дефектов. Приоритет повторного тестирования выше, чем регрессионное тестирование, поэтому оно проводится перед регрессионным тестированием.

**Методы тестирования по знанию системы(Доступ к коду программного продукта)**

**Тестирование черного ящика(Black box)-** Мы не знаем, как устроена внутри тестируемая система. Тестер имеет доступ к по только через теже интерфейсы что и пользователь. Не используется знание о внутренем устройсе объекта тестирования. Как правило такое тестирование ведётся с использование Software Requirement Specification, описывающих требования к системе. Если документация отсутствует то на основании **Product Backlog.[Product Backlog- приоритетный список требований с оценкой трудозатрат. Обычно состоит из бизнес требований, которые приносят конкретную бизнес ценность]** Тестирование, как функциональное, так и нефункциональное, не предполагающее знания внутреннего устройства компонента или системы.

тест-дизайн, основанный на технике черного ящика – процедура написания или выбора тест-кейсов на основе анализа функциональной или нефункциональной спецификации компонента или системы без знания ее внутреннего устройства.

Целью этой техники является поиск ошибок в таких категориях: – неправильно реализованные или недостающие функции; – ошибки интерфейса; – ошибки в структурах данных или организации доступа к внешним базам данных; – ошибки поведения или недостаточная производительности системы; Таким образом, мы не имеем представления о структуре и внутреннем устройстве системы. Нужно концентрироваться на том, ЧТО программа делает, а не на том, КАК она это делает.

**Тестирование белого ящика(White box)-** Тестирование методом белого ящика предполагает, что внутренняя структура/устройство/реализация системы известны тестировщику. Имеет доступ к исходному коду, базе данных. Мы выбираем входные значения, основываясь на знании кода, который будет их обрабатывать. Точно так же мы знаем, каким должен быть результат этой обработки. Знание всех особенностей тестируемой программы и ее реализации – обязательны для этой техники. Метод тестирования ПО, который предполагает, что внутренняя структура/устройство/реализация системы известны тестировщику. Мы выбираем входные значения, основываясь на знании кода, который будет их обрабатывать. Точно так же мы знаем, каким должен быть результат этой обработки. Знание всех особенностей тестируемой программы и ее реализации – обязательны для этой техники. Тестирование белого ящика – углубление во внутреннее устройство системы, за пределы ее внешних интерфейсов. Метод белого ящика применима на разных уровнях тестирования – от модульного до системного, но главным образом применяется именно для реализации модульного тестирования компонента его автором.

* Охват операторов: - Этот метод требует, чтобы каждое возможное утверждение в коде было проверено хотя бы один раз в процессе тестирования разработки ПО.
* Покрытие ветвления - этот метод проверяет все возможные пути (если-еще и другие условные циклы) программного приложения.
* Покрытие условий
* Покрытие функций- каждая ли функция программы была выполнена
* Покрытие вход/выход- все ли вызовы функций и возвраты были выполнены

**Тестирование серого ящика(Grey box)-** совмещаются приёмы используемые при тестировании чёрного и белого ящика. Сочетание происходит так: снаружи на продукт смотрим как на чёрный ящик, но выбор тестов основывается на знании внутреннего устройства, знании кода. Предполагается, например, доступ к внутренней структуре и алгоритмам работы ПО для написания максимально эффективных тест-кейсов, но само тестирование проводится с помощью техники черного ящика, то есть, с позиции пользователя. Техника серого ящика применима на разных уровнях тестирования – от модульного до системного, но главным образом применяется на интеграционном уровне для проверки взаимодействия разных модулей программы. У нас нет доступа к коду, но мы может делать например запросы к бд, дёргать API.

**Остальные виды тестирования.**

**Exploratory testing(исследовательское тестирование)-** это одновременное изучение продукта, проектирование тестов и их выполнение. Это неформальный метод проэктирования тестов, при котором тестер активно контролирует проектирование тестов в то время, как эти тесты выполняются и использует полученную во время тестирования инфу для проектирования новых тестов. Если каждый следующий тест, который выполнил тестировщик, выбирается по результатам предведущего теста, это означает что мы используем Exploratory testing. Exploratory testing- Не является методикой тестирования, это скорее подход, который можно применить к любому виду тестиования. Такое тестирование подразумевает под собой одновременно изучение проекта, функционала, проектирование тест кейсов в уме и тут же их исполнение, не записывая и не создавая тестовую документацию. Цель данного тестирования — это углубление в познании продукта, приложения и нахождения «на лету» возможных багов. Исследовательское тестирование широко используется в Agile-моделях. Иследовательское тестирование- мыслить со стороны конечного пользователя, что бы мы тестили не на собвственных предположенях, а то как это может юзать юзер.

Предугадывание ошибки- пересекается с испледовательским, больше направлено на негативные сценарии.

**Adhoc(Интуитивное тестирование или свободное)-** вид тестирования который выполняется без подготовки к тестироваемому продукту, без определения ожидаемых результатов, проектирование тестовых сценариев. Неформальное, импровизационное тестирование. Не требует никакой документации, планирования, процессов. Выполняется когда заказчик не обладает конкретными целями, документацией и поставленными задачами.

Виды свободного тестирования (ad-hoc testing):

**Buddy testing** – процесс, когда 2 человека, как правило разработчик и тестировщик, работают параллельно и находят дефекты в одном и том же модуле тестируемого продукта. Такой вид тестирования помогает тестировщику выполнять необходимые проверки, а разработчику исправлять множество дефектов на ранних этапах.

**Pair testing** – процесс, когда 2 тестировщика проверяют один модуль и помогают друг другу. К примеру, один может искать дефекты, а второй их документировать. Таким образом, у одного тестировщика будет функция, скажем так, обнаружителя, у другого – описателя. Выполняется только тестировщиками с разным уровнем знаний и опыта (такое сочетание поможет поделиться взглядами и идеями).

**Monkey testing** – произвольное тестирование продукта с целью как можно быстрее, используя различные вариации входных данных, нарушить работу программы или вызвать ее остановку (простыми словами – сломать).

**A/B тестирование**- используется для оценки и управления еффективности веб страницы. Так же этот метод называется сплит тестированием. Помогает оценивать еффективность изменений страницы, например добавление новых элементов дизайна. Практический смысл использования этого метода заключается в поиске и внедрении компонентов страницы увеличивающих её результативность. А/B тестирование это прикладной маркетинговый метод с помощью которого можно влиять на конверсию, прибыль веб проекта. Сплит тестирование начинается с оценки метрик существующей веб страницы и поиска способов её улучшения. Выбор объекта тестирования зависит от цели и задач, которые перед собой ставит маркетолог. Можно провести с помощью google Optimize. При тестировании AB мы создаем и анализируем два варианта приложения, чтобы найти, какой вариант работает лучше с точки зрения пользовательского опыта, потенциальных клиентов, конверсий или любой другой цели, а затем в конечном итоге сохранить наиболее эффективный вариант.

