**Билет №1**

**1. Тестирование как способ обеспечения качества программного обеспечения.**

**2. Разработать модульные тесты для программного модуля AsyncAdd, содержащий метод асинхронного чтения всех строк из файла. Код для программного модуля находится Resorce/ModuleC#/». Оформить тест-кейсы для модульного тестирования.**

**Тестирование** — это процесс проверки программного обеспечения, системы или приложения на соответствие определённым требованиям и оценки их качества.

Качество программного продукта характеризуется набором свойств, определяющих, насколько продукт "хорош" с точки зрения заинтересованных сторон, таких как заказчик продукта, спонсор, конечный пользователь, разработчики и тестировщики продукта, инженеры поддержки, сотрудники отделов маркетинга, обучения и продаж.

**Цель тестирования** — выявить ошибки, неполадки и нежелательное поведение программного продукта. Чем раньше обнаруживается дефект, тем дешевле его исправление и тем меньше последствий он несёт как для компании-разработчика, так и для непосредственного пользователя продукта.

**Существует несколько видов тестирования программного обеспечения:**

Модульное тестирование: Проверка отдельных модулей или компонентов программы для подтверждения их корректной работы.

Интеграционное тестирование: Тестирование взаимодействия между несколькими модулями или компонентами для выявления дефектов, возникающих при их интеграции.

Системное тестирование: Проверка всей системы в целом для обеспечения ее соответствия требованиям и спецификациям.

Приемочное тестирование: Оценка системы с точки зрения пользователя для подтверждения ее готовности к эксплуатации.

Регрессионное тестирование: Проверка того, что изменения в программном обеспечении не привели к появлению новых дефектов в уже работающих функциях.

Нагрузочное тестирование: Оценка производительности системы при высокой нагрузке для выявления ее пределов.

**Виды тестирования:**

**Статическое тестирование**. Проверка кода и документации без запуска приложения или программы.

**Динамическое тестирование**. Проводится с запуском ПО. Тестировщики оценивают, долго ли грузятся страницы, сколько оперативной памяти нужно для нормальной работы приложения.

**Функциональное тестирование**. Проверяет, справляется ли приложение с возложенными на него функциями и задачами.

**Нефункциональное тестирование**. Исследует совместимость и производительность компонентов приложения.

**Тестирование «чёрного ящика»** Тестировщик работает только с интерфейсной частью продукта и не видит его код.

**Тестирование «белого ящика»** Инженер видит код проекта.

**Тестирование «серого ящика»** Тестирование, при котором тестировщик имеет частичный доступ к исходному коду или документации.

Тестирование играет ключевую роль в обеспечении качества программного обеспечения. Это процесс оценки программного обеспечения с целью обнаружения дефектов и несоответствий требованиям. По сути, это контролируемый способ поиска проблем до того, как они попадут к конечным пользователям.

**Почему тестирование важно для обеспечения качества:**

Обнаружение дефектов: Главная задача тестирования - найти ошибки, баги, уязвимости и другие дефекты в коде. Чем раньше обнаружены эти проблемы, тем дешевле и проще их исправить.

Проверка соответствия требованиям: Тестирование подтверждает, что программное обеспечение работает так, как задумано и соответствует определенным требованиям и спецификациям.

Повышение надежности и стабильности: Обнаружив и исправив дефекты, тестирование делает программное обеспечение более надежным и стабильным, что снижает вероятность сбоев и ошибок в процессе эксплуатации.

Улучшение пользовательского опыта: Тестирование помогает убедиться, что программное обеспечение удобно в использовании, интуитивно понятно и соответствует ожиданиям пользователей.

Снижение рисков: Обнаружение и устранение дефектов до релиза снижает риски, связанные с негативными отзывами пользователей, финансовыми потерями, репутационными убытками и даже юридическими последствиями.

Обеспечение безопасности: Тестирование помогает выявить уязвимости в системе безопасности, которые могут быть использованы злоумышленниками для получения несанкционированного доступа или нанесения вреда.

Увеличение доверия: Успешно протестированное и надежное программное обеспечение вызывает больше доверия у пользователей и заинтересованных сторон.

**Различные виды тестирования:**

**Существует множество видов тестирования, каждый из которых направлен на проверку различных аспектов программного обеспечения. Некоторые из наиболее распространенных:**

Модульное тестирование (Unit Testing): Тестирование отдельных компонентов (модулей) программного обеспечения.

Интеграционное тестирование (Integration Testing): Тестирование взаимодействия между различными модулями или компонентами.

Системное тестирование (System Testing): Тестирование всего программного обеспечения как единого целого.

Приемочное тестирование (Acceptance Testing): Тестирование программного обеспечения конечными пользователями для проверки соответствия их требованиям.

Регрессионное тестирование (Regression Testing): Повторное тестирование после внесения изменений в код для убеждения, что новые изменения не привели к появлению новых дефектов.

Производительное тестирование (Performance Testing): Оценка производительности программного обеспечения под различными нагрузками.

Тестирование безопасности (Security Testing): Проверка программного обеспечения на наличие уязвимостей и обеспечение его безопасности.

Функциональное тестирование (Functional Testing): Проверка того, что программное обеспечение выполняет свои функции в соответствии со спецификациями.

Нефункциональное тестирование (Non-Functional Testing): Оценка таких характеристик программного обеспечения, как производительность, надежность, удобство использования и безопасность.

UI/UX тестирование (UI/UX Testing): Проверка удобства и визуальной привлекательности пользовательского интерфейса.

Альфа-тестирование (Alpha Testing): Внутреннее тестирование продукта перед выпуском.

Бета-тестирование (Beta Testing): Тестирование продукта реальными пользователями вне команды разработчиков.

**Стратегии тестирования:**

**Выбор подходящей стратегии тестирования зависит от конкретных потребностей и целей проекта. Вот некоторые общие стратегии:**

Водопадная модель (Waterfall Model): Тестирование выполняется после завершения каждого этапа разработки.

Agile-разработка: Тестирование выполняется итеративно и параллельно с разработкой.

TDD (Test-Driven Development): Разработка начинается с написания тестов, а затем пишется код, который проходит эти тесты.

BDD (Behavior-Driven Development): Тесты пишутся на основе поведения системы с точки зрения пользователя.

**Инструменты тестирования:**

Существует множество инструментов тестирования, которые помогают автоматизировать и упростить процесс тестирования. Они включают в себя:

Инструменты для модульного тестирования: JUnit, NUnit, pytest.

Инструменты для автоматизированного тестирования UI: Selenium, Cypress, Playwright.

Инструменты для тестирования производительности: JMeter, LoadRunner.

Инструменты для управления тестированием: TestRail, Zephyr, qTest.

**Билет №2**

**1. Тестирование, верификация и валидация: понятия, сравнение, примеры.**

**2. Разработать модульные тесты для программного модуля Collection, содержащий метод добавления данных в список. Код для программного модуля находится «Resorce/ModuleC#/». Оформить тест-кейсы для модульного тестирования.**

**Тестирование (Testing)**

Определение: Процесс выполнения программы или системы с целью обнаружения дефектов (ошибок).

Цель: Обнаружить ошибки в фактической реализации ПО.

Что делается: Запускаются тесты (наборы входных данных и ожидаемых результатов), и сравниваются фактические результаты с ожидаемыми.

Пример:

Ввести “10” в поле ввода, нажать “рассчитать”, убедиться, что результат “20”.

**Верификация (Verification)**

Определение: Процесс оценки, соответствует ли разработанное ПО спецификациям и требованиям.

Цель: Убедиться, что ПО реализовано правильно в соответствии с проектом и спецификациями.

Что делается: Анализируется проектная документация, код, тесты, проводится статическое тестирование (ревью кода, инспекции, статический анализ).

Пример:

Проверка, что код соответствует требованиям к синтаксису и кодированию

Проверка соответствия реализации алгоритма спецификации.

Анализ тестовых сценариев, чтобы убедиться, что они покрывают все требования.

**Валидация (Validation)**

Определение: Процесс оценки соответствия разработанного ПО потребностям пользователя и его ожиданиям.

Цель: Убедиться, что ПО соответствует потребностям бизнеса и пользователей.

Что делается: Оценка ПО пользователями, проведение приемочного тестирования (UAT), анализ отзывов пользователей, проверка соответствия ПО задачам, для которых оно предназначено.

Пример:

Приемочное тестирование, в ходе которого заказчик подтверждает, что ПО соответствует его требованиям.

Анализ отзывов пользователей, чтобы понять, насколько ПО соответствует их ожиданиям и потребностям.

Тестирование юзабилити, чтобы убедиться, что ПО удобно в использовании.



**Билет №3**

**1. Цели и задачи процесса тестирования.**

**2. Разработать модульные тесты для программного модуля AsyncAdd, содержащий метод асинхронной записи всех строк в файла. Код для программного модуля находится Resorce/ModuleC#/». Оформить тест-кейсы для модульного тестирования.**

Процесс тестирования программного обеспечения имеет несколько ключевых целей и задач, которые направлены на обеспечение его качества и надежности. Вот основные из них:

**Цели:**

Выявление дефектов: Основная цель тестирования — выявление дефектов и ошибок в программном обеспечении до его выпуска в эксплуатацию. Это помогает предотвратить сбои и недоработки, которые могут негативно сказаться на пользователях.

Улучшение качества: Тестирование способствует улучшению качества программного продукта, обеспечивая его соответствие требованиям и ожиданиям пользователей. Это включает проверку функциональности, производительности, безопасности и других аспектов.

Обеспечение соответствия требованиям: Тестирование помогает удостовериться, что программное обеспечение соответствует заданным требованиям и спецификациям. Это включает проверку, что все заявленные функции работают правильно и полностью реализованы.

Снижение рисков: Тестирование помогает выявить и устранить потенциальные риски, связанные с эксплуатацией программного продукта. Это может включать анализ уязвимостей, оценку производительности под нагрузкой и другие виды тестирования.

Обеспечение удовлетворенности пользователей: Качественное программное обеспечение удовлетворяет потребности и ожидания пользователей. Тестирование помогает убедиться в этом, что, в свою очередь, способствует увеличению доверия и лояльности пользователей.

**Задачи:**

Планирование тестирования: Разработка стратегии и плана тестирования, включающего определение объема работ, требуемых ресурсов, сроков и оценку рисков.

Разработка тестов: Создание тестовых сценариев, тест-кейсов и тестовых данных, которые помогут систематизировать процесс тестирования и обеспечить его полноту.

Настройка тестовой среды: Подготовка и настройка тестовой среды, в которой будут проводиться тесты. Это важно для обеспечения точности и достоверности результатов тестирования.

Выполнение тестов: Проведение тестирования согласно разработанным тест-кейсам и фиксирование результатов. На этом этапе выявляются дефекты и создаются отчеты о найденных ошибках.

Анализ результатов: Анализ результатов тестирования для оценки качества программного обеспечения и определения необходимости внесения изменений или исправлений.

Отчетность: Создание отчетов о результатах тестирования, включающих информацию о найденных дефектах, их критичности и предложениях по их исправлению.

Регрессионное тестирование: Повторное тестирование программного обеспечения после внесения изменений для проверки, что исправленные дефекты не повлияли на работу других функций.

**Билет №4**

**1. Жизненный цикл тестирования.**

**2. Разработать модульные тесты для программного модуля Equation, содержащий метод решения линейного уравнения. Код для программного модуля находится «Resorce/ModuleC#/». Оформить тест-кейсы для модульного тестирования.**

Жизненный цикл тестирования (STLC) — это последовательность действий, проводимых в процессе тестирования, с помощью которых гарантируется качество программного обеспечения и его соответствие требованиям.

