# Вопросы к экзамену



## 1) Что такое тестирование? Цели тестирования?

**Тестирование** — это процесс оценки качества программного обеспечения путем его проверки на соответствие заданным требованиям и выявления дефектов. Основная цель тестирования заключается в обнаружении ошибок, которые могут привести к сбоям в работе программы, и в обеспечении того,

что программное обеспечение работает в соответствии с ожиданиями пользователей и спецификациями.

**Цели тестирования**

1. **Обнаружение дефектов**: Основная цель тестирования — выявление ошибок и дефектов в программном обеспечении до его выпуска в эксплуатацию. Это помогает предотвратить возможные проблемы и сбои в работе системы.
2. **Проверка соответствия требованиям**: Тестирование помогает убедиться, что программное обеспечение соответствует всем заданным требованиям и спецификациям. Это включает проверку функциональности, производительности, безопасности и других аспектов.
3. **Повышение качества**: Тестирование способствует улучшению качества программного обеспечения путем выявления и устранения дефектов на ранних стадиях разработки. Это помогает снизить затраты на исправление ошибок в будущем и улучшить общее качество продукта.
4. **Оценка производительности**: Тестирование производительности (load testing) позволяет оценить, как система справляется с нагрузкой, и выявить возможные узкие места и проблемы с производительностью.
5. **Обеспечение надежности**: Тестирование помогает убедиться, что программное обеспечение работает надежно и стабильно в различных условиях эксплуатации. Это включает тестирование на отказоустойчивость и восстановление после сбоев (Failover and Recovery Testing).
6. **Удовлетворение пользователей**: В конечном итоге, тестирование направлено на обеспечение того, чтобы конечные пользователи были довольны качеством и функциональностью программного обеспечения. Это включает проверку удобства использования и соответствия ожиданиям пользователей(заказчика).

## 2) Что такое качество ПО?

**Качество программного обеспечения (ПО)** — это совокупность характеристик и свойств программного продукта, которые определяют его способность удовлетворять установленные и предполагаемые потребности

пользователей(заказчика). Качество ПО включает в себя несколько ключевых аспектов:

1. **Функциональность**: Программное обеспечение должно выполнять все заявленные функции и соответствовать требованиям спецификаций.
2. **Надежность**: ПО должно работать стабильно и без сбоев в различных условиях эксплуатации.
3. **Удобство использования**: Интерфейс и взаимодействие с ПО должны быть интуитивно понятными и удобными для пользователей.
4. **Производительность**: ПО должно эффективно использовать ресурсы системы и обеспечивать необходимую скорость выполнения операций.
5. **Безопасность**: ПО должно защищать данные пользователей и предотвращать несанкционированный доступ.
6. **Поддерживаемость**: ПО должно быть легко модифицируемым и поддерживаемым в течение всего жизненного цикла.
7. **Масштабируемость:** Способность ПО адаптироваться к росту нагрузки и расширению функциональности.

## 3) Кто такой тестировщик? QA, QC и специалист по тестированию

**Тестировщик** — это специалист, занимающийся проверкой качества программного обеспечения. Основная задача тестировщика — выявление дефектов и ошибок в ПО, а также проверка его соответствия требованиям и спецификациям.

**QA (Quality Assurance)** — это процесс обеспечения качества, который включает в себя все виды деятельности, направленные на улучшение процессов разработки и предотвращение дефектов на ранних стадиях. Специалисты по QA занимаются разработкой стандартов и процедур, которые помогают обеспечить высокое качество ПО на всех этапах его жизненного цикла.

**QC (Quality Control)** — это процесс контроля качества, который включает в себя проверку и тестирование готового продукта для выявления дефектов и

ошибок. Специалисты по QC проводят различные виды тестирования, чтобы убедиться, что ПО соответствует установленным требованиям и стандартам.

**Специалист по тестированию** — это профессионал, который непосредственно занимается тестированием программного обеспечения. Он разрабатывает тестовые сценарии, проводит тестирование, анализирует результаты и документирует найденные дефекты. Специалисты по тестированию могут работать как вручную, так и с использованием автоматизированных инструментов тестирования.

> 💡 QA фокусируется на предотвращении дефектов, а QC — на их выявлении. QA-инженеры работают на всех этапах разработки ПО, начиная с планирования и заканчивая выпуском продукта, а QC - тестируют готовый продукт

## 4) Чем занимается тестировщик?

**Тестировщик** занимается проверкой программного обеспечения на наличие дефектов и ошибок, а также проверкой его соответствия требованиям и спецификациям. Основные задачи тестировщика включают:

1. **Анализ требований**: Тестировщик изучает требования к программному обеспечению, чтобы понять, какие функции и характеристики должны быть проверены.
2. **Разработка тестовых сценариев**: Создание тестовых сценариев и тест-кейсов, которые описывают шаги для проверки различных аспектов ПО.
3. **Проведение тестирования**: Выполнение тестов, как вручную, так и с использованием автоматизированных инструментов, для выявления дефектов и ошибок.
4. **Документирование дефектов**: Запись найденных дефектов в систему отслеживания ошибок, включая подробное описание проблемы и шаги для её воспроизведения.
5. **Анализ результатов тестирования**: Оценка результатов тестирования и предоставление отчетов о качестве ПО команде разработки и менеджерам.

