**Экзаменационные вопросы по МДК 05.03 Тестирование информационных систем**

**1. Понятие тестирования, этапы процесса тестирования. Цикл тестирования, этапы прохода цикла тестирования.**

Тестирование представляет собой деятельность по проверке программного кода и документации. Она должна заранее планироваться и систематически проводиться специально назначенным сотрудником – тестировщиком. Работа тестировщика начинается до утверждения требований к программному продукту.

Этапы тестирования:

1. Проверка требований к ПП на полноту.
2. Определение методов тестирования.
3. Разработка стратегии тестирования.
4. Разработка плана тестирования.
5. Создание наборов тестов для проведения интеграционного и системного тестирований.

Цикл тестирования – совокупность действий, выполняемых тестировщиком с момента передачи базовой версии ПП тестировщику для интеграционного, системного или приемного тестирования до момента успешного завершения тестирования.

Каждый проход цикла тестирования состоит из следующих этапов:

1. Создание базовой версии ПП.
2. Проведение тестирования.
3. Анализ результатов.
4. Создание отчета о тестировании.
5. Если в процессе тестирования выявлены ошибки, то выполняется их исправление и переход к первому этапу.
6. Если в процессе тестирования ошибок не выявлено, то цикл тестирования завершается.

**2. Виды тестирования в зависимости от объекта тестирования: функциональное тестирование, тестирование безопасности, тестирование защищенности.**

Функциональное тестирование – это тип тестирования, направленный на проверку соответствия работы программного обеспечения его функциональным требованиям и спецификациям.

Цель: Убедиться, что все заявленные функции работают корректно и соответствуют требованиям.

Основные виды:

Тестирование черного ящика: Проверка функционала без знания внутренней структуры кода.

Модульное тестирование: Проверка отдельных модулей или компонентов.

Интеграционное тестирование: Проверка взаимодействия между модулями.

Системное тестирование: Проверка всей системы в целом.

Приемочное тестирование: Проверка системы на соответствие требованиям заказчика.

Тестирование безопасности – это процесс проверки программного обеспечения на наличие уязвимостей и обеспечение защиты данных от различных угроз.

Цель: Обеспечить защиту системы и данных от несанкционированного доступа, атак и других угроз.

Основные виды:

Тестирование аутентификации: Проверка корректности процедур входа в систему.

Тестирование авторизации: Проверка правильности распределения прав доступа.

Тестирование на уязвимости: Поиск и устранение уязвимостей, таких как SQL-инъекции, XSS и др.

Тестирование безопасности данных: Проверка защиты данных при передаче и хранении.

Пенетестация: Имитирование атак на систему для выявления уязвимостей.

Тестирование защищенности – это процесс проверки уровня защиты информационных систем и данных от внутренних и внешних угроз.

Цель: Обеспечить высокий уровень защищенности системы и данных от потенциальных угроз.

Основные виды:

Анализ рисков: Оценка возможных угроз и их последствий.

Тестирование на устойчивость к атакам: Проверка способности системы противостоять различным видам атак.

Тестирование резервного копирования и восстановления: Проверка эффективности механизмов резервного копирования и восстановления данных.

Контроль соответствия стандартам безопасности: Проверка соответствия системы международным и отраслевым стандартам безопасности.

Тестирование управления инцидентами: Проверка готовности системы к реагированию на инциденты безопасности.

**3. Виды тестирования в зависимости от объекта тестирования: совместимости, кроссбраузерное тестирование, кроссплатформенное тестирование. Нефункциональное тестирование: тестирование требований, тестирование прототипа.**

Тестирование совместимости – это процесс проверки программного обеспечения на его работоспособность в различных условиях и средах.

Цель: Убедиться, что ПО корректно функционирует в различных окружениях.

Основные виды:

Тестирование на совместимость с аппаратным обеспечением: Проверка работы ПО на различных устройствах и конфигурациях оборудования.

Тестирование на совместимость с операционными системами: Проверка работы ПО на различных версиях и типах операционных систем.

Тестирование на совместимость с другими приложениями: Проверка работы ПО при взаимодействии с другими программами.

Кроссбраузерное тестирование – это процесс проверки веб-приложений на корректность отображения и функционирования в различных веб-браузерах.

Цель: Убедиться, что веб-приложение выглядит и работает одинаково хорошо во всех поддерживаемых браузерах.

Основные аспекты:

Отображение интерфейса: Проверка правильности отображения элементов интерфейса.

Функциональность: Проверка корректности работы всех функций и сценариев в разных браузерах.

Производительность: Оценка времени загрузки и отклика веб-приложения в разных браузерах.