Цикл состоит из шести основных этапов:

* **Анализ требований**. QA-команда оценивает требования с точки зрения тестирования.
* **Планирование тестирования**. Определяются чёткие цели и область применения перед началом любой деятельности по тестированию.
* **Создание тест-кейсов**. Разрабатываются, пересматриваются, уточняются, дорабатываются и перерабатываются тест-кейсы, наборы тест-кейсов, тестовые сценарии и иные артефакты, которые будут использоваться при непосредственном выполнении тестирования.
* **Настройка тестового окружения**. Готовится тестовая среда, настраивается необходимое оборудование, программное обеспечение и сетевые настройки для репликации производственной среды.
* **Выполнение тестирования**. Выполняются тестовые примеры и сценарии на основе подготовленных тестовых материалов. Любые дефекты или сбои, обнаруженные во время тестирования, незамедлительно регистрируются в системе отслеживания дефектов.
* **Завершение тестирования**. Создаётся отчёт о результатах тестирования. QA-команда обсуждает и анализирует баги, делает выводы из возникших проблем, чтобы избежать подобных проблем в будущем.
* **Общее планирование и анализ требований**
* **Уточнение критериев приемки**
* **Уточнение стратегии тестирования**
* **Разработка тест кейсов**
* **Выполнение тест кейсов**
* **Фиксация найденных дефектов**
* **Анализ результатов тестирования**
* **Отчетность**

**Билет №5**

**1. Процессы тестирования при разработке программного обеспечения.**

**2. Разработать модульные тесты для программного модуля Equation, содержащий метод решения квадратного уравнения. Код для программного модуля находится Resorce/ModuleC#/». Оформить тест-кейсы для модульного тестирования.**

**Процесс тестирования при разработке программного обеспечения (ПО) состоит из нескольких этапов:**

* Планирование тестирования. На этом этапе определяются цели и задачи тестирования, выбираются методы и инструменты, а также составляется план тестирования.
* Разработка тестовых сценариев и тестовых случаев. Тестовые сценарии описывают последовательность действий, необходимых для проверки определённой функциональности ПО. Тестовые случаи представляют собой более детализированные инструкции для выполнения тестовых сценариев.
* Подготовка тестовых данных. Тестовые данные — это наборы входных значений, которые будут использоваться для проверки корректности работы ПО.
* Выполнение тестов. На этом этапе тестировщик применяет тестовые сценарии, тестовые случаи и тестовые данные для проверки работоспособности ПО.
* Анализ результатов. После выполнения тестов анализируются результаты, и обнаруженные ошибки и недоработки документируются.
* Исправление ошибок и повторное тестирование. Разработчики исправляют обнаруженные ошибки, после чего тестировщики проводят повторное тестирование, чтобы убедиться в их исправлении.

**Процессы тестирования при разработке программного обеспечения играют ключевую роль в обеспечении качества и надежности продуктов. Вот основные этапы и типы тестирования:**

* **Функциональное тестирование (Юнит-тестирование)**: Проверка отдельных компонентов (функций или методов) программы. Это помогает выявлять ошибки на ранних этапах разработки.
* **Интеграционное тестирование**: Проверка взаимодействия между разными модулями или компонентами. Цель — убедиться, что они правильно работают вместе.
* **Системное тестирование:** Тестирование всей системы в целом для проверки её соответствия требованиям. Включает функциональное и нефункциональное тестирование.
* **Приемочное тестирование:** Завершающий этап тестирования, проводимый пользователями или заказчиками для оценки, соответствует ли система их ожиданиям и требованиям.
* **Регрессионное тестирование:** Повторное тестирование системы после внесения изменений или исправлений, чтобы убедиться, что эти изменения не вызвали новых ошибок.
* **Тестирование производительности:** Проверка, как система работает под различными нагрузками. Включает нагрузочное, стрессовое и тестирование стабильности.
* **Тестирование безопасности:** Обеспечение защиты системы от внешних угроз и уязвимостей.

Тестирование может быть ручным или автоматизированным.

**Процессы тестирования в разработке ПО (ПО) зависят от выбранной методологии разработки. Рассмотрим несколько распространенных моделей:**

**Waterfall (каскадная модель):**

Требования: Тестирование начинается после завершения этапа кодирования.

Тестирование системы: Проводится после интеграции всех модулей.

Приемочное тестирование: Заказчик проверяет, соответствует ли ПО его требованиям.

**V-Model (V-образная модель):**

Требования: Каждый этап разработки имеет соответствующий этап тестирования.

Модульное тестирование: Проверка отдельных модулей.

Интеграционное тестирование: Проверка взаимодействия модулей.

Системное тестирование: Проверка всей системы на соответствие требованиям.

Приемочное тестирование: Заказчик проверяет систему.

**Agile (Scrum, Kanban):**

Итеративная разработка: Тестирование проводится в каждой итерации (спринте).

Автоматизированное тестирование: Широко используется для автоматизации регрессионных тестов и быстрого получения обратной связи.

TDD (Test-Driven Development): Тесты пишутся до написания кода.

BDD (Behavior-Driven Development): Тесты описывают поведение системы с точки зрения пользователя.

Непрерывная интеграция (CI) и непрерывная поставка (CD): Автоматизация сборки, тестирования и развертывания ПО.

\_\_\_\_\_

Процессы тестирования при разработке программного обеспечения (ПО) - это систематические и организованные мероприятия, направленные на обнаружение дефектов (багов) в ПО и проверку соответствия ПО требованиям. Целью тестирования является повышение качества, надежности и безопасности ПО.

**Вот обзор основных аспектов процессов тестирования:**

**1. Жизненный цикл тестирования (STLC - Software Testing Life Cycle):**

STLC определяет последовательность этапов, которые проходят тестирование в процессе разработки ПО. Хотя конкретные этапы могут варьироваться в зависимости от методологии разработки (например, Agile, Waterfall), обычно выделяют следующие:

Анализ требований: На этом этапе команда тестирования анализирует требования к ПО, чтобы понять, что необходимо протестировать. Выявляются неясности и противоречия в требованиях. Результатом является понимание целей тестирования и критериев приемки.

Планирование тестирования: Определение стратегии тестирования, целей, области охвата, ресурсов, графика и используемых инструментов. Создается план тестирования (Test Plan), который описывает подход к тестированию, роли и обязанности участников, риски и стратегии их смягчения.

Разработка тестовой документации: Создание тестовых сценариев (Test Cases), тестовых наборов (Test Suites) и тестовых данных (Test Data). Тестовые сценарии описывают конкретные шаги, необходимые для проверки определенной функциональности или аспекта ПО.

Настройка тестового окружения: Создание необходимой инфраструктуры для проведения тестирования, включая аппаратное обеспечение, программное обеспечение, сети и данные. Это может включать установку операционных систем, баз данных, серверов приложений и другого необходимого ПО.

Выполнение тестирования: Запуск тестовых сценариев и фиксация результатов. Тестировщики выполняют разработанные тестовые сценарии и сравнивают фактический результат с ожидаемым. Обнаруженные дефекты регистрируются в системе отслеживания ошибок (Bug Tracking System).

Оценка критериев выхода (Exit Criteria Evaluation): Определение критериев завершения тестирования. Анализ покрытия тестами, количества обнаруженных дефектов, серьезности дефектов и стабильности ПО. При достижении критериев выхода, тестирование этапа считается завершенным.

Закрытие цикла тестирования: Подготовка итогового отчета о тестировании (Test Summary Report), содержащего информацию о ходе тестирования, обнаруженных дефектах, качестве ПО и рекомендациях. Анализ уроков, извлеченных из процесса тестирования, для улучшения будущих проектов.

**2. Уровни тестирования:**

Различные уровни тестирования направлены на проверку разных аспектов ПО. Основные уровни:

Модульное тестирование (Unit Testing): Тестирование отдельных модулей или компонентов кода. Выполняется разработчиками для проверки правильности реализации отдельных функций, методов или классов.

Интеграционное тестирование (Integration Testing): Тестирование взаимодействия между различными модулями или компонентами. Проверяется, как отдельные части системы работают вместе. Существуют различные подходы к интеграционному тестированию, такие как “сверху вниз” и “снизу вверх”.

Системное тестирование (System Testing): Тестирование всей системы в целом. Проверяется соответствие системы требованиям и спецификациям. Выполняется командой тестирования после интеграции всех компонентов.

Приемочное тестирование (Acceptance Testing): Тестирование системы конечными пользователями или заказчиками. Проверяется, соответствует ли система их потребностям и ожиданиям. Успешное прохождение приемочного тестирования является условием для приемки системы в эксплуатацию.

**3. Типы тестирования:**

Разнообразие типов тестирования позволяет проверить ПО с разных точек зрения. Некоторые распространенные типы:

Функциональное тестирование: Проверка соответствия функциональности ПО требованиям.

Дымное тестирование (Smoke Testing): Быстрое поверхностное тестирование для проверки работоспособности основных функций.

Регрессионное тестирование (Regression Testing): Повторное тестирование после внесения изменений в код для проверки, что новые изменения не привели к появлению новых дефектов или ухудшению существующей функциональности.

Позитивное тестирование (Positive Testing): Тестирование с использованием корректных данных и сценариев.

Негативное тестирование (Negative Testing): Тестирование с использованием некорректных данных и сценариев для проверки обработки ошибок и исключений.

Нефункциональное тестирование: Проверка нефункциональных характеристик ПО, таких как производительность, безопасность, удобство использования и надежность.

Тестирование производительности (Performance Testing): Оценка скорости, масштабируемости и стабильности системы при различных нагрузках. Включает нагрузочное тестирование, стресс-тестирование и тестирование на выносливость.

Тестирование безопасности (Security Testing): Проверка системы на уязвимости и защита от несанкционированного доступа. Включает тестирование на проникновение, сканирование уязвимостей и аудит кода.

Тестирование юзабилити (Usability Testing): Оценка удобства использования системы для конечных пользователей. Проводится с привлечением реальных пользователей.

Тестирование локализации (Localization Testing): Проверка адаптации ПО для различных языков и регионов.

Тестирование доступности (Accessibility Testing): Проверка доступности ПО для людей с ограниченными возможностями.

Тестирование пользовательского интерфейса (UI Testing): Проверка визуальных элементов и интерактивности интерфейса.

Альфа-тестирование (Alpha Testing): Внутреннее тестирование продукта перед выпуском для ограниченного круга пользователей или тестировщиков.

Бета-тестирование (Beta Testing): Тестирование продукта реальными пользователями в реальной среде перед официальным выпуском.

**4. Подходы к тестированию:**

Ручное тестирование (Manual Testing): Тестирование, выполняемое тестировщиками вручную, без использования автоматизированных инструментов. Используется для исследования нового функционала, тестирования юзабилити и в случаях, когда автоматизация затруднена.

Автоматизированное тестирование (Automated Testing): Тестирование, выполняемое с использованием автоматизированных инструментов и скриптов. Используется для регрессионного тестирования, нагрузочного тестирования и других повторяющихся задач.

**5. Методологии тестирования:**

Waterfall: Последовательный подход, при котором тестирование начинается после завершения разработки.

Agile: Итеративный подход, при котором тестирование выполняется на каждой итерации разработки. Тестировщики работают в тесном сотрудничестве с разработчиками и другими членами команды.

V-Model: Расширение Waterfall, которое связывает каждый этап разработки с соответствующим этапом тестирования.

**6. Инструменты тестирования:**

Существует множество инструментов, облегчающих процесс тестирования:

Системы управления тестами (Test Management Tools): Jira, TestRail, Zephyr.

Инструменты автоматизации тестирования: Selenium, JUnit, TestNG, Cypress, Playwright.

Инструменты управления дефектами (Bug Tracking Tools): Jira, Bugzilla, Mantis.

Инструменты тестирования производительности: JMeter, LoadRunner.

Инструменты тестирования безопасности: OWASP ZAP, Burp Suite.

**Билет №6**

**1. Классификация тестирования по уровням.**

**2. Разработать модульные тесты для программного модуля MathFunctions, содержащий методы вычисления косинуса и синуса. Код для программного модуля находится «Resorce/ModuleC#/». Оформить тест-кейсы для модульного тестирования.**

Классификация тестирования программного обеспечения по уровням делится на следующие категории:

**Модульное тестирование (Unit Testing):**

Проверка отдельных модулей или компонентов системы.

Проводится разработчиками.