6. **Регрессионное тестирование**: Повторное тестирование ПО после исправления дефектов, чтобы убедиться, что исправления не вызвали новых проблем.
7. **Сотрудничество с командой разработки**: Взаимодействие с разработчиками, аналитиками и другими участниками проекта для обсуждения найденных дефектов и улучшения качества ПО.

## 5) Жизненный цикл тестирования

Жизненный цикл тестирования (Software Testing Life Cycle, STLC) – это последовательность этапов, которые проходит процесс тестирования ПО от начала до конца. Основные этапы STLC:

1. **Анализ требований:** Изучение требований к продукту, определение целей и задач тестирования.
2. **Планирование тестирования:** Разработка стратегии тестирования, определение ресурсов, сроков, методов и инструментов.
3. **Разработка тест-кейсов:** Создание детальных инструкций для проведения тестов.
4. **Настройка тестовой среды:** Подготовка окружения, в котором будет проводиться тестирование (установка ПО, настройка конфигурации).
5. **Выполнение тестов:** Проведение тестов согласно разработанным тест-кейсам, фиксация результатов.
6. **Отчет о дефектах:** Составление отчетов об обнаруженных ошибках и дефектах.
7. **Повторное тестирование:** Проверка исправленных дефектов, убеждение в том, что исправления не вызвали новых проблем.
8. **Завершение тестирования:** Анализ результатов тестирования, составление итогового отчета, оценка качества продукта.

## 6) 7 Принципов тестирования

- **Тестирование показывает наличие дефектов:** Тестирование может выявить наличие ошибок, но не может доказать их отсутствие. Цель

тестирования – найти как можно больше дефектов и снизить вероятность их появления в будущем.

- **Исчерпывающее тестирование невозможно:** Полностью протестировать все возможные комбинации входных данных и условий нереально. Вместо этого следует использовать анализ рисков и приоритизацию для определения наиболее важных тестовых сценариев.
- **Раннее тестирование:** Тестирование следует начинать как можно раньше в жизненном цикле разработки программного обеспечения (SDLC). Раннее выявление дефектов обходится дешевле и позволяет избежать накопления проблем.
- **Скопление дефектов:** Дефекты имеют тенденцию скапливаться в определенных модулях или областях приложения. Это может быть связано с архитектурными проблемами, сложностью кода или недостаточным вниманием к тестированию в этих областях.
- **Парадокс пестицида:** Если одни и те же тесты повторяются многократно, они перестают находить новые дефекты. Чтобы преодолеть этот парадокс, необходимо регулярно пересматривать и обновлять тестовые сценарии.
- **Тестирование зависит от контекста:** Не существует универсального подхода к тестированию. Методы тестирования, техники и инструменты должны быть адаптированы к конкретному проекту, типу приложения и требованиям к качеству.
- **Заблуждение об отсутствии ошибок:** Даже если программное обеспечение прошло все тесты, это не гарантирует, что оно будет удовлетворять потребности пользователей. Важно учитывать удобство использования, производительность и другие нефункциональные аспекты.

## 7) Нефункциональное тестирование

Нефункциональное тестирование (NFT) оценивает характеристики программного обеспечения, которые не связаны непосредственно с его функциональностью. NFT фокусируется на том, **как** система работает, а не на том, **что** она делает.

**Примеры нефункциональных требований:**

- Производительность: быстродействие, время отклика, пропускная способность, потребление ресурсов
- Надежность: устойчивость к сбоям, отказоустойчивость, возможность восстановления
- Удобство использования: интуитивность интерфейса, простота обучения, доступность
- Безопасность: защита от несанкционированного доступа, конфиденциальность данных, целостность
- Совместимость: способность работать на разных платформах, браузерах, устройствах

## 8) Функциональное тестирование

Функциональное тестирование (FT) проверяет соответствие программного обеспечения заявленным функциональным требованиям. FT фокусируется на том, **что** система делает, а не на том, **как** она это делает.

**Цели функционального тестирования:**

- **Подтверждение соответствия требованиям:** Убедиться, что все функции реализованы в соответствии со спецификацией.
- **Выявление дефектов:** Найти ошибки, которые приводят к некорректному поведению системы.
- **Повышение качества:** Улучшить надежность, стабильность и удобство использования программного обеспечения.
- **Снижение рисков:** Минимизировать вероятность появления проблем после выпуска продукта.
- **Увеличение доверия пользователей:** Продемонстрировать, что система работает должным образом и удовлетворяет их потребности.

**Примеры функциональных требований:**

- Регистрация и авторизация пользователей
- Поиск и фильтрация данных

- Оформление заказа и оплата
- Отправка сообщений и уведомлений
- Создание и редактирование контента

## 9) Что такое чек-лист. Как написать хороший чек лист?

**Чек-лист** — это документ, содержащий список проверок или действий, которые необходимо выполнить для проверки определенного аспекта программного обеспечения. Чек-листы используются для систематизации процесса тестирования и обеспечения того, что все важные проверки выполнены.

