**1. Что такое тестирование**

**Тестирование ПО (software testing) -** это процесс поиска несоответствий между фактическим и ожидаемым результатом работы программы, проводиться на определенном наборе тестов.

**Требования** - это документ, содержащий в себе описание того, что должно быть реализовано в программном продукте.

**Баг (дефект)** - это отклонение фактического результата работы ПО от ожидаемого.

**Тестировщик** - это специалист по обеспечению качества, проверяет, чтобы продукт работал правильно, согласно требованиям заказчика и не было отклонения от бизнес логики продукта.

**STLC** - (*Software testing life cycle)*

1. **Анализ требований -** На данном этапе, мы анализируем наши требования на логичность и отсутствие противоречия в требованиях друг другу.
2. **Тестовое планирование -** Оно включает в себя составление плана тестирования, в котором будут указаны сроки, способы и методы тестирования.
3. **Написание тестовых сценариев -** Очень важный этап, в котором мы производим написание тестовых сценариев тестирования программного продукта, то есть сценариев и вариантов комбинаций исходных данных и действий.
4. **Выполнение тестов -** На данном этапе происходит сам процесс тестирования программного продукта.
5. **Завершающая фаза -** Если в результате выполнения нашего тестирования, тесты выполнены успешно, то происходит передача протестированного функционала в релиз, если же нет, то возвращаем в разработку, для дальнейшего исправления найденных дефектов.

**SDLC** – (*Software development life cycle*)

1. **Требования** - заказчик встречается с руководителем проекта и бизнес аналитиками из команды разработки, обсуждают требования к продукту и составляют документ **ТРЕБОВАНИЯ**. В данных требованиях описывается сам продукт, что он должен делать – то есть его функциональные свойства, как он должен это делать – его нефункциональные свойства, на каких платформах он должен работать, сроки и т.д
2. **Проектирование -** проектирование осуществляется как графической части нашего продукта (или как принято называть frontend) – та которую видит наш конечный пользователь, то есть мы с вами, простые пользователи, так и серверная часть, то есть backend – это сервера, базы данных и т.д, данную часть конечный пользователь уже не видит.

## **Разработка -** В данном процессе участвуют Разработчики и DevOps. Они разворачивают базы данных, стенды (серверы) для разработки, интеграционные сервера и production сервер. CI/CD и Development.

1. **Тестирование** - Происходит тестирование нового функционала, если в результате него обнаружились отклонения от той бизнес-логики, которая была заложена в продукт, или нарушение в самой системе интеграции и работы функционала, то тестировщик возвращает продукт обратно на стадию разработки, подробно описав все нюансы. Если же продукт работает исправно, то осуществляет переход на стадию релиза.
2. **Релиз и поддержка -** в процессе релиза происходит загрузка рабочего кода программного продукта, на пользовательский сервер, production. Которым уже пользуется конечный пользователь, то есть мы с вами.

И наконец поддержка. Это помощь нашим конечным пользователям и заказчику, в поддержании надежности и работоспособности нашего программного продукта.

**QA (Quality assurance**) или по-русски «обеспечение качества». Это специалист, который работает над улучшением процесса разработки ПО на всех уровнях, для того, чтоб предотвратить появление дефектов, то есть оптимизация качества продукта. От написания тестовой документации, до релиза. Это включает в себя обучение персонала, подбор инструментов, формат написания документации, как будет проходить тестирования на различных стендах, приемочное тестирование и т.д.

**QC (Quality Control)** или контроль качества. Это специалисты, которые, уже реализуют данную стратегию: пишут тестовую документацию, производят тестирование. Они проверяют соответствует программный продукт требованиям или нет.

**Цель тестирования:**

1. Нахождение дефектов до того, как их найдут конечные пользователи.
2. Проверка ПО требованиям заказчика.

**Качество ПО -** способность программного продукта соответствовать потребностям конечного пользователя и заказчика ПО.

**2. Классификация и принципы тестирования**

**7 основных принципов тестирования**

1. **Тестирование демонстрирует наличие дефектов**

Тестирование только снижает вероятность наличия дефектов, которые находятся в программном продукте, но не гарантирует их отсутствия.

2. **Исчерпывающее тестирование**

Полное тестирование, с использованием всех начальных комбинаций данных и предусловий, физически невыполнимо.