**E2E - End–to–End(сквозное тестирование)**- суть, посмотреть так ли работает прогамма для конечного юзера, как расчитывалось изначально. Е2е- конечный этап тестирования. Проводится в более реалистичном сценарии конечного пользователя. Весь меин флоу какой-то части функционала. Выполнение Е2Е гарантирует что приложение проверено на основе пользовательских сценариев , которые помогают: проверять и выполнять потоки приложения от его начала до конца, тестирование всего потока приложения. Увеличить тестовое покрытие. Сквозное тестирование обычно выполняется после функционального и системного тестирования. Сквозное тестирование включает проверку внешних интерфейсов, которые могут быть сложными для автоматизации. Следовательно, ручное тестирование является предпочтительным.

**Verification(Верификация)-** проходит без запуска кода, проводится до валидации. Делаем ли мы продукт правильно? И все ли требования учитываются при этом верно или не верно. Верификация поможет определить, является ли ПО высокого качества, но она не гарантирует, что система полезна. Подтверждение того что определённые требования были выполнены. Наличие чего либо.

**Validation(Валидация)-** проверка что продукт отвечает ожиданиям и требованиям пользователя. Включает в себя- запуск кода продукта. Отвечает на вопросы- делаем ли мы правильно продукт. Работоспособность чего либо. Валидация происходит после Verification-

**Статическое тестирование**

При статическом тестировании код не выполняется. Вы вручную проверяете код, документы требований и проектные документы на наличие ошибок. Отсюда и название «статичный». Основная цель этого тестирования - повысить качество программных продуктов путем выявления ошибок на ранних этапах цикла разработки.

**Динамическое тестирование**

При динамическом тестировании выполняется код. Оно проверяет функциональное поведение ПО, использование памяти / процессора и общую производительность системы. Основная цель этого тестирования - подтвердить, что программный продукт работает в соответствии с требованиями бизнеса. Это тестирование также называется Execution technique или validation. Динамическое тестирование выполняется на всех уровнях тестирования, и это может быть либо тестирование черного, либо белого ящика.

**Превентивный (профилактический) подход:** он также известен как Verification Process. Этот подход заключается в предотвращении дефектов. При таком подходе тесты разрабатываются на ранних этапах SDLC, то есть до того, как программное обеспечение было создано. Он подпадает под анализ качества (QA).

**Реактивный подход:** он также известен как Validation Process. Этот подход заключается в выявлении дефектов. При таком подходе тесты предназначены для выполнения после того, как программное обеспечение было произведено. Здесь мы пытаемся найти недостатки. Подпадает под контроль качества (QC).

**STLC жизненный цикл тестирования.** В процессе STLC выполняются различные действия для улучшения качества продукта.

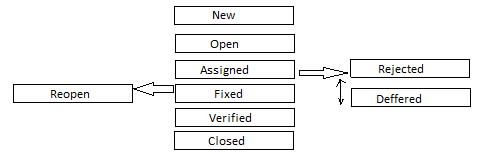
1. Анализ требований
2. Планирование тестирования- стратегии, подходы.
3. Проектирование тест кейсов.
4. Настройка тесовой среды.
5. Выполнение тестирования
6. Закрытие теста.

**SDLC жизненный цикл ПО.**

1. Идея- сбор того что хочется показать юзеру.
2. Разработка и сбор требований. Подключаются бизнес аналитики. Анализ рисков, создаются планы валидации и верификации, критерии приёмки
3. Этап дизайна. Дизайнеры на основании требований рисуют макапы.
4. Разработка- если утрвердили дизайн. Какие технологии используются, инструменты, рабочии процессы, взаимосвязи, структура БД.
5. Тестирование.
6. Ввод в эксплуатацию.
7. Поддержка.
8. Вывод из эксплуатации.

**Жизненный цикл бага**

1. Новый(New)- занесён в баг трекер.
2. Открыт(Opened)- назначить на того кто проанализирует, например ПМ и в зависимости от его решения.
3. Отложен(deffered)- фикс не несёт ценности.
4. Отклонён(Rejected)- может это не дефект.
5. 5.Дубликат(dublicate)
6. Назначен(assigned)- асайнится на разработчика который должен фиксить.
7. Исправлен(fixed)
8. Проверен(verified)- тестер проверяет действительно ли пофикшен баг.
9. Повторно открыт(reopened)
10. Закрыт(Closed).



**Основные фазы тестирования.**

**Pre-Alpha**: - ПО является прототипом. Пользовательский интерфейс завершен. Но не все функции завершены. На данном этапе ПО не публикуется.

**Alpha:** является ранней версией программного продукта. Цель - вовлечь клиента в процесс разработки. Хороший Альфа-тест должен иметь четко определенный план тестирования с комплексными тестовыми примерами. Это дает лучшее представление о надежности программного обеспечения на ранних стадиях. В некоторых случаях тестирование может быть передано на аутсорс.

**Beta:** ПО стабильно и выпускается для ограниченной пользовательской базы. Цель состоит в том, чтобы получить отзывы клиентов о продукте и внести соответствующие изменения в ПО.

**Release Candidate (RC):** основываясь на отзывах Beta Test, вы вносите изменения в ПО и хотите проверить исправления ошибок. На этом этапе вы не хотите вносить радикальные изменения в функциональность, а просто проверяете наличие ошибок. RC также выпущен для общественности

**Release:** Все работает, ПО выпущено для общественности.

**Что такое эвристики окончания тестирования?**

Это тестирование при котором стратегия тестирования основывается на предварительном опыте и информации о вероятности различных событий.

**Эвристики –** это быстрые, недорогие способы решения проблемы или принятия решения.

**Эвристика «Время вышло!»** Мы останавливаем тестирование, когда заканчивается выделенное на него время.

**Эвристика «мертвой лошади»** В программе слишком много ошибок, так что продолжение тестирования не имеет смысла. Мы знаем, что все изменится настолько, что сведет на нет результаты текущего тестирования.

**Эвристика «Задание выполнено»** Останавливаем тестирование, когда найдены ответы на все поставленные вопросы.

**Эвристика «Отмена задания»** Наш клиент сказал нам: «пожалуйста, прекратите тестирование». Это может произойти по причине перерасхода бюджета, или вследствие отмены проекта, и по любой другой причине.

**Эвристика «Я зашел в тупик!»** По какой бы то ни было причине мы останавливаемся, поскольку обнаруживаем некое препятствие. У нас нет информации, которая нам требуется (например, многие люди заявляют, что не могут тестировать без достаточного количества спецификаций). Имеется блокирующая ошибка, и таким образом мы не можем перейти в ту область продукта, которую необходимо протестировать, у нас нет необходимого оборудования или инструментария, у команды нет квалификации, требуемой для выполнения некоторых специальных тестов.