**Как написать хороший чек-лист?**

1. **Определите цель**: Четко определите, что именно вы хотите проверить с помощью чек-листа. Это может быть функциональность, производительность, безопасность и т.д.
2. **Разделите на категории**: Разделите чек-лист на логические категории или секции, чтобы упростить навигацию и использование.
3. **Будьте конкретны**: Каждая проверка должна быть четкой и конкретной. Избегайте общих формулировок.
4. **Используйте простые и понятные формулировки**: Чек-лист должен быть понятен всем членам команды, включая тех, кто не участвовал в его создании.
5. **Добавьте критерии выполнения**: Укажите, что считается успешным выполнением проверки (например, pass/fail).
6. **Обновляйте регулярно**: Чек-лист должен быть актуальным и обновляться по мере изменения требований и функциональности ПО.

Пример чек-листа

| № | Проверка | Ожидаемый результат | Проверка пройдена |
|---|---|---|---|
| 1 | Вход в gmail с верными данными | Успешный вход в Gmail | Да |

| | | | |
|---|---|---|---|
| 2 | Вход в gmail с верными данными почты и неверным паролем | Ошибка входа:Неверный пароль. Повторите попытку или нажмите на ссылку "Забыли пароль?" | Да |
| 3 | Вход в gmail с неверными данными почты и верным паролем | Ошибка входа:Не удалось найти аккаунт google | Да |
| 4 | Вход в gmail с верными данными почты и пустым паролем | Ошибка входа: Введите пароль | Да |
| 5 | Вход в gmail с несуществующим номером телефона | Ошибка входа:Не удалось найти аккаунт google | Да |

## 10) Что такое тест-кейс. Как написать хороший тест-кейс

**Тест-кейс** - это документ, описывающий последовательность действий, которые необходимо выполнить, чтобы проверить определенную функцию или требование. Тест-кейс обычно содержит следующие разделы:

**Как написать хороший тест-кейс:**

- **Идентификатор** (ID) - Уникальный номер тест-кейса
- **Название** (Title) — краткое и лаконичное описание сути теста.
- **Описание** (Description) — более подробное описание сути теста. *Необязательно*
- **Предварительные условия** (Preconditions) — действия и условия, которые необходимо выполнить перед началом проверки.
- **Шаги** (Steps to reproduce) — описание шагов для выполнения тест-кейса.
- **Ожидаемые результаты** (Expected results) — описание ожидаемых результатов проверок на каждом шаге.
- **Приоритет** (Priority) — важность теста в рамках тестирования по отношению к другим тестам.
- **Статус** (Status) - Результат выполнения тест-кейса

## 11) Отличия тест-кейса от чек-листа

### Кратко:

| **Характеристика** | **Тест-кейс** | **Чек-лист** |
|---|---|---|
| **Уровень детализации** | Подробный | Краткий |
| **Содержание** | Пошаговые инструкции | Список проверок |
| **Назначение** | Описание конкретных шагов проверки | Контроль выполнения проверок |

### Подробно:

| Характеристика | Тест-кейс | Чек-лист |
|---|---|---|
| Цель | Проверить конкретную функциональность или аспект ПО в определенных условиях. | Убедиться, что все необходимые пункты или действия выполнены в рамках определенного процесса или задачи. |
| Структура | Имеет строгую структуру: ID, название, описание шагов, предусловия, входные данные, ожидаемые результаты, фактические результаты, статус, приоритет. | Более свободный формат: список пунктов, которые нужно проверить или выполнить. |
| Детализация | Содержит детальное описание шагов и ожидаемых результатов, может включать скриншоты и другие вспомогательные материалы. | Пункты обычно сформулированы кратко и не содержат подробных инструкций. |
| Повторяемость | Предназначен для многократного выполнения в неизменном виде. | Может использоваться для разовых проверок или адаптироваться к различным ситуациям. |
| Применение | Используется для функционального, регрессионного, интеграционного и других видов тестирования, где | Применяется для исследовательского тестирования, smoke-тестов, приемочного тестирования, аудитов и других ситуациях, где важна полнота |

| | | |
|---|---|---|
| | требуется точность и повторяемость результатов. | охвата, а не строгая последовательность действий. |

## 12) Что такое тестирование, контроль качества, обеспечение качества

- **Тестирование ПО** – это процесс исследования, оценки и проверки программного обеспечения с целью выявления дефектов, ошибок или несоответствий требованиям. Тестирование помогает повысить качество ПО, снизить риски и убедиться, что оно работает должным образом.
- **Контроль качества** (Quality Control, QC) – это набор действий, направленных на выявление и устранение дефектов в готовой продукции или услуге. QC фокусируется на проверке соответствия продукта установленным стандартам и требованиям.
- **Обеспечение качества** (Quality Assurance, QA) – это более широкий процесс, включающий в себя планирование, разработку, внедрение и контроль мер, направленных на предотвращение появления дефектов и повышение качества продукции или услуги на всех этапах жизненного цикла. QA фокусируется на предотвращении проблем, а не только на их обнаружении и исправлении.