3. **Раннее тестирование**

Для того, чтобы найти дефекты как можно раньше, следует начинать тестирование на ранних стадиях жизненного цикла разработки ПО.

4. **Скопление дефектов**

Большая часть дефектов находится в ограниченном количестве модулей. Так называемый принцип 80% на 20%.

5. **Парадокс пестицида**

Если в тестировании продукта использовать один и тот же сценарий снова и снова, то со временем этот набор тестов

перестанет выявлять новые баги.

6. **Тестирование зависит от контекста**

Тестирование проводится по-разному, в зависимости от контекста. Например, банковский софт, в котором критически

важна безопасность, тестируется иначе, чем новостной портал.

7. **Заблуждение об отсутствии ошибок**

Отсутствие найденных дефектов при тестировании не всегда означает готовность продукта к релизу. Продукт должен

быть удобен и приятен в использовании.

**Тестирование классифицируют по следующим видам:**

1. Классификация по уровню функционального тестирования
2. Классификация по уровню детализации приложения
3. Классификация по запуску кода на исполнение
4. Классификация в зависимости от исполнителей
5. Классификация по позитивности сценария
6. Классификация по цели тестирования

**Функциональное тестирование** - проверяет, соответствует ли все функциональные характеристики нашего ПО согласно требованиям и работают ли они правильно. К этому виду относятся unit, интеграционное и системное тестирование.

**Нефункциональное тестирование** *-* проверка нефункциональных аспектов программного приложения: производительность, удобство использования, надежность и т. д. Он предназначен для проверки готовности системы по нефункциональным параметрам, которые никогда не учитываются при функциональном тестировании.

**Виды функционального тестирования:**

**Дымовое тестирование (smoke test)** — тестирование, которое проводится после появления нового билда. Направлено на проверку готовности разработанного продукта к проведению расширенного тестирования и определения общего качества продукта. Дает ответ готово ли приложение к выполнению своих основных функций.

**Тестирование критического пути** (critical path) — направлено для проверки функциональности, используемой обычными пользователями во время их повседневной деятельности.

**Расширенное тестирование** (extended) — направлено на исследование всей заявленной в требованиях функциональности.

**Виды нефункционального тестирования:**

1. **Тестирование производительности** (performance testing) – комплекс видов тестирования целью которых является проверка работоспособности, стабильности, потребления ресурсов и других атрибутов в условиях других сценариев использования и нагрузок

-**Нагрузочное** **(load testing)** — определение или сбор показателей производительности и времени отклика программно-технической системы или устройства в ответ на внешний запрос с целью установления соответствия требованиям, предъявляемым к данной системе (устройству). – производительность при нормальных условиях, например одновременное присутствие до 1000 пользователей на сайте (то есть наша норма), в идеале конечно установить это значение именно при 1000 пользователях

-**Стресс** (stress testing)- экстремальная нагрузка, в разы превышающая норму

-**Стабильность** – проверка приложения при длительной работе

**-Объемное** (volume testing)– тестирование при увеличенных объемах обрабатываемых данных (например, 1000 пользователей отправляют одновременно сообщения, загружают файлы)

1. **Инсталляционное тестирование (installation testing)** — тестирование, направленное на проверку успешной установки и настройки, обновления или удаления приложения. К примеру, проверка устанавливается ли наше мобильное приложение через плей-маркет, когда мы скачиваем с браузера нашего смартфона, установка с флешки, с компьютера и т.д
2. **Проверка удобства (Usability**) – проверка на удобство, соотношение дизайна ПП к функциональности, цветового оформления, текста, обучаемость и т.д то на сколько наш продукт удобен конечному пользователю. Это очень важно, так как наш продукт должен быть удобен и интуитивно понятен нашему пользователю, для того, чтоб он хотел пользоваться им снова и снова
3. **Тестирование безопасности (security 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.
4. **Тестирование локализации( L10N) и интернационализации(L18N)** - процесс адаптации нашего ПО к языку и культуре клиента (перевод, дата и время, символика, единицы измерения, цветов, валюта). К примеру, в некоторых странах используют как единицу расчета сантиметры, метры, километры, а в других дюймы, футы, мили. На сайтах должна быть возможность автоматического перевода единицы измерения или предусмотрен такой функционал.