**Эвристика «освежающей паузы»** Вместо прекращения тестирования мы приостанавливаем его на некоторое время. Мы можем остановить тестирование и сделать перерыв, когда мы устали, когда нам стало скучно или пропало вдохновение. Мы можем сделать паузу на то, чтобы выполнить некоторые исследования, разработать планы, поразмыслить над тем, что мы делали в прошлом и понять, что делать дальше. Идея заключается в том, что нам требуется определенный перерыв, после которого мы сможем вернуться к продукту со свежим взглядом или свежими мыслями.

**Эвристика Привычного завершения.** Мы останавливаем тестирование тогда, когда мы обычно останавливаем тестирование. Имеется протокол, задающий определенное количество идей для тестирования, или тест-кейсов, или циклов тестирования, или как вариант – имеется определенный объем работ по тестированию, который мы выполняем и после этого останавливаемся. Agile-команды, например, часто применяют такой подход: «когда выполнены все приемочные тесты, мы знаем, что продукт готов к поставке.

**Мнемоника** – это набор правил и приемов, которые помогают эффективно запоминать необходимые сведения (информацию), простым и доступным способом.

**Эстимация**- оценка трудозатрат на тестирование.

**Требования**  
**Характеристики качественных требований:**

* **Единичность**- требования описывают одну и только 1 вещь
* **Завершённость**- они завершённые и вся необходимая информация присутствует. Хранятся в одном месте.
* **Последовательные**- требования не противоречат другим требованиям. Логически понятны
* **Атомарность**- требование не может быть разбито на ряд более детальных требований, без потери завершённости.
* **Отслеживаемость**- требование полностью или частично соответствует деловым нуждам как заявлено заинтересованными лицами и документировано.
* **Актуальность**- Требование не стало устаревшим с течением времени
* **Выполнимость**- требование может быть реализовано в рамках проекта
* **Недвусмысленность**- возможна 1 и только 1 интерпритация. Использование отрицательных утверждений- запрещено.
* **Обязательность**- предоставляет характеристику, отсутсвие которой приедёт к неполноценному решению
* **Проверяемость**- реализованные требования могут быть протестированы. Все требования должны быть поддающиесь проверке. Общепринятая методика проверки- тесты. Если проверка тестами невозможна, тогда должен исполнятся другой метод проверки(анализ, демонстрация, осмотрт или обзор дизайна).

**Спецификация требований ПО**- документ, который содержит полное и чёткое описание разрабатываемого продукта. Так как спецификация служит в том числе и для получения обратной связи от клиента, написана она должна быть в простой и лёгкой для восприятия форме.

Спецификация нужна для:

- можно получить точную оценку стоимости рисков и времени

- клиент может более чётко сформировать своё виденье о продукте

- может выявить оптимальный набор функций

- служит основой для формирования другой тестовой документации

- процесс разработки будет оптимизирован, минимализированы затраты времени

- Никакого дублирования задач

- Позволяет структурировать проблемы, что бы их быстрее решать

- Помогает понять какие результаты считаются оптимальными при тестировании

Requirements- требования, совокупность утверждений относительно свойст, качества и как их нужно реализовывать.

**Виды требований по уровню.**  
Бизнес требований- определяют назначение ПО. описываются в документе о виденье(vision) и границах проекта(scope)

Пользовательские требования- определяют набор пользовательских задач, которые должна решать программа. Могут выражаться в виде фраз утверждений, в виде сценариев использования(use case), пользовательских историй(user stories).

Виды требований по характеру.

Функциональный характер- требования к поведению системы( что система делает). Бизнес требования. Пользовательские требования. Функциональные требования.

Не функциональные требования к характеру поведения системы.

Бизнес правила

системные требования и ограничения- определение элементарных операций, которые должна иметь система, а так же различных условий, которым она может удовлетворять.

Требования к атрибутам качества. Ограничение на программные интерфейсы

Требования к докумнетированию

Требования к дизайну и юзабилити

Требования к безопасноти и надёжности

Требования к показателям назначения(производительность, устойчивость к сбоям)

Требования к эксплуатации и персоналу

Источники требований- заказчик, конечный потребитель, документация, сегмент рынка бизнеса.

**Test metrics нужны для:**

1. оценить соотвествие определённых характеристик на проекте
2. найти возможные причины проблем
3. Принять решение по исправлению ситуации
4. Оценить прогресс/регресс после ввода коректировок
5. Улучшить определённые характеристики на проекте
6. Привести объективные аргументы

**Группа 1**- требования к разработке ПО:

- тестовое покрытие требований

- степень взаимосвязи требований

- коифициент стабильности требований

**Группа 2**- качество разрабатываемого продукта:

- плотность дефектов

- коэфициент регресии

- коефициент повторно открытых багов

- средняя стоимость исправления дефекта

**Группа 3**- возможности и эффективность команды QA

- скорость работы команды

**Группа 4**- кочество работы команды тестирования

- еффективность тестов и тестовых наборов

- коефициент ошибок пропущенных на прод

- реальное рвемя работы команды QA

**Группа 5**- обратная связь удовлетворённость пользователей

**Техники тест дизайна**

Их цель- оптимальное покрытие тестами за минимальное количество времнеи.

**Equivalence Partitioning(Эквивалентные классы)-** разбор на группы от которых ожидается сходное поведение, они должны обрабатывать аналогичным образом. Эквивалентные классы могут быть опеределены как для валидных так и для невалидных данных. Если программа правильно обработала одно значение из класса эквивалентности, значит программа будет корректно работать для всех значений из этого класса. Эквивалетные классы включают в себя определение количества каких то значений, которые при обработке или после неё, будут выдавать на выходе один и тот же результат для всех представителей этого эквивалентного класса. Мы берём из каждого класса по 1 значению, советуют брать значение из середины и тестировать его.

Их можно найти в наборах без номеров (например, листья деревьев, разделенные по цвету - желтый, зеленый и т. д.), или даже один элемент может быть классом эквивалентности (например, лифт обычно более полон на первом этаже, чем на других этажах).

**Boundary Value Analysis(Анализ граничных значений)-** границы эквивалентных классов, начало эквивалетного класса и конец. Метод и опыт гласит- вероятность найти баги как раз на этих границах. Сначала нужно определить классы эквивалентности определить границы диапазонов и провести три теста для границ(на самой границе, значение выше этой границы +1 и значение меньше этой границы -1). Если дроби то- 54,9/55/55.1 Помнить о величине шага.