Кроссплатформенное тестирование – это процесс проверки программного обеспечения на его работоспособность на различных платформах и устройствах.

Цель: Убедиться, что ПО работает корректно на всех поддерживаемых платформах.

Основные аспекты:

Проверка совместимости с различными операционными системами: Windows, macOS, Linux, iOS, Android и т.д.

Проверка работы на различных устройствах: ПК, смартфоны, планшеты и т.д.

Проверка адаптивности интерфейса: Убедиться, что интерфейс корректно адаптируется под разные размеры экранов и разрешения.

Тестирование требований – это процесс проверки полноты, четкости, однозначности и выполнимости требований к программному обеспечению.

Цель: Убедиться, что требования к ПО корректны, полны и не противоречат друг другу.

Основные аспекты:

Анализ требований: Проверка логической согласованности и полноты требований.

Проверка на выполнимость: Оценка реальности и достижимости заявленных требований.

Валидация требований: Убедиться, что требования соответствуют ожиданиям и потребностям заказчика.

Тестирование прототипа – это процесс проверки ранней версии продукта или его отдельных компонентов для выявления и исправления недостатков на ранних стадиях разработки.

Цель: Оценить основные концепции и функциональность, выявить возможные проблемы и получить раннюю обратную связь.

Основные аспекты:

Оценка пользовательского интерфейса: Проверка удобства использования и навигации.

Проверка основных функций: Убедиться, что ключевые функции реализованы и работают корректно.

Сбор обратной связи: Получение мнений пользователей и заинтересованных сторон для внесения корректировок.

**4. Виды тестирования в зависимости от объекта тестирования: тестирование пользовательского интерфейса, тестирование удобства использования, тестирование доступности, интернационализации.**

Тестирование пользовательского интерфейса – это процесс проверки визуальных элементов и взаимодействий на экране для обеспечения их корректной работы и соответствия требованиям дизайна.

Цель: Убедиться, что все элементы интерфейса правильно отображаются и работают согласно спецификациям.

Основные аспекты:

Визуальная проверка: Оценка соответствия дизайну (цвета, шрифты, размеры элементов).

Функциональная проверка: Проверка работы интерактивных элементов (кнопки, поля ввода, меню и т.д.).

Проверка навигации: Убедиться, что навигационные элементы (ссылки, кнопки переходов) работают корректно.

Тестирование удобства использования – это процесс оценки продукта с точки зрения удобства и эффективности его использования конечными пользователями.

Цель: Обеспечить, чтобы продукт был интуитивно понятным, удобным и эффективным для пользователей.

Основные аспекты:

Легкость изучения: Оценка времени, необходимого для освоения работы с продуктом.

Эффективность использования: Проверка, насколько быстро и точно пользователи могут выполнять задачи.

Удовлетворенность пользователя: Сбор обратной связи от пользователей о степени удовлетворенности продуктом.

Ошибки и их частота: Определение частоты ошибок, которые совершают пользователи, и их серьезность.

Тестирование доступности – это процесс проверки программного обеспечения на соответствие требованиям доступности для людей с ограниченными возможностями.

Цель: Обеспечить, чтобы продукт был доступен и удобен для использования всеми категориями пользователей, включая людей с ограниченными возможностями.

Основные аспекты:

Поддержка вспомогательных технологий: Проверка совместимости с программами для чтения с экрана, увеличителями и другими вспомогательными технологиями.

Использование клавиатуры: Убедиться, что все функции доступны через клавиатуру.

Цветовая контрастность: Проверка контрастности текста и фона для людей с нарушениями зрения.

Аудио- и видеоконтент: Обеспечение наличия субтитров и текстовых описаний для мультимедийных элементов.

Тестирование интернационализации – это процесс проверки программного обеспечения на способность адаптироваться к различным языкам и региональным особенностям без необходимости изменения кода.

Цель: Убедиться, что продукт может быть легко адаптирован для разных локалей.

Основные аспекты:

Проверка локализации: Убедиться, что интерфейс и сообщения правильно переведены и отображаются на разных языках.

Форматы данных: Проверка корректности отображения форматов дат, времени, чисел и валют для разных регионов.

Символы и шрифты: Убедиться, что все символы и шрифты корректно отображаются для разных языков.

Тестирование ввода: Проверка корректности ввода и обработки данных на различных языках и раскладках клавиатуры.

**5. Виды тестирования в зависимости от объекта тестирования: тестирование локализации, установки, производительности (нагрузочное, объемное, стрессовое, тестирование на отказ и восстановление).**

Тестирование локализации – это процесс проверки программного обеспечения на его соответствие культурным, языковым и региональным особенностям конкретного рынка.