-на сколько наш продукт может в дальнейшем адаптироваться к той или иной локации – особенности языка, удобность перестройки под него. Например, в азиатских странах.

1. **Тестирование доступности(accessibility) –** доступность людям с ограниченными возможностями, необязательно инвалиды. Тут можно привести пример с возможностью просматривать субтитры, например в том же ютубе, если у вас нет наушников, а вы не хотите пользоваться звуком, то можете включить субтитры.

**По уровню детализации приложения:**

**Модульное тестирование или юнит-тестирование** (англ. unit testing) — проводится для тестирования какого-либо одного, логически выделенного и изолированного элемента (модуля) системы в коде. Проводится самими разработчиками, так как предполагает полный доступ к коду.

**Интеграционное тестирование** (Integration testing) — это тестирование, направленное на проверку корректности взаимодействия нескольких модулей, объединенных в единое целое. Это взаимодействие компонентов, модулей. Тестирование части системы, состоящей из 2-х и более модулей (например, как со страницы корзины интернет-магазина произвести оплату платежной системы). Другими словами, это проверка как отдельные модули одного приложения взаимодействуют между собой.

**Системное тестирование** (System testing) — это тестирование взаимодействия между всеми компонентами системы или разных систем между собой или тестирование интерфейсов, между которыми взаимодействует система. Полная проверка приложения, всех модулей, можно ли пройти весь бизнес путь, тут это схоже с дымовым тестированием.

**Приемочное тестирование** (acceptance test ) — это тестирование на сдачи приемки программного продукта Заказчику или его конкретной части.

1. Пользовательское приемочное тестирование (User Acceptance Testing) – перед релизом собирается группа конечных пользователей, тестируется основной функционал, при наличии дефектов - устраняются.
2. Эксплуатационное (operational acceptance testing) – производится пользователем или администратором, производится тестирование резервного копирования, аварийное восстановление системы, безопасность ПО и т.д.

**По запуску кода на исполнение:**

**Статическое тестирование -** это тестирование программного продукта без запуска кода на исполнение. Это процесс обнаружения и устранения ошибок и дефектов в различных сопроводительных документах (например, требованиях к программному обеспечению).

**Динамическое тестирование** — тестирование проводится на работающей системе, не может быть осуществлено без запуска программного кода приложения. Запускаться на исполнение может как код, всего приложения целиком (системное тестирование), так и код нескольких взаимосвязанных частей (интеграционное тестирование), отдельных частей (модульное или компонентное тестирование) и даже отдельные участки кода.

**Регрессионное тестирование** – это тестирование ранее разработанного функционала**,** с целью удостовериться, что после появления новой функциональности, не повлияло на прежнюю функциональность. Другими словами, работает ли наш старый функционал как должен, после появления нового функционала.

**Re-test –** проверка правильности исправления дефекта.

**По позитивности сценария:**

Позитивное тестирование – это тестирование с применением сценариев, в которых ожидается нормальное, правильное поведение системы. С его помощью мы можем определить, что система делает то, для чего и была создана. То есть тестирование с корректным использованием данных и выполнение шагов, соответствующих бизнес-логике продукта.

Негативное тестирование - это тестирование с применением сценариев в которых ожидается нештатное поведение системы, неверное. Когда минимум одно используемое данное не корректно.

Позитивное тестирование должно нам всегда давать результат в виде отсутствия багов.

Негативные проверки могут дать 2 результата:

1. На данный ввод у продукта есть ответ в виде сообщения/контроля.

2. Система не знает, как реагировать на введенные данные.

Чтобы в зависимости от ситуации, верно, расставлять приоритеты в тестировании. Сперва мы производим позитивное тестирование, а далее негативное. Ведь сперва мы должны убедиться, что система работает согласно своей бизнес логике.

Сначала мы проверяем работу системы, когда наш условный пользователь работает с системой «правильно», то есть вводит корректные данные и делает действия которые соответствуют логике продукта. А уже потом приступаем к проверке отклика системы на пользователя, который допускает различные ошибки (ввод неверных данных). И наша система должна быть готова ответить на неверный запрос. Это и есть цель негативного тестирования.