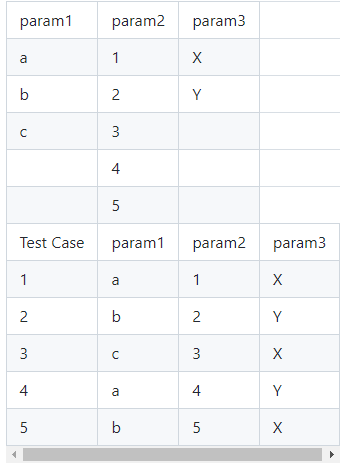
**State transition testing(Тестирование таблицы переходов)-** суть в том что бы понять в каких состояних может перебывать наша система, что с ней может происходить и какие между этими состояниями есть действия. Используется для проверки способности ПО входить и выходить из определённых состояний. Состоит из 4 основных частей- состояния которые может принимать ПО. Переходы из 1 состояния в другое. Действия которые вызывают переходы. Действия которые возникают в результате перехода. В этом методе тестировщик предоставляет как положительные, так и негативные входные значения теста и записывает поведение системы.

**Cause/Effect(Техника Причина/следствие)-** есть некие условия, комбинация условий(причин) и есть ожидание того что мы хотим получить от этих ожиданий(следствие).

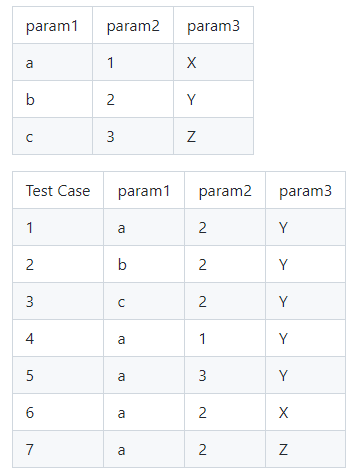
**Exhaustive testing (Техника исчерпывающее тестирование)-** проверка всех возможных состояний и комбинаций. Это крайний случай. Возможно можно применить в банковской сфере, если была утечка данных.

**Pairwise testing(Попарное тестирование)-** Сформулировать суть можно, например, вот так: формирование таких наборов данных, в которых каждое тестируемое значение каждого из проверяемых параметров хотя бы единожды сочетается с каждым тестируемым значением всех остальных проверяемых параметров. Анализируем и тестим только то что актуальное, по приоритету. Для pairwise достаточно что бы каждое значение всех пар, хотя бы один раз сочеталось с другими значениями оставшихся параметров. Смысл метода не в том, чтобы перебрать все возможные пары параметров, а в том, чтобы подобрать пары, обеспечивающие максимально эффективную проверку при минимальном количестве выполняемых тестов.

Тестирование каждого выбора (Each choice testing): эта стратегия означает, что каждое значение каждого конкретного параметра должно использоваться как минимум один раз в тестовом наборе. Таким образом, полученное количество случаев будет равно количеству значений параметра с наибольшим диапазоном. Каждый выбор - это минимальная стратегия покрытия.



Тестирование базового выбора (Base choice testing): для этой стратегии мы должны определить наши базовые значения для каждого параметра. Это могут быть самые распространенные, самые маленькие / самые большие, самые простые или значения по умолчанию. После того, как мы сделали наш базовый выбор, мы должны изменять значение каждого параметра по одному, сохраняя при этом значения других параметров фиксированными при базовом выборе. Пусть a, 2 и Y будут нашим базовым выбором. Тогда кейсы будут:



Так же можно использовать **PICT**- генерирует компактный набор элементов. В cmd- cd c:\PICT pict abc.txt > abc.xml

**Тестовые артефакты (тестовая документация)**

**Test plan**- документ описывающий весь объём работ по тестированию, начиная с описания объекта, стратегии, расписания, критериев начала и окончания тестирования, до необходимого оборудования, знаний, а так же оценки рисков с вариантами решения.

Шаблон тест плана:

* Introduction- что за документ, что описывает.
* Test items- что будет покрыто тест планом.
* Features to be tested
* features not be tested
* Approach- стратегия тестирования(какие виды, модели)
* Item pass/fair criteria- критерии, согласно которым завершаем или не завершаем тест
* Suspension criteria and resumption requirements- критерии начала и окончания определённых стадий тестирования.
* Test deliverables- список документов по итогу проделанной работы
* Test tasks- список задач, которые нужно выполнить преде/в процессе работы(создать окружение, выполнить тест кейсы, создать автотесты)
* Environmental needs- необходимое оборудование
* Responsibilites- кто за что отвечает
* Staffing and training need- необходимое в обучении либо необходимые специалисты

**Test cases(тестовый случай)**- это артефакт, описывающий совокупность шагов, конкретных условий и параметров, необходимых для проверки, реализации тестируемой функции.

Отвечает на вопросы- как мы будем тестировать?

**Обязательно должен содержать следующую информацию:**

**Уникальный идентификатор тест-кейса** — необходим для удобной организации хранения и навигации по нашим тест-наборам.

**Название** — основная тема, или идея тест-кейса. Кратное описание его сути.

**Предусловия** — описание условий, которые не имеют прямого отношения к проверяемому функционалу, но должны быть выполнены.

Например, оставить комментарий на вашем портале может только зарегистрированный пользователь. Значит для тест-кейса «Создание комментария» будет необходимо выполнение предусловия «пользователь зарегистрирован», и «пользователь авторизован»

**Шаги** — описание последовательности действий, которая должна привести нас к ожидаемому результату

**Ожидаемый результат** — результат: что мы ожидаем увидеть после выполнения шагов.

В теории рекомендуется придерживаться правила- не указывать фактический результат, но если есть статус failed- это иногда может потребоваться.

**Рекомендации при создании test cases**

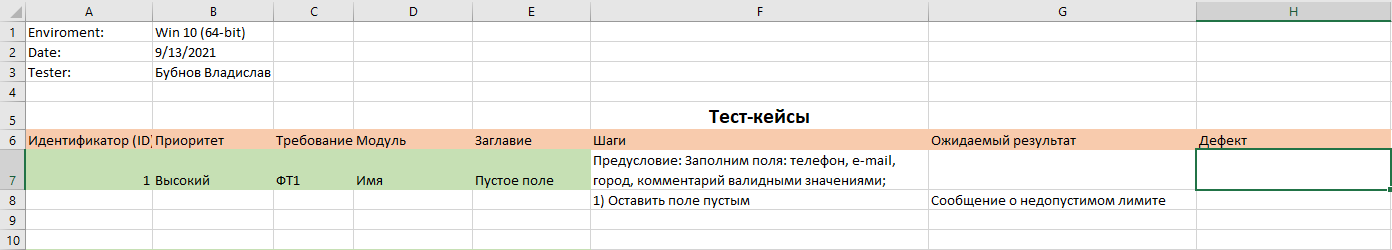
1. Правило. Название тест кейса должно быть уникальным. Важно что бы из названия было понятно что он будет проверять.
2. Текст кейсы должны быть независимы друг от друга. Нужно что бы было неважно в какой последовательности их проводить. Потому что 1 и тот же тест кейс может входить в разные тестовые наборы.
3. Тест кейс не должен содержать информации не важной для его прохождения. Очевидные вещи можно не рассписывать подробно. А то что важно нужно рассписать
4. Если есть повторяющиеся шаги, которые можно объеденить одним действием или настройки, использующихся в нескольких тестовых кейсах- это может быть описано в отдельном документе
5. Желательно удалять данные созданные по ходу прохождения тест кейса. Это может быть описано в Pre condition