**Интеграционное тестирование (Integration Testing):**

Проверка взаимодействия между модулями.

Выявление проблем взаимодействия.

**Системное тестирование (System Testing):**

Полная проверка всей системы в целом.

Проводится командами качества.

**Приемочное тестирование (Acceptance Testing):**

Проверка системы на соответствие требованиям клиента.

Может проводиться непосредственно пользователями.

\_\_\_\_\_

**1. Модульное тестирование (Unit Testing):**

Цель: Проверка корректности работы отдельных модулей, компонентов или функций кода.

Кто выполняет: Обычно разработчики.

Объект тестирования: Отдельные функции, методы, классы или небольшие модули.

Методы: Как правило, автоматизированное тестирование с использованием фреймворков для модульного тестирования (например, JUnit для Java, pytest для Python, NUnit для .NET).

Фокус: Изолированная проверка логики, входных и выходных данных, обработка исключений.

Преимущества:

Раннее обнаружение дефектов.

Упрощение отладки (баги локализованы в конкретных модулях).

Улучшение дизайна кода (требует модульного дизайна).

Облегчение рефакторинга.

Недостатки:

Не проверяет взаимодействие между модулями.

Требует хорошего понимания кода.

Может быть сложно писать тесты для некоторых модулей.

**2. Интеграционное тестирование (Integration Testing):**

Цель: Проверка корректности взаимодействия между интегрированными модулями, компонентами или подсистемами.

Кто выполняет: Обычно команда тестирования или разработчики (в зависимости от организации).

Объект тестирования: Комбинации интегрированных модулей, API, сервисы.

Методы:

Сверху вниз (Top-Down): Начинается с тестирования верхнего уровня модулей и постепенно переходит к нижним. Требуются “заглушки” (stubs) для имитации поведения нижних уровней.

Снизу вверх (Bottom-Up): Начинается с тестирования нижнего уровня модулей и постепенно переходит к верхним. Требуются “драйверы” (drivers) для имитации поведения верхних уровней.

Большой взрыв (Big Bang): Все модули интегрируются одновременно и тестируются как единое целое. Сложно отлаживать.

Инкрементное тестирование: Модули интегрируются и тестируются постепенно, по одному или небольшими группами. Самый распространенный и рекомендуемый подход.

Фокус: Проверка передачи данных между модулями, обработка ошибок при взаимодействии, соответствие интерфейсов.

Преимущества:

Обнаружение дефектов, связанных с интеграцией.

Проверка соответствия интерфейсов.

Уменьшение риска серьезных ошибок на более поздних этапах.

Недостатки:

Более сложно отлаживать, чем модульное тестирование.

Требует более сложной подготовки тестового окружения.

**3. Системное тестирование (System Testing):**

Цель: Проверка соответствия всей системы требованиям и спецификациям.

Кто выполняет: Независимая команда тестирования (тестировщики).

Объект тестирования: Полностью интегрированная система.

Методы: Черный ящик (Black-Box testing) – тестирование без знания внутренней структуры системы. Тестирование проводится на основе требований и спецификаций.

Типы тестирования, часто выполняемые на этом уровне:

Функциональное тестирование (например, тестирование сценариев использования, тестирование граничных значений).

Нефункциональное тестирование (например, тестирование производительности, тестирование безопасности, тестирование юзабилити).

Фокус: Проверка сквозной функциональности, производительности, безопасности и других системных характеристик.

Преимущества:

Проверка соответствия системы требованиям заказчика.

Выявление серьезных дефектов, которые могут повлиять на работу системы.

Недостатки:

Может быть трудоемким и дорогостоящим.

Сложно выявить причину обнаруженных дефектов.

Требует полностью интегрированной системы.

**4. Приемочное тестирование (Acceptance Testing):**

Цель: Проверка готовности системы к эксплуатации и соответствия ожиданиям конечных пользователей или заказчиков.

Кто выполняет: Конечные пользователи, заказчики или представители заказчика.

Объект тестирования: Полностью интегрированная система, готовая к развертыванию.

Методы:

Тестирование, определяемое пользователем (User Acceptance Testing - UAT): Реальные пользователи тестируют систему в реальных условиях.

Тестирование по контракту (Contract Acceptance Testing): Проверка соответствия системы условиям контракта.

Альфа-тестирование: Тестирование пользователями или тестировщиками, но в контролируемой среде разработчика.

Бета-тестирование: Тестирование реальными пользователями в реальной среде.

Фокус: Проверка удобства использования, соответствия бизнес-процессам, соответствия требованиям заказчика.

Преимущества:

Гарантия соответствия системы ожиданиям заказчика.

Улучшение удовлетворенности пользователей.

Снижение риска проблем после развертывания.

Недостатки:

Зависит от доступности конечных пользователей.

Может быть сложно организовать и контролировать.

**4. Приемочное тестирование (Acceptance Testing):**

**4.1 Альфа-тестирование (Alpha Testing):**

Цель: Оценка продукта в контролируемой среде разработчика, часто внутренними пользователями или специально нанятыми тестировщиками. Выявление багов, проблем юзабилити и общей функциональности перед выпуском для более широкой аудитории.

Кто выполняет: Внутренние тестировщики, разработчики, сотрудники компании, либо специально нанятые тестировщики, работающие в офисе разработчика.

Среда: Контролируемая среда, часто в лаборатории или офисе разработчика.

Фокус: Поиск критических багов, проверка общей функциональности, юзабилити и соответствия требованиям.

Преимущества:

Раннее выявление проблем, которые могут повлиять на пользовательский опыт.

Возможность получить обратную связь от пользователей до публичного релиза.

Недостатки:

Может не отражать реальное использование продукта.

Требует значительных ресурсов на организацию и проведение.

**4.2 Бета-тестирование (Beta Testing):**

Цель: Оценка продукта реальными пользователями в реальной среде. Получение обратной связи о функциональности, юзабилити, производительности и надежности в условиях, максимально приближенных к реальным.

Кто выполняет: Конечные пользователи (бета-тестеры), которые используют продукт в своей обычной среде.

Среда: Реальная среда, где пользователи будут использовать продукт в повседневной жизни.

Фокус: Получение обратной связи о пользовательском опыте, выявление багов, которые не были обнаружены на предыдущих этапах, оценка производительности и надежности в реальных условиях.

Преимущества:

Получение ценной обратной связи от реальных пользователей.

Выявление проблем, которые трудно обнаружить в контролируемой среде.

Повышение удовлетворенности пользователей и снижение риска негативных отзывов после релиза.

Недостатки:

Сложно контролировать процесс бета-тестирования.

Может быть сложно обработать всю полученную обратную связь.

**4.3 Пользовательское приемочное тестирование** (UAT - User Acceptance Testing): (Описание как в предыдущем ответе, но теперь более конкретно как подвид приемочного тестирования)

**4.4 Операционное приемочное тестирование** (OAT - Operational Acceptance Testing): (Проверка готовности системы к операционной поддержке, включая резервное копирование, восстановление, мониторинг и т.д.)

**5 Выходное тестирование (Exit Testing/Release Testing/Production Readiness Testing):**

Цель: Подтверждение готовности программного обеспечения к выпуску в производственную среду. Это последний этап тестирования перед развертыванием, направленный на проверку, что все критические функции работают корректно, а риски развертывания минимальны. Некоторые считают это подмножеством системного или приемочного тестирования.

Кто выполняет: Команда тестирования, DevOps-инженеры, или команда Release Management.

Объект тестирования: Полностью развернутая система в тестовой среде, максимально приближенной к производственной. Проверяются процессы развертывания, конфигурации и интеграции.

Методы:

Повторное выполнение ключевых тестовых сценариев, которые были пройдены на предыдущих этапах.

Проверка процессов развертывания и отката изменений.

Тестирование интеграции с другими системами.

Оценка производительности и масштабируемости в условиях, приближенных к производственным.

Проверка безопасности и соответствия нормативным требованиям.

Фокус: Минимизация рисков при развертывании, проверка готовности системы к реальной нагрузке, подтверждение корректной работы всех критических функций. Убедиться, что система развертывается правильно и работает стабильно.

Преимущества:

Снижение риска проблем в производственной среде.

Повышение уверенности в качестве программного обеспечения.

Улучшение удовлетворенности пользователей.

Недостатки:

Требует хорошо настроенной тестовой среды, максимально приближенной к производственной.

Может быть сложно организовать и провести, особенно для сложных систем.

**Билет №7**

**1. Чек-листы, тест-кейсы: определение, основные элементы.**

**2. Разработать модульные тесты для программного модуля MathFunctions, содержащий методы вычисления квадратного корня и степени числа. Код для программного модуля находится «Resorce/ModuleC#/». Оформить тест-кейсы для модульного тестирования.**

**Чек-лист** — это список задач или пунктов, которые необходимо выполнить для достижения определенной цели. В тестировании чек-листы используются для проверки функциональности продукта.

Основные элементы чек-листа:

* Название чек-листа: Четкое и понятное наименование.
* Описание: Краткое объяснение цели и содержания.
* Пункты для проверки: Подробный перечень задач или шагов.
* Ожидаемый результат: Что должно произойти после выполнения каждого шага.
* Фактический результат: То, что произошло на самом деле.
* Статус: Отметка об успешности выполнения (успех, неудача, на рассмотрении).

**Тест-кейс** — это документ, который описывает набор действий, условий и данных для проверки определенной функции или аспекта приложения.

Основные элементы тест-кейса:

* Идентификатор тест-кейса: Уникальный номер или код.
* Название тест-кейса: Понятное и информативное наименование.
* Описание: Краткое описание цели теста и что он проверяет.
* Предварительные условия: Любые условия или состояния, которые должны быть выполнены перед началом теста.
* Шаги: Подробное описание каждого действия, которое необходимо выполнить.
* Ожидаемый результат: Что должно произойти после выполнения каждого шага.
* Фактический результат: То, что произошло на самом деле после выполнения шагов.
* Статус: Успешность или неудача выполнения тест-кейса.
* Комментарии: Дополнительные замечания или наблюдения.

\_\_\_\_\_

**Чек-лист (Checklist):**

Определение: Перечень пунктов (проверок, действий), которые необходимо выполнить при тестировании.

Основные элементы:

* ID (идентификатор) пункта.
* Описание проверки (что нужно проверить).
* Результат (пройдено/не пройдено/блокировано).
* Комментарии (дополнительная информация).
* Ответственный.

**Тест-кейс (Test Case):**

Определение: Подробное описание шагов, входных данных, ожидаемого результата и предварительных условий для проверки конкретной функциональности или компонента системы.

Основные элементы:

* ID (идентификатор) тест-кейса.
* Название тест-кейса (описывает цель проверки).
* Предварительные условия (условия, которые должны быть выполнены перед выполнением тест-кейса).
* Входные данные (данные, которые используются для выполнения шагов тест-кейса).
* Шаги тест-кейса (пошаговое описание действий).
* Ожидаемый результат (описание того, что должно произойти после выполнения каждого шага).
* Фактический результат (описание того, что фактически произошло).
* Статус (пройден, не пройден, заблокирован и т.д.).
* Комментарии (дополнительная информация).
* Ответственный.

**Билет №8**

**1. Анализ и тестирование требований.**

**2. Разработать модульные тесты для программного модуля ReadAndWrite, содержащий метод чтения всех строк из файла. Код для программного модуля находится «Resorce/ModuleC#/». Оформить тест-кейсы для модульного тестирования.**

**Анализ требований:**

Анализ требований — это процесс детального изучения и понимания требований к программному обеспечению (ПО), предоставленных заказчиком или пользователем. Цель анализа — выявить неясности, противоречия, неполноту или нереалистичность требований.

Задачи анализа требований:

* Сбор требований: Получение информации о потребностях заказчика и пользователей.
* Документирование требований: Формализация требований в виде спецификаций, пользовательских историй, диаграмм и т.д.
* Проверка требований: Убеждение в том, что требования являются полными, непротиворечивыми, тестируемыми и реализуемыми.
* Уточнение требований: Внесение изменений в требования на основе анализа и обратной связи от заинтересованных сторон.
* Приоритизация требований: Определение приоритетности требований для планирования разработки и тестирования.