## 13) Системы управления тестами (Test Management Systems, TMS)

Системы управления тестами (TMS) – это программные инструменты, предназначенные для организации, планирования, выполнения и анализа процесса тестирования программного обеспечения. TMS помогают командам тестировщиков эффективно управлять тестовыми артефактами (тест-кейсами, требованиями, дефектами), отслеживать прогресс тестирования и принимать обоснованные решения на основе полученных данных.

**Основные функции TMS:**

- **Хранение и организация тестовых артефактов:** TMS предоставляют централизованное хранилище для всех тестовых данных, обеспечивая удобный доступ и структурированную организацию информации.

- **Планирование тестирования:** TMS позволяют создавать и управлять тестовыми планами, определять приоритеты и распределять задачи между участниками команды.
- **Выполнение тестов:** TMS поддерживают выполнение как ручных, так и автоматизированных тестов, предоставляя инструменты для регистрации результатов и создания отчетов.
- **Отслеживание дефектов:** TMS интегрируются с баг-трекинговыми системами, позволяя отслеживать жизненный цикл дефектов от обнаружения до исправления.
- **Анализ результатов:** TMS предоставляют инструменты для анализа результатов тестирования, выявления тенденций и оценки качества продукта.
- **Отчетность:** TMS генерируют разнообразные отчеты, которые помогают команде и заинтересованным сторонам оценить прогресс тестирования и принять решения.

**Популярные TMS:**

- **HP ALM (Micro Focus ALM)**
- **Jira (Xray, Zephyr)**
- **TestRail**
- **qTest**
- **PractiTest**
- **Test IT**

## 14) Термины и определения баг-трекинговой системы

> Без терминов из тест-кейсов, чек-листов и баг репортов

1. **Баг (Bug)**: Ошибка или дефект в программном обеспечении, который приводит к неправильной работе или сбоям.
2. **Тикет (Ticket)**: Запись в баг-трекинговой системе, содержащая информацию о баге, включая его описание, шаги для воспроизведения,

приоритет и статус.

3. **Серьезность (Severity)**: Влияние бага на работу системы. Может быть критической, серьезной, средней или незначительной.
4. **Назначение (Assignment)**: Назначение бага на конкретного разработчика или тестировщика для его исправления или проверки.
5. **Репродукция (Reproduction)**: Способность воспроизвести баг, следуя описанным шагам.
6. **Решение (Resolution)**: Описание того, как баг был исправлен или почему он был закрыт без исправления.

## Из лекции:

- Project: Указание, к какому проекту относится дефект
- Issue type: Указание типа создаваемого артефакта (Task, Bug, Story, Epic)
- Summary: Заголовок баг-репорта
- Description: Подробное описание дефекта (включая preconditions, steps to reproduce, expected result, actual result)
- Severity: Уровень серьезности дефекта
- Reporter: Автор баг-репорта
- Priority: Приоритет исправления дефекта
- Labels: Метки для группировки и классификации дефектов
- Environment: Описание аппаратных и программных конфигураций, в которой проявляется дефект
- Attachments: Прикрепление скриншотов и видеофайлов
- Linked issues: Связь с требованиями, связанными с обнаруженным багом
- Assignee: Ответственное лицо, которое будет исправлять баг
- Epic link: Ссылка на Epic
- Sprint: Указание итерации, в которой необходимо устранить баг

## 15) Виды тестирования

**По уровню тестирования:**

- **Модульное тестирование (Unit Testing):** Проверка отдельных компонентов или модулей кода изолированно друг от друга.
- **Интеграционное тестирование (Integration Testing):** Проверка взаимодействия между различными компонентами или модулями системы.
- **Системное тестирование (System Testing):** Проверка всей системы на соответствие требованиям в целом.
- **Приемочное тестирование (Acceptance Testing):** Проверка соответствия системы ожиданиям и потребностям пользователей.

**По степени автоматизации:**

- **Ручное тестирование (Manual Testing):** Тестирование, выполняемое человеком без использования специальных инструментов.
- **Автоматизированное тестирование (Automated Testing):** Тестирование, выполняемое с помощью специальных программных инструментов и скриптов.

**По типу тестируемой функциональности:**

- **Функциональное тестирование (Functional Testing):** Проверка соответствия системы заявленным функциональным требованиям.
- **Нефункциональное тестирование (Non-functional Testing):** Проверка нефункциональных характеристик системы, таких как производительность, надежность, удобство использования и безопасность.

**По знанию внутренней структуры системы:**

- **Тестирование белого ящика (White Box Testing):** Тестирование, основанное на знании внутренней структуры кода и логики работы системы.
- **Тестирование черного ящика (Black Box Testing):** Тестирование, основанное только на знании внешнего поведения системы, без учета ее

внутренней реализации.

- **Тестирование серого ящика (Gray Box Testing):** Комбинация методов белого и черного ящика, использующая ограниченные знания о внутренней структуре системы.

**Другие виды тестирования:**

- **Регрессионное тестирование (Regression Testing):** Повторное выполнение тестов после внесения изменений в систему, чтобы убедиться, что новые ошибки не появились.
- **Дымовое тестирование (Smoke Testing):** Быстрая проверка основных функций системы, чтобы убедиться, что она работает в целом.
- **Нагрузочное тестирование (Load Testing):** Проверка производительности системы под различными нагрузками.
- **Стресс-тестирование (Stress Testing):** Проверка устойчивости системы к экстремальным нагрузкам.