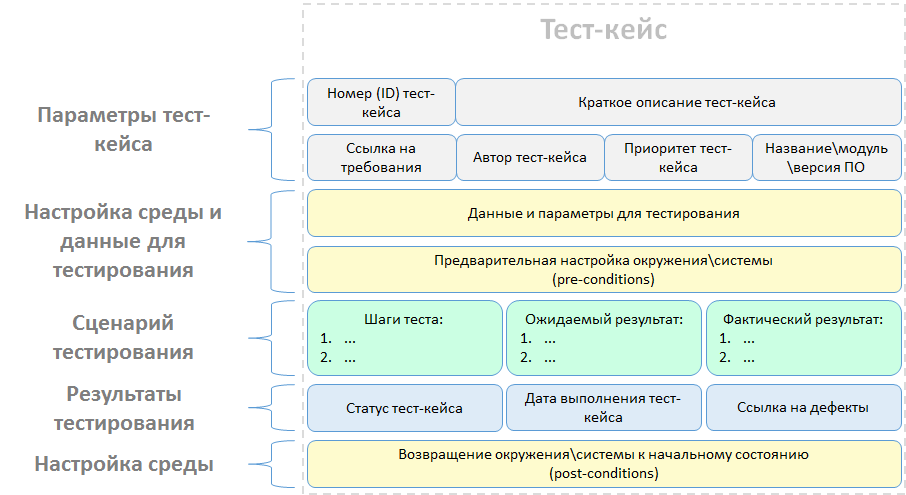
**Рекомендации по прохождению тест кейсов.**

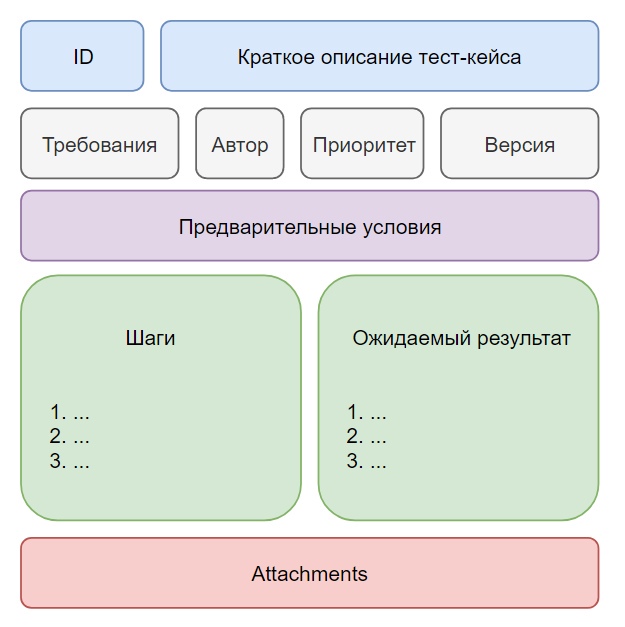
Если в тест кейсе не указаны конкретные входные данные, то проходится он по позитивному сценарию.

Даже если результаты прохождения тест кейсас отлиичаются от ожидаемых, нужно попытаться закончить прохождение тест кейса.

Не отклонятся от шагов.







Тест-кейс для авторизации на сайте

1.ID

Пусть будет №1, так как это наш первый тест-кейс

2. Краткое описание тест-кейса (Name)

Авторизация существующего пользователя.

3. Ссылка на требования

В нашем случае требований нет. Значит поле оставляем пустым

4. Автор тест-кейсы (Author)

Иванов И.

5. Приоритет (Priority)

Высокий, так как функциональность важная. В двух словах, чем важнее объект тестирования и проверки, тем выше приоритет.

6. Название/модуль/версия продукта (Component/Version)

Кейс относится напрямую к авторизации, следовательно этот модуль и укажем.

7. Предварительные условия (pre-condition)

Во-первых, нужно зайти на сайт по адресу https://msk.farfor.ru. Во-вторых, пользователь должен существовать и быть не залогинен.

8. Шаги (steps)

1) Вводим в поле телефона “+7 900 000-00-00”,

2) Вводим в поле password пароль “123”,

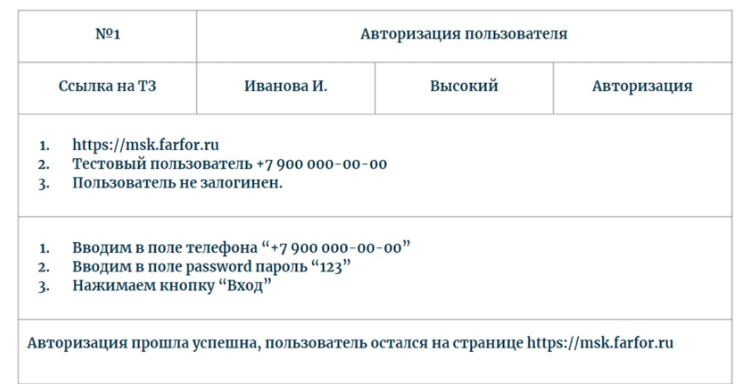
3) Нажимаем кнопку “Вход”.

9. Ожидаемый результат (expected result)

Авторизация прошла успешна, пользователь остался на странице https://msk.farfor.ru

10. Приложения (attachments)

В этот раз файлы нам не нужны, поэтому обойдемся без них.



**Test Suite** – это набор test cases, которые объеденены тем что относятся к одному тестироваемому модую

Отладка - этап, следующий после разработки и тестирования. Тестирование предназначено для поиска ошибок, а отладка - для поиска причины конкретной ошибки.

**Use Case(Юзкейс) —** это перечень действий, сценарий по которому пользователь взаимодействует с приложением, программой для выполнения какого-либо действия для достижения конкретной цели. Тестирование по юзкейсам проводится для того чтобы обнаружить дополнительные логические дыры и баги в приложении, которые сложно найти в тестировании индивидуальных модулей, частей приложения отдельно друг от друга.

**User story**- способ описания требований к разрабатываемой системе, на повседневном или деловом языке пользователя. Формируется по формуле: Как < роль, персона >, я <что-то хочу получить>, <с такой-то целью>. Я как Х, хочу Y, что- бы Z. X- персонаж, от его имени происходит повествование, для него будет строится функционал. Y- задача, действие, свойстов- необходимое персонажу. Z- бизнес ценность, которую получит.