**Тестирование требований:**

Тестирование требований — это процесс проверки того, что требования к ПО являются тестируемыми и могут быть реализованы. Тестирование требований начинается на ранних этапах разработки, еще до написания кода.

Задачи тестирования требований:

* Определение тестовых сценариев: Разработка тестовых сценариев на основе требований.
* Проверка реализуемости требований: Убеждение в том, что требования могут быть реализованы с использованием доступных технологий и ресурсов.
* Выявление дефектов в требованиях: Обнаружение неясностей, противоречий, неполноты или нереалистичности требований.
* Оценка покрытия требований тестами: Определение, насколько полно требования покрыты тестами.
* Сообщение об ошибках в требованиях: Передача информации об ошибках в требованиях разработчикам для исправления.

**Анализ требований включает:**

* Сбор требований: обсуждения с клиентами и пользователями для выявления их потребностей и ожиданий.
* Документирование требований: создание спецификаций, диаграмм и других документов, которые подробно описывают требования.
* Анализ на предмет выполнимости: оценка того, насколько требования реалистичны и осуществимы.
* Приоритизация требований: определение наиболее важных и критичных требований.

**Тестирование требований предполагает:**

* Верификацию требований: проверка на соответствие требованиям стандартам и спецификациям.
* Валидацию требований: обеспечение того, что требования правильно отражают потребности пользователей.
* Создание тест-кейсов: разработка сценариев тестирования, основанных на требованиях.
* Исполнение тест-кейсов: выполнение тестов и документирование результатов.

\_\_\_\_\_

Анализ требований - это процесс изучения и понимания требований, предъявляемых к программному обеспечению.

Тестирование требований - это процесс проверки требований на соответствие их описанию и правильность их реализации.

\_\_\_\_\_

**1. Анализ Требований (Requirements Analysis)**

**Цель:**

Понять потребности заинтересованных сторон (заказчиков, пользователей, бизнеса).

Собрать, задокументировать и детализировать требования к системе.

Убедиться, что требования полные, непротиворечивые, однозначные, реализуемые и тестируемые.

**Процесс:**

Сбор требований (Requirements Elicitation):

Идентификация заинтересованных сторон.

Проведение интервью, опросов, семинаров (workshop), мозговых штурмов.

Анализ существующих систем, документации, бизнес-процессов.

Документирование требований:

Создание спецификации требований (SRS - Software Requirements Specification) или другого документа, описывающего функциональные и нефункциональные требования.

Использование различных форматов: текстовое описание, Use Case Diagram, User Stories, Acceptance Criteria.

Анализ и уточнение требований:

Проверка требований на полноту, непротиворечивость, однозначность.

Разрешение конфликтов между требованиями.

Приоритизация требований (например, с использованием MoSCoW – Must have, Should have, Could have, Won’t have).

Валидация требований:

Представление требований заинтересованным сторонам для получения обратной связи и подтверждения.

Проведение обзоров требований (Requirements Reviews).

**Типы требований:**

Функциональные требования (Functional Requirements): Описывают, что система должна делать. (Например, “Система должна позволять пользователю зарегистрироваться”).

Нефункциональные требования (Non-Functional Requirements): Описывают как система должна работать, её характеристики. (Например, “Система должна отвечать на запросы пользователей в течение 2 секунд”). К ним относятся:

Производительность (Performance)

Надежность (Reliability)

Безопасность (Security)

Удобство использования (Usability)

Масштабируемость (Scalability)

Переносимость (Portability)

Поддержка (Maintainability)

Бизнес-требования (Business Requirements): Описывают цели и задачи бизнеса, которые система должна поддерживать.

Пользовательские требования (User Requirements): Описывают потребности пользователей системы.

**2. Тестирование Требований (Requirements Testing)**

**Цель:**

Проверить качество требований: полноту, непротиворечивость, однозначность, тестируемость, реализуемость.

Убедиться, что требования соответствуют бизнес-потребностям.

Выявить дефекты в требованиях на ранних стадиях разработки.

**Процесс:**

Разработка тест-кейсов на основе требований: Каждое требование должно быть покрыто тест-кейсами.

Проведение статического тестирования:

Обзоры требований (Requirements Reviews): Участники (аналитики, разработчики, тестировщики) читают и анализируют требования, выявляя недостатки.

Инспекции требований: Более формальный процесс обзора, с четкими ролями и этапами.

Проведение динамического тестирования (на более поздних этапах):

Выполнение тест-кейсов для проверки реализации требований.

Использование трассировки требований (Requirements Traceability Matrix) для отслеживания связи между требованиями, тест-кейсами и компонентами системы.

**Методы тестирования требований:**

Обзоры требований (Requirements Reviews): Наиболее распространенный и эффективный метод.

Разработка прототипов: Создание прототипа системы для проверки понимания требований.

Матрица трассировки требований (Requirements Traceability Matrix - RTM): Инструмент для отслеживания связи между требованиями, проектной документацией, кодом и тест-кейсами. Обеспечивает полное покрытие требований тестами.

Use Case Testing: Тестирование на основе Use Case диаграмм, фокусируясь на взаимодействии пользователя с системой.

Boundary Value Analysis (BVA): Анализ граничных значений для проверки валидации данных. Используется для функциональных требований, связанных с вводом данных.

Equivalence Partitioning: Разбиение входных данных на классы эквивалентности и тестирование только одного значения из каждого класса.

**Что тестировать в требованиях:**

Полнота: Убедиться, что все необходимые требования указаны.

Непротиворечивость: Убедиться, что требования не противоречат друг другу.

Однозначность: Убедиться, что каждое требование имеет только одно толкование.

Тестируемость: Убедиться, что для каждого требования можно разработать тест-кейсы.

Реализуемость: Убедиться, что требование может быть реализовано с учетом доступных ресурсов и технологий.

Прослеживаемость: Убедиться, что каждое требование можно отследить от его источника до реализации и тестирования.

**Билет №9**

**1 Организация тестирования. Фазы тестирования, основные проблемы тестирования.**

**2. Разработать модульные тесты для программного модуля ReadAndWrite, содержащий метод записи всех строк в файл. Код для программного модуля находится «Resorce/ModuleC#/». Оформить тест-кейсы для модульного тестирования.**

**Организация тестирования** - это систематический процесс планирования, разработки, выполнения и оценки тестов программного обеспечения с целью обнаружения дефектов и проверки соответствия требованиям.

**Ключевые аспекты организации тестирования:**

* Определение целей и задач тестирования: Четкое понимание, что нужно протестировать и зачем.
* Выбор методологии тестирования: Agile, Waterfall, V-Model и т.д.
* Разработка стратегии тестирования: Определение подходов к тестированию на разных уровнях.
* Планирование тестирования: Определение сроков, ресурсов, тестового окружения, тестовых данных, критериев начала и завершения тестирования.
* Подготовка тестовой документации: Тест-планы, тест-кейсы, чек-листы, отчеты о тестировании.
* Выбор инструментов тестирования: Инструменты автоматизации тестирования, управления тестами, отслеживания ошибок.
* Определение ролей и обязанностей: Тест-менеджер, тест-аналитик, разработчик тестов, тестировщик.
* Организация коммуникации: Обеспечение эффективного взаимодействия между членами команды тестирования, разработчиками и другими заинтересованными сторонами.
* Управление дефектами: Процесс отслеживания, приоритизации и исправления дефектов.
* Оценка результатов тестирования: Анализ тестовых данных, составление отчетов, предоставление рекомендаций по улучшению качества продукта.
* Непрерывное улучшение: Анализ процесса тестирования, выявление узких мест и внедрение улучшений.

**Фазы процесса тестирования:**

* Определение целей (требований к тестированию). Нужно конкретизировать, какие части системы будут тестироваться, какие аспекты их работы будут выбраны для проверки, каково желаемое качество и т. п..
* Планирование. Создаётся график разработки тестов для каждой тестируемой подсистемы, оцениваются необходимые человеческие, программные и аппаратные ресурсы, разрабатывается расписание тестовых циклов.
* Разработка тестов. Создаётся тестовый код для тестируемой системы, если необходимо — код системы автоматизации тестирования и тестовые процедуры (выполняемые вручную).
* Выполнение тестов. Реализуются тестовые циклы.
* Анализ результатов. Оцениваются выполнение требований, обнаруженные дефекты, покрытие тестами и эффективность тестирования.
* Завершение и отчётность. Составляется отчёт о выполненном тестировании, включающий информацию о проведённых тестах, обнаруженных дефектах, выполнении требований и других важных аспектах. Отчёт передаётся заинтересованным сторонам, таким как руководство проекта или заказчику.

**Основные проблемы тестирования:**

* Изменение требований. Руководство может существенно менять требования в середине спринта, что ведёт к изменению объёма тестирования.
* Неясные ожидания или цели. Если команда тестирования не имеет ясного представления о конечном результате, может получиться совсем не тот продукт, который ожидался.
* Отсутствие оценки качества. Часто у agile-команд нет количественного способа измерения общего качества продукта.
* Отсутствие сотрудничества между командами. Это приводит к постоянным простоям, поскольку каждая команда пытается подстроиться под то, что сделала другая.
* Время простоя разработчиков. Когда в тестировании возникают проблемы, разработчики обычно простаивают. Из-за таких простоев разработчикам часто приходится работать сверхурочно, чтобы завершить задачу.
* Проблемные тестовые данные. Плохое качество тестовых данных может существенно повлиять на стабильность и эффективность тестов.
* Нехватка времени и ресурсов: Ограниченные сроки и бюджет могут привести к недостаточному тестированию.
* Неполные или нечеткие требования: Отсутствие четких и полных требований затрудняет разработку тестовых случаев и оценку результатов тестирования.
* Сложность программного обеспечения: Сложные системы с большим количеством взаимосвязей требуют более тщательного и всестороннего тестирования.
* Недостаточная квалификация тестировщиков: Для эффективного тестирования требуется квалифицированный персонал, обладающий необходимыми знаниями и опытом.
* Отсутствие автоматизации тестирования: Ручное тестирование отнимает много времени и подвержено человеческим ошибкам.
* Трудности с моделированием реальных условий: Воссоздание реальных условий использования программного обеспечения в тестовой среде может быть сложным и дорогостоящим.
* Проблемы с тестовыми данными: Подготовка и управление тестовыми данными может быть трудоемким процессом.
* Сложность отслеживания и управления дефектами: Отслеживание и приоритизация дефектов, а также контроль за их исправлением могут быть сложными задачами.
* Недостаточная коммуникация: Плохая коммуникация между членами команды тестирования, разработчиками и другими заинтересованными сторонами может привести к недопониманию и ошибкам.
* Изменения в требованиях в процессе разработки: Частые изменения требований могут привести к необходимости переработки тестовой документации и повторному тестированию.
* Технические долги: Неустраненные вовремя ошибки и некачественный код усложняют процесс тестирования и увеличивают вероятность появления новых дефектов.
* Устаревшие инструменты и технологии: Использование устаревших инструментов и технологий может ограничивать возможности тестирования и снижать его эффективность.
* Отсутствие метрик качества: Отсутствие четких метрик качества затрудняет оценку эффективности тестирования и принятие решений о выпуске продукта.

\_\_\_\_\_

**Организация тестирования** - это комплекс процессов, направленных на обеспечение качества программного обеспечения (ПО) путем обнаружения и устранения дефектов. Эффективная организация тестирования требует планирования, координации ресурсов и применения подходящих методик.

**Фазы процесса тестирования**

В процессе тестирования выделяют следующие фазы:

Определение целей (требований к тестированию), включающее следующую конкретизацию: какие части системы будут тестироваться, какие аспекты их работы будут выбраны для проверки, каково желаемое качество и т.п.