## 16) Баг, ошибка, отказ

**Баг (Bug)** — это дефект или ошибка в программном обеспечении, который приводит к неправильной работе или сбоям. Баги могут возникать из-за ошибок в коде, неправильных требований или проблем с дизайном системы.

**Ошибка (Error)** — это действие или решение, которое приводит к неправильному результату. Ошибки могут быть допущены разработчиками, тестировщиками или пользователями. Ошибка может привести к появлению багов в программном обеспечении.

**Отказ (Failure)** — это неспособность системы или компонента выполнять требуемую функцию в соответствии с заданными спецификациями. Отказ может быть вызван багом, ошибкой или внешними факторами, такими как сбои оборудования или сети.

**Примеры:**

- **Баг**: В форме регистрации пользователь не может ввести дату рождения, так как поле неактивно.

- **Ошибка**: Разработчик неправильно реализовал алгоритм сортировки, что приводит к некорректным результатам.
- **Отказ**: Приложение перестает отвечать на запросы пользователей из-за перегрузки сервера.

## 17) Что такое баг-репорт?

**Баг-репорт** — это документ, в котором тестировщик описывает неисправность в ПО, её причины и условия воспроизведения. Баг-репорты помогают разработчикам устранять баги и улучшать качество ПО

**Структура баг-репорта**

Баг-репорты обычно содержат следующие элементы:

1. Идентификатор (ID) - Уникальный номер баг-репорта *(Необязательно, но чаще указывается, чем нет)*
2. Заголовок (summary) — краткое описание проблемы.
3. Проблема (problem) — дополнение к заголовку, описывающее проблему.
4. Приоритет/важность (priority/severity) — атрибуты, определяющие очередность исправления и влияние на работоспособность приложения.
    - Severity: влияние на работоспособность
        - **Blocker:** Блокирует работу программы.
        - **Critical:** Нарушает работу основного функционала.
        - **Major:** Затрудняет работу основного функционала.
        - **Minor:** Незначительно влияет на функционал.
    - Priority: очередность исправления
        - **Critical: Критичная важность**
        - **High:** Высокая
        - **Medium:** Средняя
        - **Low:** Маленькая

> 💡 Иногда возникают ситуации с высоким приоритетом и низкой серьезностью
>
> 1. Кнопки перекрывают друг друга. Да, они кликабельны, но визуальное впечатление портится.
> 2. Логотип компании на главной странице содержит орфографическую ошибку. На функционал это вообще не влияет, но портит пользовательский опыт. Этому багу нужно сразу ставить высокий приоритет, несмотря на то, что на работоспособность продукта он никак не влияет.

5. Предусловия (preconditions) — начальные условия для воспроизведения бага.
6. Шаги воспроизведения (steps to reproduce) — конкретные действия для воспроизведения бага.
7. Ожидаемый результат (expected result) — ожидаемое поведение ПО.
8. Фактический результат (actual result) — фактическое поведение ПО при выполнении шагов воспроизведения.
9. Окружение (environment) — условия, в которых был обнаружен дефект (например, версия ОС, браузер и его версия).
    - **Dev** - Окружение для разработки
    - **Stage** - Окружение для стабильной версии
    - **Prod** - Окружения для продакшена
    - Версия ОС, Браузера, Тестируемого продукта
10. Другая информация, которая поможет пофиксить баг, воспроизвести в будущем и проверить.

## 18) Жизненный цикл баг-репорта

**Жизненный цикл баг-репорта** - это путь, который проходит баг-репорт (и баг вместе с ним) в баг-трекинговой системе, от создания до окончательного

закрытия.

> 💡 Жизненный цикл баг-репорта может варьироваться в зависимости от проекта, приоритета бага или решения команды.

> 💡 На проектах может использоваться разная терминология стадий, которые проходит баг-репорт.

Стадии жизненного цикла баг-репорта:

- Открыт (Open): Баг выявлен и заведен в баг-трекинговой системе. Может остаться в этом статусе или быть переведен в один из следующих: Отклонен, Отсрочен, Не является багом.
- (Rejected): Исправлению бага мешает ошибка в баг-репорте.
- Отсрочен (Deferred): Баг признан неприоритетным, его исправление переносится.
- Не является багом (Not a bug): Дефект не является функциональной возможностью приложения.
- В работе (In Progress): Разработчик получает информацию о баге. Может отклонить баг-репорт или начать исправление.
- Исправлен/На тестировании (Resolved/Testing): Разработчик исправил баг и передал его на повторную проверку. Если баг все еще существует, статус меняется на Переоткрыт (Reopen).
- Закрыт (Closed): Баг исправлен и больше не воспроизводится.

Основная работа тестировщика происходит на этапе Open, где заводится баг-репорт, и в Resolved/Testing.