**Check list**- документ описывающий что должно быть протестировано.

Цель чек листа- учесть все действия для наиболее полного покрытия тестами.

Рекомендации для составления.

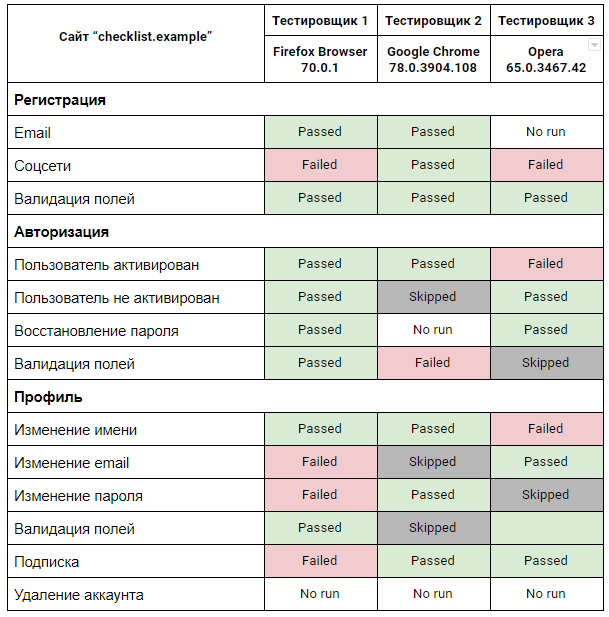
- Один пункт = одна проверка

- Пункты начинаются с существительных и глагола

- Объединяйте чек-листы в матрицы, где можно отразить не только сами проверки, но и условия проверки (платформа, версия продукта, сотрудник и т.п.)

- Лучше всего составлять тесты в том виде, который будет последовательный

Выполненные пункты отмечаются статусами, например: “Passed”, “Failed”, “Blocked”, “Skipped”, “Not run”.

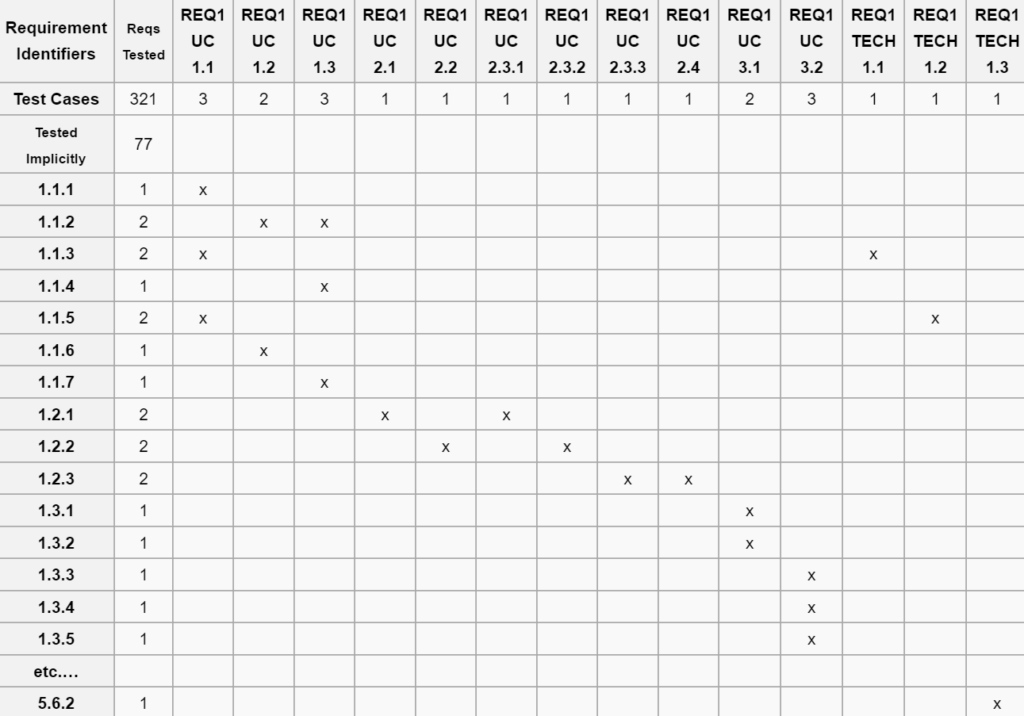


**Smoke check list**- список приоритетных проверок, подтверждающих успешную работоспособность продукта. Должен быть быстро проходим.

**Mind Mapping** - это способ представления информации с помощью диаграммы связей. Суть интеллект-карты - построить ассоциативные цепочки. В основе карты разума нужно поставить основную мысль, а дальше рисовать ветки, разбирая общую идею на более подробные.

Несмотря на то, что чаще всего чек-лист выглядит как перечень блоков, секций, страниц, других элементов, которые следует протестировать, он может быть и визуализирован в виде майнд карт.

**Матрица соответствия требований (RTM - Requirement Traceability Matrix)-** двумерная таблица, содержащая соответствие функциональных требований продукта и подготовленных тест кейсов. В заголовках колонок- расположены требования, а в заголовках строк- тестовые сценарии. На пересечении отметка- означает что требование покрыто тест кейсом. Используется QA для валидации покрытия требований по продукту- тестами. Цель- выяснить какие требования покрыты тестами, а какие нет. Избыточность тестов(одно функциональное требование покрыто большим количеством тестов).



**Модели разработки ПО/Методологии тестирования**

**Waterfall (каскадная модель, или «водопад»)** В этой модели разработка осуществляется поэтапно: каждая следующая стадия начинается только после того, как заканчивается предыдущая.



**V-образная модель (разработка через тестирование)** Это усовершенствованная каскадная модель, в которой заказчик с командой программистов одновременно составляют требования к системе и описывают, как будут тестировать ее на каждом этапе.



**Agile**- принцип гибкого подхода.

**Scrum- фреймворк.**

**Фреймворк** — это более сформированная методология со строгими правилами.

**Scrum- это итеративно-инкрементальная разработка**. Слово «итеративная» означает, что разработка разбивается на равные по длительности промежутки времени — спринты. Один спринт занимает от одной до четырех недель. Слово «инкрементальная» подразумевает, что в результате итерации получается новый, потенциально рабочий продукт, решающий бизнес-проблему. Такой продукт называется инкрементом продукта;

**Sprint backlog**- служит для предоставления отчёта о работе, которую команда проделала для достижения цели спринта.

**Scrum- это подход структуры.** Scrum подход делит рабочий процесс на равные спринты. Scrum- работа делится на этапы и к концу каждого этапа разрабатываемый продукт потенциально готов к применению , хотя и с неполным набором функций. Все планируемые в процессе разработки задачи- попадают в бек лог, а задачи на конкретный этап(спринт) в беклог спринта.