Планирование: создание графика (расписания) разработки тестов для каждой тестируемой подсистемы; оценка необходимых человеческих, программных и аппаратных ресурсов; разработка расписания тестовых циклов. Важно отметить, что расписание тестирования обязательно должно быть согласовано с расписанием разработки создаваемой системы, поскольку наличие исполняемой версии разрабатываемой системы ( Implementation Under Testing (IUT) или Application Under Testing (AUT) – часто употребляемые обозначения для тестируемой системы) является одним из необходимых условий тестирования, что создает взаимозависимость в работе команд тестировщиков и разработчиков.

Разработка тестов, то есть тестового кода для тестируемой системы, если необходимо - кода системы автоматизации тестирования и тестовых процедур (выполняемых вручную).

Выполнение тестов: реализация тестовых циклов.

Анализ результатов.

**Организация тестирования включает следующие этапы:**

Проработка требований к продукту. Тестировщики внимательно изучают требования продукта — это могут быть документы, спецификации, описание того, как пользователь взаимодействует с продуктом. Чёткое понимание требований помогает определить области, которые нужно протестировать.

Анализ требований. Позволяет выяснить, какие возможные риски или сложности могут возникнуть при тестировании. Также на этом этапе можно выявить возможные несоответствия или недостаточно ясные требования, которые требуют уточнения у разработчиков или заказчика.

Разработка стратегии и плана тестирования. Когда все требования к продукту понятны, остаётся разработать план тестирования. В него входит выбор методов тестирования, анализ потенциальных рисков, которые могут повлиять на качество и успешность тестирования, и планирование мер по их минимизации.

Создание тестовой документации. На основе требований и анализа тестировщики создают тестовые случаи, тест-планы, отчётность и другую документацию, которая будет использоваться во время тестирования. Тестовая документация определяет, какие тесты будут проведены, как будут собраны результаты и как будет оценено качество ПО.

Тестирование. После того как команда утверждает стратегию тестирования и тестовую документацию, проводится тестирование.

Эксплуатация и поддержка. После того как разработчики устраняют дефекты и выпускают продукт, тестировщик переходит к тестированию продукта в рабочей среде.

**Некоторые основные проблемы тестирования:**

Неясные ожидания или цели. Если команда тестирования не имеет ясного представления о конечном результате, может получиться совсем не тот продукт, который ожидался.

Отсутствие оценки качества. Часто у agile-команд нет количественного способа измерения общего качества продукта. Команды используют отдельные показатели, такие как покрытие тестами и сложность кода, но эти элементы не дают полного представления о качестве.

Отсутствие сотрудничества между командами. Это приводит к постоянным простоям, поскольку каждая команда постоянно пытается подстроиться под то, что сделала другая.

Время простоя разработчиков. Когда в тестировании возникают проблемы, разработчики обычно простаивают. А при внесении изменений вновь приходится ждать результатов тестирования. Если продолжить работу над текущей задачей до этого, есть риск появления дополнительных ошибок.

\_\_\_\_\_

**Процесс тестирования программного обеспечения (ПО) состоит из нескольких ключевых фаз. Вот основные из них:**

Планирование и анализ требований: На этом этапе формируются и анализируются требования к ПО, разрабатывается стратегия тестирования, выбираются инструменты и ресурсы.

Разработка тестовых случаев: Создание тестовых сценариев и случаев на основе требований. Это помогает обеспечить полноту тестирования и охватывает все возможные сценарии использования.

Настройка тестовой среды: Создание и настройка среды, в которой будут выполняться тесты. Это может включать в себя установку ПО, настройку серверов и баз данных.

Выполнение тестов: Проведение самих тестов, как автоматических, так и ручных. На этом этапе выявляются дефекты и ошибки.

Отчеты и анализ результатов: Сбор и анализ данных, полученных в ходе тестирования. Создание отчетов о дефектах и предложений по их устранению.

Заключительное тестирование и релиз: Проведение финальных тестов для проверки исправлений и подготовки ПО к выпуску.

**Основные проблемы, с которыми можно столкнуться в процессе тестирования:**

Неопределенные требования: Если требования к ПО не полностью определены или постоянно меняются, это может затруднить разработку тестов и проведение тестирования.

Недостаток ресурсов: Недостаток времени, людей или оборудования может существенно повлиять на качество тестирования.

Сложность среды: Настройка тестовой среды может быть сложной и отнимать много времени.

Неполное тестирование: Невозможно протестировать все возможные сценарии, особенно в больших и сложных системах, поэтому всегда остается вероятность появления необнаруженных дефектов.

**Билет №10**

**1. Тестирование по методу «белого ящика».**

**2. Разработать модульные тесты для программного модуля Serializer, содержащий метод сериализации объекта в Json. Код для программного модуля находится «Resorce/ModuleC#/». Оформить тест-кейсы для модульного тестирования.**

**Тестирование “белого ящика”** (или структурное тестирование, тестирование прозрачного ящика) - это метод тестирования программного обеспечения, при котором тестовые случаи разрабатываются на основе знания внутренней структуры кода, его алгоритмов и логики. Тестировщик имеет доступ к исходному коду и использует эти знания для создания тестов, которые проверяют различные пути выполнения кода, условия и циклы.

**Основные характеристики тестирования “белого ящика”:**

* Знание исходного кода: Тестировщик должен понимать структуру и логику кода.
* Проверка путей выполнения: Тесты разрабатываются для проверки различных путей выполнения кода, включая условные переходы, циклы и обработку исключений.
* Покрытие кода: Целью является достижение высокого покрытия кода тестами, то есть максимального процента строк кода, которые выполняются тестами.
* Методы: Используются различные методы, такие как покрытие операторов, покрытие решений, покрытие условий, покрытие путей.

**Цели тестирования “белого ящика”:**

* Проверить правильность логики и алгоритмов: Убедиться, что код выполняет правильные вычисления и принимает верные решения.
* Проверить обработку ошибок: Убедиться, что код правильно обрабатывает исключения и ошибки.
* Проверить все возможные пути выполнения кода: Убедиться, что все части кода выполняются хотя бы один раз.
* Оптимизировать код: Выявить узкие места в коде и предложить улучшения для повышения производительности.

**Преимущества тестирования “белого ящика”:**

* Позволяет выявить скрытые ошибки: Ошибки, которые трудно обнаружить с помощью тестирования “черного ящика”.
* Оптимизирует код: Помогает выявить неэффективные участки кода.
* Улучшает понимание кода: Тестировщик лучше понимает код, что облегчает выявление ошибок.

**Билет №11**

**1. Тестирование по методу «черного ящика».**

**2. Разработать модульные тесты для программного модуля Serializer, содержащий метод десериализации файла Json в объект. Код для программного модуля находится «Resorce/ModuleC#/». Оформить тест-кейсы для модульного тестирования.**

**Тестирование “черного ящика”** (или функциональное тестирование, тестирование непрозрачного ящика) - это метод тестирования программного обеспечения, при котором тестовые случаи разрабатываются на основе знания спецификации и функциональности программы, без знания ее внутренней структуры, кода и алгоритмов. Тестировщик взаимодействует с программой как конечный пользователь, подавая на вход различные данные и проверяя соответствие выходных данных ожидаемым результатам, описанным в спецификации.

**Основные характеристики тестирования “черного ящика”:**

* Отсутствие знания исходного кода: Тестировщику не нужно знать, как реализована программа.
* Фокус на функциональность: Тесты разрабатываются для проверки того, что программа выполняет свои функции в соответствии со спецификацией.
* Использование спецификации: Тестировщик использует спецификацию, требования или пользовательские истории для разработки тестов.
* Различные техники: Используются различные техники, такие как анализ граничных значений, эквивалентное разделение, таблицы решений и тестирование на основе использования.

**Цели тестирования “черного ящика”:**

* Проверить соответствие требованиям: Убедиться, что программа выполняет все функции, описанные в спецификации.
* Выявить функциональные ошибки: Найти ошибки, связанные с неправильной работой функций программы.
* Проверить удобство использования: Оценить, насколько легко и удобно пользователю взаимодействовать с программой.
* Оценить производительность и надежность: Проверить, как быстро и надежно работает программа в различных условиях.

**Преимущества тестирования “черного ящика”:**

* Независимость от разработчиков: Тестировщики могут работать независимо от разработчиков.
* Легко понять и использовать: Не требует специальных знаний о коде.
* Обнаружение ошибок, связанных с требованиями: Помогает выявить ошибки, связанные с неправильным пониманием или реализацией требований.

**Билет №12**

**1. Модульное и интеграционное тестирование. Особенности модульного тестирования, подходы к тестированию на основе потока управления, потока данных. Организация модульного тестирования. валидности**

**2. Разработать модульные тесты для программного модуля Validation, содержащий метод проверки email. Код для программного модуля находится «Resorce/ModuleC#/». Оформить тест-кейсы для модульного тестирования.**

**Особенности модульного тестирования**

Модульное тестирование (unit testing) направлено на проверку отдельных компонентов или модулей системы. Основные особенности:

Изолированное тестирование: Модули тестируются отдельно друг от друга, что помогает выявить ошибки на ранней стадии разработки.

Автоматизация: Часто используется автоматизация тестов, что позволяет быстро проводить регрессионное тестирование.

Использование заглушек: Для изоляции модулей применяются заглушки (stubs) и мок-объекты (mocks), заменяющие реальные зависимости.

Подходы к тестированию на основе потока управления и потока данных

**Тестирование на основе потока управления:**

Фокусируется на проверке различных путей выполнения программы, включающих условия, циклы и ветвления.

Используются методы, такие как покрытие условий и покрытие ветвей, чтобы гарантировать полное покрытие кода.

Тестирование на основе потока данных:

Внимание уделяется переменным и их состояниям в ходе выполнения программы.

Проверяются все пути данных, по которым переменные принимают значения и используются в выражениях.

**Организация модульного тестирования**

**Организация модульного тестирования включает несколько этапов:**

Подготовка тестовой среды: Создание необходимых тестовых данных, заглушек и мок-объектов.

Разработка тестовых случаев: Определение наборов тестов, покрывающих различные сценарии и условия.

Запуск тестов и анализ результатов: Проведение тестирования и анализ полученных результатов на предмет ошибок и проблем.

Регрессионное тестирование: Повторное тестирование после внесения изменений в код, чтобы убедиться, что исправления не вызвали новых ошибок.

Интеграционное тестирование

Интеграционное тестирование (integration testing) направлено на проверку взаимодействия между различными модулями или компонентами системы. Оно позволяет убедиться, что компоненты корректно работают вместе после их объединения.

**Особенности интеграционного тестирования:**

Иерархическое тестирование: Тестирование начинается с небольших групп модулей и постепенно расширяется до всей системы.

Использование драйверов и заглушек: Для изоляции отдельных компонентов используются специальные драйверы и заглушки, заменяющие недостающие компоненты.

Обнаружение интерфейсных ошибок: Интеграционное тестирование позволяет выявить ошибки в точках взаимодействия между модулями, которые могут не проявиться при модульном тестировании.

**Подходы к интеграционному тестированию:**

**Нисходящее тестирование:**

Тестирование начинается с самых верхних уровней системы и постепенно спускается к нижним.

Используются заглушки для имитации нижестоящих модулей, пока они не будут реализованы.

**Восходящее тестирование:**

Тестирование начинается с самых нижних уровней и постепенно поднимается к верхним.

Используются драйверы для имитации вышестоящих модулей, пока они не будут реализованы.

**Тестирование в виде "сандвича":**

Сочетает элементы нисходящего и восходящего тестирования.

Тестирование начинается одновременно с верхних и нижних уровней, постепенно сходясь к середине.

**Большой взрыв:**

Все модули интегрируются сразу, и затем проводится тестирование.

Это подход может быть рисковым, так как все ошибки приходится искать и исправлять одновременно.

**Организация интеграционного тестирования:**

Подготовка тестовой среды: Настройка окружения, включающего все необходимые компоненты и данные.

Разработка тестовых сценариев: Определение наборов тестов, охватывающих взаимодействие между модулями и системы в целом.

Запуск тестов и анализ результатов: Проведение тестирования и анализ полученных результатов на предмет ошибок и проблем взаимодействия.