## 19) Использование данных баг-репорта для улучшения качества разработки ПО

Данные баг-репорта играют ключевую роль в улучшении качества разработки программного обеспечения. Вот как они могут быть

использованы:

1. **Анализ дефектов**: Баг-репорты предоставляют подробную информацию о дефектах, включая их природу, частоту и серьезность. Анализ этих данных помогает выявить наиболее проблемные области в коде и сосредоточить усилия на их улучшении.
2. **Улучшение процессов разработки**: Данные баг-репортов могут указывать на систематические проблемы в процессе разработки, такие как недостаточное тестирование или нечеткие требования. Это позволяет команде пересмотреть и улучшить свои процессы, чтобы предотвратить появление подобных дефектов в будущем.
3. **Обучение и развитие команды**: Анализ баг-репортов помогает выявить области, в которых разработчики и тестировщики нуждаются в дополнительном обучении. Это может включать обучение новым методам тестирования, улучшение навыков кодирования или понимание требований.
4. **Приоритизация исправлений**: Баг-репорты помогают определить, какие дефекты требуют немедленного исправления, а какие могут быть отложены. Это позволяет эффективно распределять ресурсы и сосредоточиться на наиболее критичных проблемах.
5. **Отслеживание прогресса**: Использование баг-репортов для отслеживания прогресса в исправлении дефектов помогает команде видеть, насколько эффективно они справляются с проблемами и где требуется дополнительное внимание.

## 20) Что включает в себя тестовая документация

Тестовая документация — это совокупность документов, которые описывают процесс тестирования, его цели, методы и результаты. Она включает в себя следующие основные элементы:

1. **Тест-план**: Документ, описывающий стратегию тестирования, объем работ, ресурсы, график и критерии завершения тестирования. Тест-план определяет, что будет тестироваться, как и когда.
2. **Тест-кейсы**: Подробные сценарии тестирования, включающие шаги выполнения, ожидаемые результаты и фактические результаты. Тест-

кейсы помогают систематизировать процесс тестирования и обеспечивают воспроизводимость тестов.

3. **Чек-листы**: Списки проверок или действий, которые необходимо выполнить для проверки определенного аспекта ПО. Чек-листы используются для быстрого и поверхностного тестирования.
4. **Баг-репорты**: Документы, описывающие обнаруженные дефекты, включая их описание, шаги для воспроизведения, серьезность и приоритет. Баг-репорты служат средством коммуникации между тестировщиками и разработчиками.
5. **Отчеты о тестировании**: Документы, содержащие результаты тестирования, анализ дефектов, метрики качества и рекомендации по улучшению. Отчеты о тестировании помогают оценить текущее состояние ПО и принять решения о его готовности к выпуску.

## 21) Что из себя представляет отчет о тестировании (тестовый отчет).

**Тестовый отчет** (test report) - Документ, обобщающий результаты тестирования, предоставляющий информацию для сравнения текущего состояния проекта с тест-планом и принятия обоснованных управленческих решений.

**Содержание:**

- **Объект тестирования:** Подробное описание протестированных компонентов, модулей или систем.
- **Тестовые данные:** Указание источников и типов данных, использованных в процессе тестирования.
- **Инструментарий:** Перечисление всех инструментов, применяемых для проведения тестирования (системы управления тестированием, средства автоматизации и т.д.).
- **Выявленные проблемы:** Полный список обнаруженных дефектов с указанием их типа, серьезности, приоритета и статуса.
- **Команда:** Состав команды тестирования с указанием ролей и ответственностей каждого участника.

- **Версии:** Указание версий всех компонентов системы (фронтенд, бэкенд, базы данных и т.д.), на которых проводилось тестирование.
- **Статистика дефектов:**
  - Общее количество найденных дефектов.
  - Количество дефектов, исправленных в ходе тестирования.
  - Количество оставшихся дефектов с указанием их критичности.
- **Статистика тестов:**
  - Общее количество тест-кейсов.
  - Количество автоматизированных тестов.
- **Ссылки:** Ссылки на результаты прогонов тестов, отдельные тест-кейсы и автотесты (если применимо).

**Структура отчета:**

1. **Краткое резюме:** Сжатое описание проведенного тестирования, достаточное для понимания общей картины без углубления в детали.
2. **Команда тестирования:** Список участников с указанием их ролей и зон ответственности.
3. **Процесс тестирования:** Пошаговое описание всех этапов тестирования, включая подготовку, выполнение и анализ результатов.
4. **Расписание:** Подробный график проведения тестирования с указанием дат начала и окончания каждого этапа.
5. **Статистика дефектов:** Таблица с перечнем всех выявленных дефектов, их описаниями и оценкой важности.
6. **Рекомендации:** Конкретные предложения по дальнейшим действиям, основанных на результатах тестирования (корректировка тест-плана, выделение дополнительных ресурсов и т.д.).
7. **Приложения:** Дополнительные материалы, такие как скриншоты, логи, отчеты об ошибках и т.д.

**Оценка качества:**

Отчет должен содержать объективную оценку объема и качества проведенных работ, описание возникших сложностей и рекомендации по их устранению. Предоставляемая информация должна быть полной, точной и подкрепленной конкретными фактами и цифрами.

## 22) Что такое цикл разработки ПО (SDLC)?

Цикл разработки программного обеспечения (Software Development Life Cycle, SDLC) – это структурированный процесс создания программного обеспечения, включающий в себя последовательность этапов от планирования до развертывания и сопровождения.