**Scrum состоит из ролей**: человек со стороны заказчиков — product owner, или владелец продукта, скрам мастер, команда. Процессы- планировени спринта, обзор спринта, Ретроспектива, скрам митинг, спринт

Задачи не меняются во время спринта- должно быть сделано любой ценой до окончания спринта. То что не сделалось- переносится на след. спринт.

**Ретроспектива**- это анализ прошедших итераций и релизов, позволяет повысить эффективности дальнейшей работы, улучшить результаты тестирования и оптимизировать свои задачи.

Весь период разработки разбит на промежутки времени — спринты. Длина спринта устанавливается в начале проекта и меняется только в том случае, если всплывают неучтенные детали, мешающие уложиться в заданные рамки

Ежедневные короткие SCRUM-митинги. Они дают понять, как движется процесс, а команда в курсе того, идут ли они к цели спринта или нет

Команда обычно состоит из 5-9 человек.

**Kanban**.

Команда ведет работу с помощью виртуальной доски, которая разбита на этапы проекта. Каждый участник видит, какие задачи находятся в работе, какие — застряли на одном из этапов, а какие уже дошли до его столбца и требуют внимания. В отличие от scrum, в Kanban можно взять срочные задачи в разработку сразу, не дожидаясь начала следующего спринта. Kanban удобно использовать не только в работе, но и в личных целях — распределять собственные планы или задачи семьи на выходные, наглядно отслеживать прогресс.

**Kanban**- это принцип “подход баланса”. Его задача сбалансировать разных специалистов. Вся команда едина, нет ролей, владельца и scrum мастера. **Главный принцип**- это среднее прохождение задачи на доске. Задача прошла быстро- команда работала продуктивно. Задача затянулась- надо думать, почему и где возникли задержки. Kanban- даёт больше гибкости, если под гибкостью понимать частоту смены приоритетов.

1. **Функциональное тестирование: используется для проверки того, соответствует ли ваш продукт спецификациям, а также функциональным требованиям, которые вы наметили для него в документации по разработке. Включает в себя:**

* Проверка, что все ссылки на ваших веб-страницах работают правильно и что нет битых ссылок.
* Текстовые формы работают как положено.
* Тестовые куки работают как положено.
* Тест бизнес-воркфлоу - это будет включать в себя
  + Тестирование вашего end-to-end workflow / бизнес-сценариев
  + Также проверьте отрицательные сценарии, чтобы при выполнении пользователем неожиданного шага в веб-приложении отображалось соответствующее сообщение об ошибке или справка.
* Примеры функциональных тест-кейсов:
  + Все обязательные поля должны быть валидированы.
  + Звездочка должна отображаться для всех обязательных полей.
  + Не должно отображаться сообщение об ошибке для дополнительных полей.
  + Проверьте, что високосные годы проверены правильно и не вызывают ошибок.
  + Числовые поля не должны принимать буквы и должно отображаться соответствующее сообщение об ошибке.
  + Проверьте наличие отрицательных чисел, если это разрешено для числовых полей.
  + Тестовое деление на ноль должно быть правильно обработано.
  + Проверьте максимальную длину каждого поля, чтобы убедиться, что данные не усекаются.
  + Тест всплывающего сообщения («Это поле ограничено 500 символами») должно отображаться, если данные достигают максимального размера поля.
  + Проверьте, должно ли отображаться подтверждающее сообщение для операций обновления и удаления.
  + Величины должны быть в подходящем формате.
  + Проверьте все поля ввода на ввод специальных символов.
  + Проверьте функциональность тайм-аута.
  + Проверьте функциональность сортировок.
  + Проверьте, что FAQ и Политика конфиденциальности четко определены и доступны для пользователей.
  + Проверьте, все ли работает и не перенаправляется ли пользователь на страницу ошибки.
  + Все загруженные документы открываются правильно.
  + Пользователь должен иметь возможность скачать загруженные файлы.
  + Проверьте функциональность электронной почты системы. Тестируемый скрипт корректно работает в разных браузерах (IE, Firefox, Chrome, Safari и Opera).
  + Проверьте, что произойдет, если пользователь удалит файлы cookie, находясь на сайте.
  + Проверьте, что произойдет, если пользователь удалит файлы cookie после посещения сайта.

1. **Юзабилити-тестирование стало важной частью любого веб-проекта. Его могут провести тестировщики или небольшая фокус-группа, похожая на целевую аудиторию веб-приложения.**

* Навигация:
  + Меню, кнопки или ссылки на разные страницы вашего сайта должны быть легко видны и согласованы на всех веб-страницах.
* Проверьте содержимое:
  + Содержание должно быть разборчивым, без орфографических или грамматических ошибок.
  + Изображения, если они присутствуют, должны содержать «альтернативный» текст
* Примеры тестов юзабилити:
  + Содержание веб-страницы должно быть правильным без каких-либо орфографических или грамматических ошибок
  + Все шрифты должны быть в соответствии с требованиями.
  + Весь текст должен быть правильно выровнен.
  + Все сообщения об ошибках должны быть правильными без каких-либо орфографических или грамматических ошибок, а сообщение об ошибке должно соответствовать метке поля.
  + Текст подсказки должен быть там для каждого поля.
  + Все поля должны быть правильно выровнены.
  + Должно быть достаточно места между метками полей, столбцами, строками и сообщениями об ошибках.
  + Все кнопки должны быть в стандартном формате и размере.
  + Домашняя ссылка должна быть на каждой странице.
  + Отключенные поля должны быть недоступны.
  + Проверьте наличие битых ссылок и изображений.
  + Сообщение о подтверждении должно отображаться для любого вида операции обновления и удаления. Проверить сайт на разных разрешениях (640 х 480, 600х800 и т. д. )
  + Убедитесь, что вкладка должна работать правильно.
  + Полоса прокрутки должна появляться только при необходимости.
  + Если при отправке появляется сообщение об ошибке, информация, заполненная пользователем, должна быть там.
  + Название должно отображаться на каждой веб-странице
  + Все поля (текстовое поле, раскрывающийся список, переключатель и т. д. ) И кнопки должны быть доступны с помощью сочетаний клавиш, и пользователь должен иметь возможность выполнять все операции с помощью клавиатуры.
  + Проверьте, не усекаются ли выпадающие данные из-за размера поля.
  + Также проверьте, жестко ли закодированы или управляются данные через администратора.

1. **Тестирование интерфейсов: Здесь тестируются три области: приложение, веб-сервер и сервер базы данных.**

* Приложение: тестовые запросы правильно отправляются в базу данных и вывод на стороне клиента отображается правильно. Ошибки, если таковые имеются, должны быть обнаружены приложением и должны отображаться только администратору, а не конечному пользователю.
* Веб-сервер: тестовый веб-сервер обрабатывает все запросы приложений без какого-либо отказа в обслуживании.
* Сервер базы данных: убедитесь, что запросы, отправленные в базу данных, дают ожидаемые результаты. Проверьте реакцию системы, когда невозможно установить соединение между тремя уровнями (Приложение, Интернет и База данных) и соответствующее сообщение отображается конечному пользователю.