Регрессионное тестирование: Повторное тестирование после внесения изменений, чтобы убедиться в отсутствии новых ошибок.

**Билет №13**

**1. Динамические и статические методы при структурном и объектном подходах тестирования. Валидности**

**2. Разработать модульные тесты для программного модуля Validation, содержащий метод проверки email. Код для программного модуля находится «Resorce/ModuleC++/». Оформить тест-кейсы для модульного тестирования.**

**Динамические методы тестирования**

Динамическое тестирование включает выполнение кода и наблюдение за его поведением во время выполнения. Ключевые виды динамического тестирования включают:

**Функциональное тестирование:**

Проверяет, что система соответствует требованиям и спецификациям.

Включает тестирование на уровне модулей, интеграционное, системное и приемочное тестирование.

**Нефункциональное тестирование:**

Проверяет аспекты, не связанные с функциональностью системы, такие как производительность, безопасность и удобство использования.

Примеры включают нагрузочное тестирование, стресс-тестирование и тестирование на удобство использования.

**Статические методы тестирования**

Статическое тестирование не требует выполнения кода. Оно фокусируется на анализе кода и документации. Основные виды статического тестирования включают:

**Рецензирование (Review):**

Включает анализ кода, проектной документации и требований.

Варианты: формальное рецензирование, неформальное рецензирование, парное программирование.

**Анализ кода:**

Использует автоматизированные инструменты для анализа кода на предмет ошибок, нарушений стиля и потенциальных уязвимостей.

Примеры включают статический анализ кода, анализ потока данных и метрики программного обеспечения.

**Структурный подход к тестированию**

Структурный подход фокусируется на тестировании внутренней структуры системы. Ключевые аспекты:

**Тестирование на основе покрытия кода:**

Покрытие условий, ветвей, путей и операторов.

Гарантирует, что тесты охватывают различные части кода.

**Тестирование потоков данных:**

Анализирует, как данные передаются и обрабатываются внутри системы.

Выявляет ошибки, связанные с неправильным использованием данных.

Объектно-ориентированный подход к тестированию

**Объектно-ориентированный подход учитывает особенности объектно-ориентированного программирования. Основные моменты:**

**Тестирование классов и объектов:**

Проверка методов, атрибутов и взаимодействий объектов.

Использование мок-объектов для изоляции тестируемых классов.

**Тестирование наследования и полиморфизма:**

Проверка корректности наследуемых методов и переопределений.

Тестирование различных реализаций интерфейсов и абстрактных классов.

**Тестирование взаимодействий:**

Анализ взаимодействий между объектами и их зависимостей.

Использование диаграмм последовательностей для идентификации и тестирования взаимодействий.

**\_\_\_\_\_**

**Динамические методы тестирования**

Динамическое тестирование включает выполнение кода программы для проверки его работы и выявления дефектов.

**Структурный подход:**

* Покрытие операторов: Убедиться, что каждый оператор в коде выполняется хотя бы один раз.
* Покрытие ветвей: Проверка всех возможных ветвей условных операторов.
* Покрытие путей: Тестирование всех возможных путей выполнения программы.
* Покрытие условий: Проверка всех логических условий в коде.

**Объектно-ориентированный подход:**

* Тестирование классов и методов: Проверка корректности работы отдельных классов и методов.
* Тестирование взаимодействий: Проверка взаимодействий между объектами.
* Инкапсуляция: Проверка корректного использования инкапсуляции и взаимодействия между компонентами через открытые интерфейсы.
* Статические методы тестирования
* Статическое тестирование проводится без выполнения кода, путем анализа документации, кода и других артефактов.

**Статическое тестирование**

**Структурный подход**:

* Ревью кода: Проверка исходного кода на соответствие стандартам и выявление ошибок.
* Анализ кода (статический анализ): Использование инструментов статического анализа для выявления потенциальных дефектов в коде.
* Инспекции кода: Проверка кода другими разработчиками на предмет ошибок и соответствия стандартам кодирования.
* Проверка соответствия стандартам: Убедиться, что код соответствует установленным стандартам и требованиям.

**Объектно-ориентированный подход:**

* Ревью дизайна классов: Анализ дизайна классов и их взаимодействий для выявления потенциальных проблем.
* Инспекции кода: (Аналогично структурному подходу)
* Проверка UML-диаграмм: Анализ диаграмм классов, последовательностей и других UML-диаграмм для проверки соответствия требованиям.
* Анализ зависимостей: Проверка корректности зависимостей между классами и компонентами.

**Преимущества и недостатки**

**Динамическое тестирование:**

* Преимущества: Обнаружение дефектов во время выполнения программы, проверка реального поведения системы.
* Недостатки: Требует больше времени и ресурсов, зависит от качества тест-кейсов.

**Статическое тестирование:**

* Преимущества: Обнаружение дефектов на ранних стадиях разработки, экономия времени на исправление ошибок.

Недостатки: Не выявляет ошибки, возникающие во время выполнения программы.

**Билет №14**

**1. Тестирование по функциональному критерию.**

**2. Разработать модульные тесты для программного модуля Validation, содержащий метод проверки валидности пароля. Код для программного модуля находится «Resorce/ModuleC#/». Оформить тест-кейсы для модульного тестирования.**

Функциональное тестирование — это метод тестирования программного обеспечения, целью которого является проверка того, что программный продукт работает в соответствии с требуемыми функциональными спецификациями. Этот тип тестирования проверяет, что каждый аспект системы функционирует в соответствии с требованиями.

**Основные этапы функционального тестирования включают:**

* Анализ требований: изучение и понимание функциональных требований.
* Планирование тестов: определение, какие тесты необходимо выполнить.
* Разработка тест-кейсов: создание сценариев тестирования, которые охватывают все функциональные аспекты.
* Выполнение тестов: запуск тест-кейсов и наблюдение за их выполнением.
* Анализ результатов: сравнение полученных результатов с ожидаемыми.
* Отчет о тестировании: документирование результатов тестирования и выявление проблем.

Функциональное тестирование включает различные виды тестирования, такие как модульное тестирование, интеграционное тестирование, системное тестирование и приемочное тестирование.

\_\_\_\_\_

Функциональный критерий тестирования обеспечивает контроль степени выполнения требований заказчика в программном продукте. Требования формулируются к продукту в целом, они отражают взаимодействие тестируемого приложения с окружением.

**Основные элементы функционального тестирования**

Проверка корректности: Убедиться, что все функции системы выполняются корректно и выдают ожидаемые результаты.

Проверка полноты: Убедиться, что все заявленные функции реализованы и доступны пользователю.

Интерфейсное тестирование: Проверка взаимодействия между различными компонентами системы и их интеграции.

Тестирование потока данных: Проверка корректности обработки данных на различных этапах работы системы.

Тестирование пользовательских интерфейсов: Проверка удобства использования и функциональности пользовательских интерфейсов.

Регрессионное тестирование: Убедиться, что внесенные изменения не нарушили работу существующих функций.

**Подходы к функциональному тестированию**

Черный ящик: Тестировщики проверяют систему, не зная внутренней структуры кода. Они ориентируются только на входные и выходные данные.

Белый ящик: Тестировщики имеют доступ к исходному коду и проверяют внутренние механизмы работы системы.

Серый ящик: Комбинация двух подходов, когда тестировщики имеют ограниченный доступ к информации о внутренней структуре системы.

**Виды функционального тестирования**

**Модульное тестирование (Unit Testing):**

Проверяет отдельные модули или компоненты кода на их корректную работу.

Обычно проводится разработчиками.

**Интеграционное тестирование (Integration Testing):**

Проверяет взаимодействие между различными модулями или компонентами.

Цель – выявить ошибки в интеграции и взаимодействии.

**Системное тестирование (System Testing):**

Проверяет всю систему в целом, чтобы убедиться, что она соответствует требованиям.

Включает функциональные и нефункциональные аспекты.

**Приемочное тестирование (Acceptance Testing):**

Проверяет систему с точки зрения пользователя.

Обычно выполняется для подтверждения, что продукт готов к выпуску.

**Регрессионное тестирование (Regression Testing):**

Проверяет, что изменения или исправления кода не нарушили существующую функциональность.

Включает повторное выполнение ранее выполненных тест-кейсов.

**Санитарное тестирование (Sanity Testing):**

Быстрое тестирование для проверки основных функций системы после небольших изменений.

Цель – убедиться, что система работает на базовом уровне.

**Дымовое тестирование (Smoke Testing):**

Проверяет основные функции системы, чтобы убедиться, что она "не падает".

Обычно выполняется при каждом новом билде.

**Бета-тестирование (Beta Testing):**

Проверяет продукт в реальных условиях с участием конечных пользователей.

Помогает выявить проблемы, которые могут возникнуть при реальной эксплуатации.

**Билет №15**

**1. Тестирование производительности.**

**2. Разработать модульные тесты для программного модуля Validation, содержащий метод проверки валидности номера телефона. Код для программного модуля находится «Resorce/ModuleC#/». Оформить тест-кейсы для модульного тестирования.**

**Тестирование производительности** – это тип тестирования программного обеспечения, который направлен на оценку скорости, стабильности и масштабируемости программного продукта в различных условиях нагрузки. Целью является выявление узких мест, которые могут привести к снижению производительности, и обеспечение соответствия продукта требованиям к быстродействию и ресурсоемкости.

**Основные виды тестирования производительности:**

Нагрузочное тестирование (Load Testing): Проверка производительности системы при ожидаемой нагрузке. Цель - определить, может ли система обрабатывать ожидаемый объем трафика и транзакций.

Стрессовое тестирование (Stress Testing): Проверка производительности системы при экстремальных нагрузках. Цель - определить предел устойчивости системы и ее поведение при отказе.

Тестирование стабильности (Endurance Testing/Soak Testing): Проверка производительности системы в течение длительного периода времени при нормальной нагрузке. Цель - выявить утечки памяти, проблемы с ресурсами и другие факторы, которые могут привести к снижению производительности с течением времени.

Тестирование масштабируемости (Scalability Testing): Проверка способности системы масштабироваться для обработки возрастающей нагрузки. Цель - определить, как изменяется производительность системы при увеличении количества пользователей, транзакций или данных.

Тестирование объема (Volume Testing): Проверка производительности системы при работе с большим объемом данных.

**Основные этапы тестирования производительности**

Определение целей: Определение ключевых показателей производительности (KPIs), таких как время отклика, пропускная способность и использование ресурсов.

Планирование тестов: Разработка сценариев тестирования, определение тестовых данных и инструментов для проведения тестирования.

Подготовка тестовой среды: Настройка тестовой инфраструктуры, обеспечение соответствия тестовой среды реальным условиям эксплуатации.

Выполнение тестов: Запуск тестов и сбор данных о производительности системы.

Анализ результатов: Анализ собранных данных для выявления проблем и узких мест в системе.

Оптимизация: Внесение изменений для улучшения производительности и повторное тестирование для оценки эффективности этих изменений.

**Инструменты для тестирования производительности**

JMeter: Инструмент с открытым исходным кодом для нагрузки и стресс-тестирования веб-приложений.

LoadRunner: Коммерческий инструмент для проведения различных видов тестирования производительности.

Gatling: Инструмент для нагрузочного тестирования с акцентом на высокую производительность и удобство использования.

Apache Benchmark (ab): Простой инструмент для нагрузочного тестирования веб-серверов.

**Билет №16**

**1. Средства разработки технической документации.**

**2. Разработать модульные тесты для программного модуля Collection, содержащий методы работы с коллекцией. Код для программного модуля находится «Resorce/ModuleC++/». Оформить тест-кейсы для модульного тестирования.**

**Средства разработки технической документации при тестировании программного обеспечения (ПО):**

План тестирования. Это документ, описывающий весь объём работ по тестированию, включая объект, стратегию, расписание, критерии начала и окончания тестирования, а также необходимое оборудование и оценку рисков.

Тест-дизайн. Процесс проектирования и создания тестовых случаев для проверки ПО с учётом спецификации проекта и требований.