**3. Техники тест-дизайна**

**Тест дизайн** - этап процесса тестирования ПО, на котором проектируются и создаются тест кейсы.

**Классы эквивалентности** - это разделение функционала или данных на определенные наборы, с которыми тестируемое приложение должно работать одинаково. **Граничные значения** - это значения, в которых один класс эквивалентности переходит в другой.



## **State & Transition Diagram (сокращенно S&T) —** схема состояний и переходов. Техника для визуализации ТЗ. Она наглядно показывает, как некий объект переходит из одного состояния в другое.

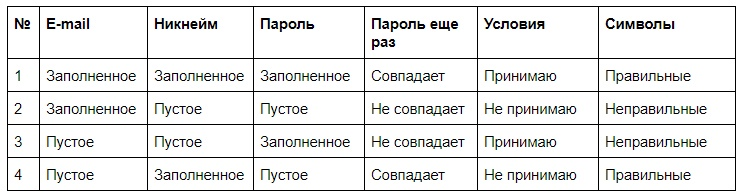
## 

## **Таблица принятия решений (decision table) -** способ компактного представления модели со сложной логикой; инструмент для упорядочения сложных бизнес требований, которые должны быть реализованы в продукте. Это взаимосвязь между множеством условий действий.

## 

**Попарное тестирование (pairwise testing)** — Эта техника используется, когда нам необходимо комбинировать очень много различный вариантов входных данных. Цель ее состоит в том, чтобы сократить количество полученных тестов, но при этом сохранить качественное покрытие.





**4. Методологии разработки ПО**

**Методология разработки ПО –** это процесс описания того, как определенный продукт будет разрабатываться, то есть один из способов организации коллективной разработки.

**Водопадная модель** - каждый этап разработки последовательного цикла продолжает предыдущий. По факту это наш Жизненный цикл разработки программного продукта – SDLC, который мы обсуждали с вами ранее.

Данная модель подходит под долголетние проекты в крупных компаниях, когда есть возможность первоначально составить требования, полноценные, включить в них все и на протяжении всего процесса разработки их не менять, подготовить дизайн-проект, написать полностью код проекта и только потом его протестировать. Это очень важно, тестирование идет не параллельно с разработкой, а только после ее полнейшего завершения. Далее идет релиз, всего продукта, а не поэтапно и поддержка.

**V-образная модель** - этомодифицированная версия каскадной – тестирование начинается со стадии написания требований и происходит на каждом этапе.

**Когда использовать V-модель?**

* Если требуется тщательное тестирование продукта;
* Для малых и средних проектов, где требования четко определены и фиксированы;
* В условиях доступности инженеров необходимой квалификации, особенно тестировщиков;

**Agile –** набор методов и принципов гибкого управления проектами

Основная составляющая данной методологии это - **Спринт** (итерация) – временной отрезок в 2-4 недели, в течении которого у нас создается часть продукта, которая соответствует тем требованиям, которые мы берем в начале спринта. В начале каждого спринта мы набираем часть задач, которые должны реализовать за этот промежуток времени, **Agile** включает в себя две вариации: Kanban и Scrum

Kanban – это конвейерная схема. У нас имеется backlog, где хранятся задачи по приоритету и где они берутся и выполняются поочередно. Мы переносим задачу в разработку, тестируем ее и отправляем в релиз и пока она не будет выполнена, мы не берем следующую.

**Scrum**– это так же конвейерная схема, но в ней возможно производство одновременно нескольких задач. У нас имеется общий backlog задач, мы выбрали на этот спринт 10 штук (к примеру), они могут параллельно или последовательно, в зависимости от численности нашей команды разрабатываться.

Теперь сравним все 3 методологии:

На разработку 1 блока уходит месяц, следовательно если мы используем схему Водопадной модели или V-образную, то наш продукт будет готов через 10 месяцев и только после этого у нас начнут появляться пользователи, отзывы и пожелания от них, и на исправление или если Мы захотим внести что-то новое, то на это уйдет очень много времени. А если мы используем Agile, то наш продукт начнет функционировать уже со второго месяца, и к концу разработки проекта, мы будем иметь большое количество пользователей нашего продукта и иметь обратную связь от них, так же мы сможем в любой момент изменить направление нашего продукта, в ту которая будет нам выгоднее.