1. **Тестирование базы данных: База данных является одним из важнейших компонентов вашего веб-приложения, и необходимо тщательно провести тестирование. Тестирование будет включать в себя:**

* Проверьте, отображаются ли какие-либо ошибки при выполнении запросов
* Целостность данных поддерживается при создании, обновлении или удалении данных в базе данных.
* Проверьте время ответа на запросы.
* Тестовые данные, полученные из вашей базы данных, точно отображаются в вашем веб-приложении.
* Примеры тест-кейсов для тестирования базы данных:
  + Проверьте имя базы данных: имя базы данных должно соответствовать спецификациям.
  + Проверьте таблицы, столбцы, типы столбцов и значения по умолчанию: все должно соответствовать спецификациям.
  + Проверьте, допускает ли столбец null значение или нет.
  + Проверьте первичный и внешний ключ каждой таблицы.
  + Проверьте хранимую процедуру:
  + Проверьте, установлена ​​ли сохраненная процедура или нет.
  + Проверьте имя хранимой процедуры
  + Проверьте имена параметров, типы и количество параметров.
  + Проверьте требуемые параметры.
  + Проверьте хранимую процедуру, удалив некоторые параметры
  + Проверьте, когда выход равен нулю, это должно повлиять на нулевые записи.
  + Проверьте хранимую процедуру, написав простые запросы SQL.
  + Проверьте, возвращает ли хранимая процедура значения
  + Проверьте хранимую процедуру с образцами входных данных.
  + Проверьте поведение каждого флага в таблице.
  + Убедитесь, что данные правильно сохраняются в базе данных после каждой отправки страницы.
  + Проверьте данные, если выполняются операции DML (Обновить, удалить и вставить).
  + Проверьте длину каждого поля: длина поля на Frontend и backend должна быть одинаковой.
  + Проверьте имена баз данных QA, UAT и production. Имена должны быть уникальными.
  + Проверьте зашифрованные данные в базе данных.
  + Проверьте размер базы данных.
  + Также проверьте время ответа каждого выполненного запроса.
  + Проверьте данные, отображаемые на Frontend, и убедитесь, что они совпадают с backend.
  + Проверьте достоверность данных, вставив неверные данные в базу данных.
  + Проверьте триггеры.

1. **Тестирование на совместимость. Тесты на совместимость гарантируют, что ваше веб-приложение правильно отображается на разных устройствах.**

* Вам нужно проверить, правильно ли отображается ваше веб-приложение в браузерах, работает ли JavaScript, AJAX и аутентификация нормально. Вы также можете проверить совместимость мобильного браузера. Рендеринг веб-элементов, таких как кнопки, текстовые поля и т. д. , изменяется с изменением в операционной системе. Убедитесь, что ваш сайт работает нормально для различных комбинаций операционных систем, таких как Windows, Linux, Mac и браузеров, таких как Firefox, Internet Explorer, Safari и т. д.
* Примеры тестов на совместимость:
  + Протестируйте сайт в разных браузерах (IE, Firefox, Chrome, Safari и Opera) и убедитесь, что сайт отображается правильно.
  + Используемая версия HTML совместима с соответствующими версиями браузера.
  + Проверьте правильность отображения изображений в разных браузерах.
  + Протестируйте шрифты, которые можно использовать в разных браузерах.
  + Протестируйте код Javascript в разных браузерах.
  + Проверьте анимированные GIF-файлы в разных браузерах.

1. **Тестирование производительности: Это нужно, чтобы обеспечить работу вашего сайта при любых нагрузках. Деятельность по тестированию ПО будет включать, но не ограничиваться:**

* Время отклика приложения сайта на разных скоростях соединения
* Нагрузочное тестирование вашего веб-приложения, чтобы определить его поведение при нормальной и пиковой нагрузке.
* Стресс-тест вашего веб-сайта, чтобы определить его точку остановки при превышении нормальных нагрузок в пиковое время.
* Проверьте, происходит ли сбой из-за пиковой нагрузки, как сайт восстанавливается после такого события, убедитесь, что методы оптимизации, такие как сжатие gzip и кэш включены, чтобы сократить время загрузки

1. **Тестирование безопасности жизненно важно для сайта электронной коммерции, который хранит конфиденциальную информацию о клиентах, например, кредитные карты. Деятельность по тестированию будет включать:**

* Проверка несанкционированного доступа к защищенным страницам
* Запрещенные файлы не должны быть загружаемыми без соответствующего доступа
* Сессии автоматически прекращаются после длительного отсутствия активности пользователя
* При использовании SSL-сертификатов веб-сайт должен перенаправить на зашифрованные SSL-страницы.
* Примеры тестовых сценариев для тестирования безопасности:
  + Убедитесь, что веб-страница, содержащая важные данные, такие как пароль, номера кредитных карт, секретные ответы на секретный вопрос и т. д. , Должна быть отправлена ​​через HTTPS (SSL).
  + Убедитесь, что важная информация, такая как пароль, номера кредитных карт и т. д. , Должна отображаться в зашифрованном виде.
  + Правила проверки пароля применяются на всех страницах аутентификации, таких как Регистрация, забытый пароль, смена пароля.
  + Убедитесь, что, если пароль изменен, пользователь не должен иметь возможность войти со старым паролем. Убедитесь, что сообщения об ошибках не должны отображать важную информацию.
  + Убедитесь, что, если пользователь вышел из системы или сеанс пользователя истек, пользователь не должен перемещаться по сайту авторизованным.
  + Проверьте доступ к защищенным и незащищенным веб-страницам напрямую без входа в систему.
  + Убедитесь, что опция «Просмотр исходного кода» отключена и не должна быть видна пользователю. Убедитесь, что учетная запись пользователя заблокирована, если пользователь вводит неправильный пароль несколько раз.
  + Убедитесь, что куки не должны хранить пароли.
  + Убедитесь, что, если какая-либо функция не работает, система не должна отображать информацию о приложении, сервере или базе данных. Вместо этого она должна отображать пользовательскую страницу ошибки.
  + Проверьте атаки SQL-инъекций.
  + Проверьте роли пользователей и их права. Например, запрашивающая сторона не должна иметь доступа к странице администратора.
  + Убедитесь, что важные операции записаны в файлы журналов, и эта информация должна быть отслеживаемой.
  + Убедитесь, что значения сеанса находятся в зашифрованном формате в адресной строке.
  + Убедитесь, что информация о файлах cookie хранится в зашифрованном формате.
  + Проверьте приложение на брутфорс-атаки