Тестовый случай. Строго сформированный документ с подробным описанием шагов и действий для проверки одной части функционала, со строго заданными критериями прохождения.

Тест-кейс. Документ для тестирования продукта или ПО, который используется для предугадывания и построения на основе предполагаемых сценариев использования путей тестирования приложения.

Чек-лист. Документ, который содержит краткое описание функциональности всего приложения для проверки ПО. Создание и написание чек-листа обычно заменяет написание тестовых случаев и ускоряет подготовку к проведению тестирования.

Спецификация требований. Полное, чёткое описание разрабатываемой программы.

**Примеры программ для автоматической документации:**

Sphinx – инструмент для создания документации, который особенно популярен в Python-сообществе. Он позволяет легко генерировать документацию из исходного кода.

Doxygen – мощный инструмент для автоматического создания документации на основе комментариев в исходном коде. Поддерживает множество языков программирования, включая C++, C, Java и другие.

MkDocs – еще один популярный инструмент, который позволяет создавать статические сайты для документации. Поддерживает форматирование Markdown и легко настраивается.

Swagger/OpenAPI – используются для создания и документирования RESTful API. Эти инструменты позволяют генерировать документацию автоматически на основе определений API.

Javadoc – стандартный инструмент для документирования Java-программ. Он генерирует HTML-документацию на основе комментариев в исходном коде.

AsciiDoc – гибкий текстовый формат для написания технической документации, который может быть легко преобразован в различные форматы, такие как HTML, PDF и другие.

\_\_\_\_\_

Для разработки технической документации используется множество инструментов, которые помогают упрощать процесс создания, редактирования и управления документацией. Вот некоторые из наиболее популярных:

**Текстовые редакторы и процессоры документов:**

Microsoft Word: Широко используемый текстовый редактор с поддержкой стилей, шаблонов и форматирования.

Google Docs: Облачный текстовый редактор, обеспечивающий совместную работу в реальном времени и доступ к документам из любого места.

**Средства для создания структурированных документов:**

LaTeX: Система верстки документов, особенно популярная для создания научных и технических документов благодаря поддержке сложных математических формул и структурированных документов.

MadCap Flare: Инструмент для создания комплексной технической документации, включая онлайн-руководства и печатные документы, с поддержкой многоканальной публикации.

**Платформы для совместной работы:**

Confluence: Платформа для совместной работы от Atlassian, часто используемая для создания документации и вики-страниц, с интеграцией с другими инструментами разработки.

Notion: Универсальный инструмент для создания документации, заметок и управления проектами с возможностью совместной работы.

**Специализированные средства для генерации документации:**

Doxygen: Инструмент для автоматической генерации документации из исходного кода, поддерживающий различные языки программирования, такие как C++, Java и Python.

Sphinx: Генератор документации для Python-проектов с поддержкой reStructuredText, позволяющий создавать как HTML, так и PDF-версии документации.

**Средства для разметки и форматирования:**

Markdown: Легкий язык разметки для создания текста с форматированием, поддерживаемый множеством платформ и инструментов.

AsciiDoc: Язык разметки для создания текстов, схожий с Markdown, но с большими возможностями для создания сложных документов.

**Средства для управления версиями и публикации:**

GitBook: Платформа для создания и публикации документации на основе Markdown с поддержкой версионного контроля и совместной работы.

Read the Docs: Сервис для хостинга и публикации документации, интегрированный с инструментами для генерации документации, такими как Sphinx.

**Билет №17**

**1. Системное тестирование.**

**2. Разработать модульные тесты для программного модуля CreateProcess, содержащий методы создания процесса. Код для программного модуля находится «Resorce/ModuleC++/». Оформить тест-кейсы для модульного тестирования.**

**Системное тестирование** — это комплексная процедура в процессе разработки программного обеспечения, которая оценивает готовность и функциональность всей системы или приложения в целом.

**Цель системного тестирования** — убедиться, что продукт соответствует техническим требованиям и работает без ошибок.

В процессе системного тестирования применяют метод «чёрного ящика». Тестировщик опирается на документацию, но не знает, как работает код. Его задача — воссоздать условия и сценарии, близкие к реальности, и проверить, как система себя поведёт.

**Некоторые виды системного тестирования:**

Функциональное. Проверка функций системы, которые указаны в документации.

Нагрузочное. Проверка работы системы в ситуациях, когда резко подскакивает количество пользователей.

Восстановительное. Проверка способности системы к восстановлению после сбоев.

Совместимости. Проверка работы системы на различных платформах и совместимость с другими системами.

Производительности. Оценка работы системы при нормальных условиях и при различных уровнях нагрузки.

Безопасности. Насколько система защищена от злоумышленников.

Удобства использования. Насколько пользователю понятен интерфейс, может ли он ориентироваться в нём без подсказок.

Конфигурации. Проверка работы системы с учётом разных параметров.

Отказоустойчивости. Проверка работы системы в ситуациях, когда какие-то компоненты не работают или, случился форс-мажор.

**Подходы к системному тестированию:**

**Тестирование на основе сценариев:**

Создание и выполнение сценариев тестирования, охватывающих различные аспекты работы системы.

Использование реальных сценариев использования для проверки соответствия системы требованиям пользователей.

**Регресс-тестирование:**

Повторное выполнение тестов после внесения изменений в систему для проверки того, что исправления или новые функции не вызвали новых дефектов.

**Автоматизация тестирования:**

Использование инструментов автоматизированного тестирования для выполнения повторяющихся тестов и ускорения процесса тестирования.

Selenium, Appium, JMeter и другие инструменты могут быть использованы для автоматизации тестов.

\_\_\_\_\_

**Системное тестирование** – это важный этап в процессе разработки программного обеспечения, направленный на проверку всей системы в целом. Оно включает в себя несколько ключевых этапов и методов, каждый из которых имеет свои задачи и особенности. Вот несколько основных компонентов системного тестирования:

**1. Планирование тестирования**

План тестирования включает описание всех работ, которые необходимо выполнить, и их последовательность. Это включает в себя:

Цели и задачи тестирования

Критерии начала и окончания тестирования

Ресурсы, необходимые для тестирования

Расписание проведения тестирования

**2. Тест-дизайн**

Процесс тест-дизайна включает разработку и создание тестовых случаев, которые покрывают все функциональные и нефункциональные требования системы. Это включает:

Определение входных данных

Описание шагов для выполнения теста

Определение ожидаемых результатов

**3. Выполнение тестирования**

Во время выполнения тестирования происходит непосредственная проверка системы с помощью разработанных тестовых случаев. Это этап включает:

Запуск тестов

Запись результатов выполнения тестов

Сравнение фактических результатов с ожидаемыми

**4. Анализ результатов**

После выполнения тестов необходимо провести анализ полученных результатов. Это включает:

Оценку успешности тестов

Идентификацию и анализ найденных дефектов

Предложение улучшений для последующих циклов тестирования

**5. Отчётность и документация**

На последнем этапе системного тестирования составляется отчет о проведенном тестировании, который включает:

Список выполненных тестов

Список найденных дефектов и их текущий статус

Рекомендации по исправлению дефектов

**Методы системного тестирования**

Функциональное тестирование: проверка функциональных требований системы.

Тестирование производительности: проверка скорости работы и устойчивости системы под нагрузкой.

Тестирование безопасности: проверка системы на уязвимости и устойчивость к различным атакам.

Тестирование совместимости: проверка работы системы на различных платформах и устройствах.

\_\_\_\_\_

Системное тестирование – это этап тестирования программного обеспечения, который охватывает проверку всей системы целиком. Цель системного тестирования – убедиться, что все компоненты и модули работают корректно в интегрированной системе и что система соответствует требованиям и ожиданиям пользователей. Вот основные аспекты системного тестирования:

**Основные цели системного тестирования**

Проверка функциональности: Убедиться, что все функциональные требования выполнены, и система работает правильно.

Проверка безопасности: Убедиться, что система защищена от потенциальных угроз и уязвимостей.

Проверка производительности: Убедиться, что система функционирует стабильно и эффективно под нагрузкой.

Проверка совместимости: Убедиться, что система совместима с различными аппаратными и программными средами.

Проверка пользовательского интерфейса: Убедиться, что пользовательский интерфейс удобен и соответствует ожиданиям пользователей.

**Виды системного тестирования**

Функциональное тестирование: Проверка соответствия системы функциональным требованиям.

Нагрузочное тестирование: Оценка производительности системы под разной нагрузкой.

Стресс-тестирование: Проверка поведения системы при экстремально высокой нагрузке.

Тестирование безопасности: Поиск уязвимостей и проверка защиты данных.

Тестирование на совместимость: Проверка работы системы на различных платформах и в различных конфигурациях.

Регрессионное тестирование: Проверка, что внесенные изменения не нарушили работу существующих функций.

**Этапы системного тестирования**

Подготовка тестового плана: Определение целей, подходов и методов тестирования.

Разработка тестовых сценариев: Создание сценариев тестирования на основе требований и спецификаций.

Подготовка тестовой среды: Настройка тестовой инфраструктуры и создание необходимых данных.

Выполнение тестов: Запуск тестовых сценариев и сбор данных о работе системы.

Анализ результатов: Анализ собранных данных и выявление дефектов.

Отчет о результатах тестирования: Составление отчета с описанием выявленных дефектов и предложением по их исправлению.

**Билет №18**

**1. Регрессионное тестирование: особенности и виды регрессионного тестирования, методы отбора тестов, оценка эффективности.**

**2. Разработать модульные тесты для программного модуля Equation, содержащий метод решения квадратного уравнения. Код для программного модуля находится «Resorce/ModuleC++/». Оформить тест-кейсы для модульного тестирования.**

Регрессионное тестирование – это вид тестирования программного обеспечения, направленный на проверку работоспособности системы после внесения изменений. Его цель – убедиться, что обновления, исправления и новые функции не нарушили работу существующего функционала.

**Особенности регрессионного тестирования**

Периодичность: Регрессионное тестирование проводится после каждого изменения в коде, будь то исправление ошибки или добавление новой функции.

Масштаб: Оно охватывает всю систему или отдельные ее модули, в зависимости от масштаба изменений.

Автоматизация: Часто используется автоматизированное тестирование для быстрого и эффективного выполнения множества тестов.

Приоритеты: Тесты обычно ранжируются по степени риска, связанному с изменениями.

**Виды регрессионного тестирования**

Коррекционное регрессионное тестирование: Проверка системы после исправления ошибок.

Прогрессивное регрессионное тестирование: Проверка нового функционала и его взаимодействия с уже существующими функциями.

Избирательное регрессионное тестирование: Тестирование только тех частей системы, которые напрямую связаны с изменениями.

Полное регрессионное тестирование: Полный пересмотр всей системы после значительных обновлений.

**Методы отбора тестов для регрессионного тестирования**

Анализ кода: Выявление зависимых модулей и функций, которые могли быть затронуты изменениями.

Анализ влияния изменений: Определение частей системы, которые могут быть затронуты в результате изменений в коде.

Анализ покрытия: Использование покрытия кода для определения, какие тесты уже покрывают изменения.

Использование тестовых приоритетов: Ранжирование тестов по приоритету в зависимости от риска и критичности функционала.

**Оценка эффективности регрессионного тестирования**

Покрытие тестов: Процент кода, покрытый тестами.

Количество найденных дефектов: Количество ошибок, выявленных в результате тестирования.

Скорость выполнения: Время, затраченное на выполнение тестов.

Автоматизация: Уровень автоматизации тестов и его влияние на скорость и качество тестирования.

**Билет №19**

**1. Документирование процесса тестирования. Тестовые процедуры программного продукта**

**2. Разработать модульные тесты для программного модуля ReadAndWrite, содержащий методы чтения и записи в файл. Код для программного модуля находится «Resorce/ModuleC++/». Оформить тест-кейсы для модульного тестирования.**

**Документирование процесса тестирования** – это важная часть обеспечения качества программного обеспечения. Оно позволяет отслеживать ход тестирования, анализировать результаты и использовать полученный опыт для улучшения будущих процессов тестирования.