**Основные этапы SDLC:**

1. **Планирование (Planning):** Определение целей проекта, требований к продукту, ресурсов, сроков и бюджета.
2. **Анализ требований (Requirements Analysis):** Сбор, документирование и анализ требований к функциональности, производительности, безопасности и другим аспектам ПО.
3. **Проектирование (Design):** Создание архитектуры системы, проектирование интерфейсов, баз данных и других компонентов ПО.
4. **Разработка (Development):** Написание кода, создание модулей и компонентов ПО.
5. **Тестирование (Testing):** Проверка соответствия ПО заявленным требованиям и выявление дефектов.
6. **Развертывание (Deployment):** Установка и настройка ПО на целевой среде.
7. **Сопровождение (Maintenance):** Исправление ошибок, обновление функциональности, адаптация ПО к новым условиям.

## 23) Модели SDLC:

1. **Водопадная модель или Каскадная (Waterfall Model)**: Линейная и последовательная модель, где каждый этап разработки должен быть завершен перед началом следующего. Этапы включают сбор требований,

проектирование, реализацию, тестирование, развертывание и сопровождение.

2. **Итеративная модель (Iterative Model)**: Процесс разработки делится на небольшие итерации, каждая из которых включает в себя полный цикл разработки. Это позволяет постепенно улучшать и дорабатывать продукт.
3. **Спиральная модель (Spiral Model)**: Комбинирует элементы итеративной модели и модели управления рисками. Процесс разработки проходит через несколько циклов (спиралей), каждый из которых включает планирование, анализ рисков, разработку и оценку.
4. **Модель V-образного цикла (V-Model)**: Расширение водопадной модели, где этапы верификации и валидации (тестирования) соответствуют каждому этапу разработки. Это обеспечивает более строгий контроль качества на каждом этапе.
5. **Аджайл (Agile)**: Гибкая методология, ориентированная на быструю доставку функционального ПО через короткие итерации (спринты). Включает такие фреймворки, как Scrum и Kanban.
6. **Модель прототипирования (Prototype Model)**: Создание прототипов ПО для уточнения требований и получения обратной связи от пользователей до начала основной разработки.
7. **Модель RAD (Rapid Application Development)**: Фокусируется на быстрой разработке с использованием компонентного подхода и частых итераций.

**Выбор модели SDLC зависит от:**

- Специфики проекта
- Требований заказчика
- Опыта команды
- Доступных ресурсов
- Уровня риска

## 24) V-модель разработки ПО

**V-модель разработки ПО** (V-Model) — это расширение водопадной модели, которая подчеркивает важность верификации и валидации на каждом этапе

разработки. V-модель получила свое название из-за графического представления, напоминающего букву "V", где левая сторона "V" представляет этапы разработки, а правая — соответствующие этапы тестирования.

**Основные этапы V-модели:**

1. **Сбор и анализ требований**:
    - **Этап разработки**: Сбор и документирование требований.
    - **Этап тестирования**: Приемочное тестирование (User Acceptance Testing, UAT) для проверки соответствия ПО требованиям.
2. **Системное проектирование**:
    - **Этап разработки**: Разработка общей архитектуры системы.
    - **Этап тестирования**: Системное тестирование для проверки всей системы в целом.
3. **Архитектурное проектирование**:
    - **Этап разработки**: Разработка архитектуры отдельных модулей.
    - **Этап тестирования**: Интеграционное тестирование для проверки взаимодействия модулей.
4. **Детальное проектирование**:
    - **Этап разработки**: Подробное проектирование каждого модуля.
    - **Этап тестирования**: Модульное тестирование для проверки отдельных модулей.
5. **Реализация**:
    - **Этап разработки**: Написание кода и реализация функциональности.

**Преимущества V-модели:**

- **Строгий контроль качества**: Каждому этапу разработки соответствует этап тестирования, что обеспечивает высокое качество ПО.
- **Прозрачность процесса**: Четко определенные этапы и их последовательность делают процесс разработки прозрачным и

управляемым.

- **Раннее выявление дефектов**: Тестирование начинается на ранних этапах, что позволяет выявлять и исправлять дефекты до их накопления.

**Недостатки V-модели:**

- **Малоподходящая для гибких изменений**: Как и водопадная модель, V-модель плохо подходит для проектов с часто меняющимися требованиями.
- **Высокие затраты на документацию**: Требует значительных усилий на документирование каждого этапа.

## 25) Итеративная модель разработки ПО

**Итеративная модель разработки ПО** (Iterative Model) – это подход, при котором процесс создания программного обеспечения разбивается на последовательные циклы (итерации). Каждая итерация включает в себя все этапы разработки: анализ требований, проектирование, кодирование, тестирование и внедрение.

**Принципы итеративной модели:**

- **Поэтапная разработка:** Функциональность системы реализуется постепенно, итерация за итерацией.
- **Обратная связь:** После каждой итерации заказчик получает рабочую версию продукта и может дать обратную связь, которая учитывается в следующих итерациях.
- **Адаптивность:** Модель позволяет гибко реагировать на изменения требований и вносить корректировки в процессе разработки.
- **Управление рисками:** Риски выявляются и оцениваются на каждой итерации, что позволяет своевременно принимать меры по их снижению.