**Основные документы процесса тестирования:**

**План тестирования:**

Описание целей, стратегий, объема и методов тестирования.

Распределение ролей и обязанностей в команде тестировщиков.

Определение ресурсов и временных рамок.

**Тестовые сценарии (Test Scenarios):**

Описание высокоуровневых функций или компонентов системы, которые должны быть протестированы.

Определение областей, требующих внимания.

**Тест-кейсы (Test Cases):**

Подробные инструкции по выполнению тестов, включая входные данные, шаги выполнения и ожидаемые результаты.

Уникальные идентификаторы для каждого тест-кейса для удобства отслеживания.

**Тестовые данные:**

Наборы данных, используемые для выполнения тестов.

Описание условий и сценариев использования данных.

**Отчеты о дефектах (Bug Reports):**

Описание обнаруженных дефектов, включая шаги воспроизведения, ожидаемые и фактические результаты.

Уникальные идентификаторы для каждого дефекта.

**Отчеты о тестировании (Test Reports):**

Обобщение результатов тестирования, включая выполненные тесты, обнаруженные дефекты и статус тестирования.

Рекомендации по улучшению качества программного продукта.

Документирование процесса тестирования включает в себя создание и поддержку документов, которые описывают все шаги, процедуры и результаты тестирования программного продукта. Это может включать в себя планы тестирования, тестовые сценарии, тестовую документацию, отчеты о выполненных тестах и другие связанные документы.

**Тестовые процедуры** – это пошаговые инструкции, которые описывают, как выполнять тесты для проверки определенной функциональности или компонента системы.

**Тестовые процедуры программного продукта включают следующие этапы:**

Планирование. Определение целей, масштаба и графика тестирования.

Проектирование тест-кейсов. Создание сценариев для проверки конкретных функций и функциональных областей.

Выполнение тестов. Использование ручных или автоматизированных методов для запуска и выполнения тест-кейсов.

Отчётность об ошибках. Документирование и передача обнаруженных дефектов соответствующей команде.

Анализ. Оценка результатов, определение покрытия и выявление областей для улучшения. 1

Документирование. Фиксация результатов тестирования и процесса, гарантирующая прозрачность и прослеживаемость.

**Некоторые виды тестовых процедур программного продукта:**

Функциональное тестирование. Проверяет соответствие программы или системы заранее определённым функциональным требованиям и ожиданиям.

Нефункциональное тестирование. Проверяет нефункциональные аспекты программы — производительность, безопасность, надёжность, масштабируемость и совместимость.

Статическое тестирование. Выполняется без запуска программы, тестировщики анализируют исходный код или другие составляющие, например, документацию.

Динамическое тестирование. Выполняется во время работы программы.

Ручное тестирование. Тестировщик взаимодействует с программой как обычный пользователь.

Автоматизированное тестирование. Проверка программного обеспечения с использованием специальных программных инструментов, которые выполняют тесты автоматически, без участия человека.

\_\_\_\_\_

**Документирование процесса тестирования** – это важная часть обеспечения качества программного обеспечения. Оно позволяет отслеживать ход тестирования, анализировать результаты и использовать полученный опыт для улучшения будущих процессов тестирования.

Документирование процесса тестирования помогает команде однозначно трактовать шаги, сроки тестирования, результаты, обращаться к этой информации в спорных моментах.

**Некоторые виды документации, используемой для документирования процесса тестирования:**

План тестирования. Содержит критерии начала и окончания тестирования, описание конкретных параметров: что именно подлежит тестированию, с помощью каких техник, на каких платформах будет проверяться функционал.

Тест-кейс. Это набор условий, действий и ожидаемых результатов, направленных на проверку какого-либо функционала.

Чек-лист. Содержит перечень элементов, которые подлежат тестированию: блоки, секции, страницы и другие.

Юзкейс. Содержит сценарии взаимодействия пользователя с системой, описание того, что именно делает программа.

Баг-репорт. Это документ, в котором содержится полная информация о найденном баге (шаги воспроизведения, описание, локализация и т.д.).

Отчёт по тестированию. Отчёт о проделанной работе с описанием результатов. Может содержать текст, таблицы, графики и диаграммы.

**Тестовые процедуры программного продукта включают несколько этапов:**

Планирование тестирования. На этом этапе определяются цели и задачи тестирования, разрабатывается план, включающий в себя выбор методологий, инструментов и ресурсов. Также проводится оценка рисков и определение критериев завершения тестирования.

Разработка тестовых сценариев. Это наборы условий и действий, которые должны быть выполнены для проверки функциональности ПО. На этом этапе создаются тестовые случаи, которые охватывают все возможные сценарии использования продукта.

Подготовка тестовой среды. Тестовая среда должна быть настроена таким образом, чтобы максимально точно имитировать рабочие условия ПО. Это включает установку необходимого оборудования, программного обеспечения и данных.

Выполнение тестирования. На этом этапе проводятся тесты в соответствии с разработанными сценариями. Тестировщики выполняют тестовые случаи, фиксируют результаты и выявляют дефекты.

Анализ результатов и отчётность. После выполнения тестов результаты анализируются, составляются отчёты о найденных дефектах и их критичности. Эти отчёты помогают разработчикам исправить ошибки и улучшить качество продукта.

Регрессное тестирование. После исправления дефектов проводится регрессное тестирование, чтобы убедиться, что внесённые изменения не повлияли на уже работающий функционал.

**Билет №20**

**1. Документирование и жизненный цикл дефекта.**

**2. Разработать модульные тесты для мобильного приложения Calculate, выполнить тестирование интерфейса. Приложение находится «Resorce/Calc/». Оформить тест-кейсы для модульного тестирования.**

**Документирование дефекта** заключается в том, что каждый дефект, обнаруженный в процессе тестирования, заносится в базу дефектов. Для этого используют специализированные базы, поддерживающие хранение и отслеживание дефектов. При занесении нового дефекта указывают, как минимум, следующую информацию:

наименование подсистемы, в которой обнаружен дефект;

версию продукта, на которой дефект был найден;

описание дефекта;

описание процедуры (шагов, необходимых для воспроизведения дефекта);

номер теста, на котором дефект был обнаружен;

уровень дефекта, то есть степень его серьёзности с точки зрения критериев качества продукта или заказчика.

**Жизненный цикл дефекта** — это набор определённых этапов, через которые проходит дефект от выявления до устранения. Занесённый в базу дефектов новый дефект находится в состоянии «New». После того, как команда разработчиков проанализирует дефект, он переводится в состояние «Open» с указанием конкретного разработчика, ответственного за исправление дефекта. После исправления дефект переводится разработчиком в состояние «Resolved». При этом разработчик должен указать следующую информацию:

причину возникновения дефекта;

место исправления, как минимум, с точностью до исправленного файла;

краткое описание того, что было исправлено;

время, затраченное на исправление.

**Документирование и жизненный цикл дефекта** - это важные процессы в управлении качеством программного обеспечения. Давайте рассмотрим основные этапы жизненного цикла дефекта:

Обнаружение дефекта: В этом этапе дефект обнаруживается пользователями, тестировщиками или разработчиками. Он может быть выявлен в процессе тестирования или эксплуатации системы.

Регистрация дефекта: Обнаруженный дефект регистрируется в системе отслеживания дефектов (например, Jira, Bugzilla). Важно указать все необходимые данные, такие как описание дефекта, шаги воспроизведения, ожидаемый и фактический результат, а также приоритет и серьезность дефекта.

Анализ дефекта: На этом этапе дефект анализируется командой разработчиков и тестировщиков для определения причины проблемы и возможных решений. Важно понять, как дефект влияет на систему и какие изменения необходимы для его исправления.

Исправление дефекта: Разработчики вносят изменения в код для исправления дефекта. После этого код проходит тестирование, чтобы убедиться, что дефект был исправлен и не вызвал новых проблем.

Повторное тестирование: После исправления дефект проходит повторное тестирование, чтобы убедиться, что он действительно был исправлен и система работает правильно. Это может включать регрессионное тестирование, чтобы проверить, не возникли ли новые дефекты.

Закрытие дефекта: Если дефект успешно исправлен и повторное тестирование прошло без проблем, дефект может быть закрыт в системе отслеживания дефектов. Важно задокументировать все действия, связанные с исправлением дефекта, чтобы иметь историю и возможность последующего анализа.

\_\_\_\_\_

Документирование и управление дефектами являются важной частью процесса разработки программного обеспечения, так как они помогают отслеживать и устранять ошибки на всех этапах проекта. Давайте рассмотрим, как документирование дефектов и их жизненный цикл могут быть организованы.

**Этапы жизненного цикла дефекта**

Идентификация: Обнаружение дефекта тестировщиком или пользователем.

Регистрация: Запись дефекта в систему отслеживания дефектов с указанием подробностей (идентификатор, описание, шаги для воспроизведения, ожидаемые и фактические результаты, приоритет, серьезность, окружение).

Классификация: Оценка дефекта и присвоение ему приоритета и серьезности.

Назначение: Назначение дефекта разработчику или команде для анализа и исправления.

Анализ и исправление: Разработчик анализирует причину дефекта и вносит изменения в код для его исправления.

Тестирование исправления: Тестировщик проверяет исправление и повторно тестирует систему, чтобы убедиться, что дефект устранен и не повлиял на другие функции (ретестирование и регрессионное тестирование).

Закрытие: Если дефект устранен, он закрывается в системе отслеживания дефектов. Если он не устранен, процесс возвращается к этапу "Анализ и исправление".

Анализ дефектов: Анализируется история дефекта для выявления причин и предотвращения подобных проблем в будущем (например, анализ первопричин).

**Документирование дефектов**

Идентификатор дефекта: Уникальный номер или код для каждого дефекта.

Название дефекта: Краткое описание проблемы.

Описание: Подробное описание проблемы, включая шаги для воспроизведения, ожидаемые и фактические результаты.

Шаги для воспроизведения: Подробная инструкция для воспроизведения дефекта.

Окружение: Информация о программной и аппаратной среде, в которой был обнаружен дефект.

Приоритет: Оценка важности и срочности исправления дефекта.

Серьезность: Оценка влияния дефекта на систему.

Ответственный: Имя разработчика или команды, назначенной для исправления дефекта.

Статус: Текущий статус дефекта (открыт, в работе, исправлен, закрыт).

Комментарии: Дополнительные заметки и комментарии по поводу дефекта и его исправления.

**Примеры систем отслеживания дефектов**

JIRA: Популярный инструмент для управления проектами и отслеживания дефектов, предоставляющий гибкие возможности настройки и интеграции.

Bugzilla: Открытый инструмент для отслеживания дефектов, используемый многими крупными проектами с открытым исходным кодом.

Redmine: Инструмент управления проектами с функциями отслеживания дефектов и поддержкой множества плагинов.

Trello: Инструмент для управления задачами, который также может использоваться для отслеживания дефектов с помощью досок и карточек.

**Билет №21**

**1. Ошибки, дефекты, сбои, отказы.**

**2. Разработать модульные тесты для мобильного приложения Calculate, выполнить тестирование функции performOperation. Приложение находится «Resorce/Calc/». Оформить тест-кейсы для модульного тестирования.**

Ошибка — это действие или решение, которое приводит к неправильному результату. Ошибки могут быть допущены разработчиками, тестировщиками или пользователями.

Дефект — это несоответствие между ожидаемым результатом и фактическим поведением системы. Дефект может быть вызван ошибкой в коде, неправильной конфигурацией системы или другими факторами.

Сбой — это временное нарушение работы системы, которое может привести к неправильному результату или поведению. Сбои могут быть вызваны внешними факторами, такими как отключение электроэнергии или ошибка пользователя. Сбои обычно восстанавливаются автоматически или вручную.

Отказ — это постоянное нарушение работы системы, которое делает её непригодной для использования. Отказы могут быть вызваны серьёзными дефектами или ошибками, которые не были обнаружены и исправлены. Отказы требуют серьёзного вмешательства для восстановления работоспособности системы.