**Преимущества итеративной модели:**

- **Гибкость:** Позволяет адаптироваться к изменениям требований и получать обратную связь от заказчика на ранних этапах.
- **Снижение рисков:** Риски выявляются и управляются на каждой итерации, что уменьшает вероятность неудачи проекта.

- **Раннее получение результатов:** Заказчик получает рабочую версию продукта уже после первой итерации, что позволяет оценить прогресс и внести коррективы.

**Недостатки итеративной модели:**

- **Сложность планирования:** Трудно точно оценить сроки и стоимость проекта в начале разработки.
- **Потенциальные проблемы с архитектурой:** При неправильном подходе могут возникнуть проблемы с архитектурой системы из-за частых изменений.

## 26) Agile модель. Scrum. Kanban.

**Agile модель** — это гибкий подход к разработке программного обеспечения, который фокусируется на быстрой доставке функционального ПО через короткие итерации (спринты). Agile основывается на принципах, изложенных в Манифесте Agile, таких как взаимодействие с клиентом, гибкость в изменении требований и сотрудничество внутри команды.

**Основные характеристики Agile:**

1. **Итеративность и инкрементальность**: Разработка ведется короткими циклами, каждый из которых добавляет новую функциональность.
2. **Гибкость**: Возможность быстро адаптироваться к изменениям требований.
3. **Сотрудничество**: Тесное взаимодействие между разработчиками, тестировщиками и заказчиками.
4. **Фокус на ценности**: Приоритет отдается задачам, которые приносят наибольшую ценность пользователям.

**Scrum** — это фреймворк Agile, который структурирует процесс разработки через короткие, фиксированные итерации, называемые спринтами (обычно 2-4 недели). Основные элементы Scrum включают:

1. **Роли**:
    - **Scrum-мастер**: Помогает команде следовать принципам Scrum и устраняет препятствия.

- **Владелец продукта (Product Owner)**: Определяет приоритеты и управляет бэклогом продукта.
- **Команда разработки**: Самоорганизующаяся группа, которая выполняет работу по созданию продукта.

2. **Артефакты**:
   - **Бэклог продукта**: Список всех задач и требований к продукту.
   - **Бэклог спринта**: Список задач, выбранных для выполнения в текущем спринте.
   - **Инкремент**: Готовый к выпуску результат работы команды за спринт.
3. **События**:
   - **Планирование спринта**: Определение задач для текущего спринта.
   - **Ежедневные встречи (Daily Stand-up)**: Краткие ежедневные встречи для обсуждения прогресса.
   - **Обзор спринта (Sprint Review)**: Демонстрация результатов работы за спринт.
   - **Ретроспектива спринта (Sprint Retrospective)**: Анализ прошедшего спринта и поиск способов улучшения.

**Kanban** — это метод управления проектами, который фокусируется на визуализации рабочего процесса и ограничении незавершенной работы. Основные элементы Kanban включают:

1. **Доска Kanban**: Визуальное представление рабочего процесса, разделенное на колонки (например, "To Do", "In Progress", "Done").
2. **Карточки (Kanban Cards)**: Представляют задачи или элементы работы, которые перемещаются по доске.
3. **Ограничение незавершенной работы (WIP Limits)**: Ограничение количества задач, которые могут находиться в каждой колонке одновременно, чтобы избежать перегрузки команды.
4. **Непрерывное улучшение**: Постоянный анализ и оптимизация рабочего процесса.

## 27) Состав ЕСПД. ГОСТ 19 общая структура

**ЕСПД** (Единая система программной документации) — это комплекс государственных стандартов, устанавливающих взаимосвязанные правила разработки, оформления и обращения программных документов. ЕСПД обеспечивает единообразие документации на всех этапах жизненного цикла программного обеспечения.

**Состав ЕСПД:**

ЕСПД включает в себя стандарты, регламентирующие разработку следующих видов документов:

- **Проектные документы:**
  - Техническое задание (ТЗ)
  - Эскизный проект (ЭП)
  - Технический проект (ТП)
  - Рабочий проект (РП)
- **Документы по эксплуатации:**
  - Руководство пользователя
  - Руководство системного программиста
  - Руководство оператора
  - Описание языка
- **Документы по тестированию:**
  - Программа и методика испытаний (ПМИ)
  - Технические условия (ТУ)

**ГОСТ 19 (Общая структура ЕСПД):**

ГОСТ 19 определяет общую структуру ЕСПД и устанавливает требования к содержанию, оформлению и обращению программных документов. Он включает следующие основные разделы:

- **Общие положения:** Определяет область применения стандарта, термины и определения.

- **Виды программных документов:** Перечисляет виды документов, входящих в ЕСПД, и устанавливает требования к их разработке.
- **Правила оформления программных документов:** Устанавливает требования к оформлению документов, включая структуру, нумерацию разделов, шрифты и т.д.
- **Правила обращения программных документов:** Определяет порядок внесения изменений в документы, их утверждение и хранение.