Цель: Убедиться, что продукт адаптирован для использования в различных локалях и корректно отображает все переводы и региональные настройки.

Основные аспекты:

Проверка перевода: Убедиться, что все строки текста правильно переведены и отображаются без ошибок.

Форматы данных: Проверка корректности форматов дат, времени, чисел и валют для конкретной локали.

Интерфейс: Проверка отображения интерфейса, включая длину текстов, размеры элементов и отсутствие обрезанных строк.

Культурные особенности: Убедиться, что контент и символика соответствуют культурным и региональным стандартам.

Тестирование установки – это процесс проверки процесса установки, обновления и удаления программного обеспечения.

Цель: Обеспечить, что установка, обновление и удаление ПО проходят без ошибок и не вызывают проблем.

Основные аспекты:

Установка: Проверка правильности установки программного обеспечения на целевые платформы.

Обновление: Проверка корректности обновления ранее установленных версий программного обеспечения.

Удаление: Убедиться, что ПО корректно удаляется, не оставляя ненужных файлов и записей в системе.

Восстановление: Проверка возможности восстановления ПО после неполной установки или удаления.

Тестирование производительности – это процесс оценки скорости, стабильности и масштабируемости программного обеспечения под различными нагрузками.

Нагрузочное тестирование

Цель: Оценить производительность системы под ожидаемой нагрузкой.

Основные аспекты:

Проверка времени отклика: Измерение времени отклика системы при нормальной и увеличенной нагрузке.

Производительность: Оценка способности системы справляться с заданными объемами запросов и транзакций.

Объемное тестирование

Цель: Оценить поведение системы при обработке больших объемов данных.

Основные аспекты:

Работа с большими объемами данных: Проверка корректности обработки и хранения больших объемов данных.

Производительность: Оценка времени выполнения операций с большими объемами данных.

Стрессовое тестирование

Цель: Оценить поведение системы в условиях предельной нагрузки и выявить ее пределы.

Основные аспекты:

Пиковая нагрузка: Проверка работы системы при нагрузках, превышающих расчетные.

Стабильность: Оценка устойчивости и времени восстановления системы после критических нагрузок.

Тестирование на отказ и восстановление

Цель: Оценить способность системы восстанавливаться после сбоев и продолжать работу.

Основные аспекты:

Проверка отказоустойчивости: Оценка способности системы продолжать работу в условиях частичных отказов компонентов.

Восстановление: Проверка времени и корректности восстановления системы после сбоев.

**6. Виды тестирования в зависимости от знания кода: белого ящика, серого ящика, черного ящика.**

Тестирование белого ящика – это метод тестирования, при котором тестировщик имеет доступ к исходному коду и внутренней структуре системы.

Цель: Проверить внутреннюю логику, структуру и код программы на соответствие спецификациям и выявить дефекты.

Основные аспекты:

Покрытие кода: Проверка покрытия кода тестами (покрытие операторов, ветвей, условий и путей).

Анализ потоков данных: Проверка корректности потоков данных внутри программы.

Тестирование циклов: Проверка выполнения циклов и их пределов.

Проверка корректности алгоритмов: Убедиться, что реализованные алгоритмы работают правильно.

Преимущества:

Высокая точность в обнаружении дефектов.

Полное покрытие всех логических путей программы.

Возможность оптимизации кода.

Недостатки:

Требует глубокого понимания внутренней структуры и кода.

Может быть трудоемким и затратным по времени.

Тестирование серого ящика – это метод тестирования, при котором тестировщик имеет ограниченные знания о внутренней структуре системы и доступ к некоторой документации или архитектуре кода.

Цель: Использовать частичное знание о внутренней структуре для более эффективного тестирования.

Основные аспекты:

Тестирование интеграции: Проверка взаимодействия между модулями с использованием информации о их интерфейсах.

Тестирование безопасности: Проверка уязвимостей и потенциальных точек отказа на основе архитектуры системы.

Анализ тестовых сценариев: Создание тестов на основе анализа архитектуры и структуры данных.

Преимущества:

Баланс между глубиной проверки и затратами на тестирование.

Повышенная эффективность в обнаружении дефектов на уровне интеграции.

Использование знаний о системе для создания более точных тестов.

Недостатки:

Ограниченные возможности по сравнению с тестированием белого ящика.

Меньшая детализация проверки внутренней логики.

Тестирование черного ящика – это метод тестирования, при котором тестировщик не имеет доступа к исходному коду и внутренней структуре системы и проверяет систему только с точки зрения внешнего поведения.

Цель: Проверить соответствие функциональных требований и спецификаций без учета внутренней структуры.

Основные аспекты:

Функциональное тестирование: Проверка корректности выполнения всех функций системы.

Тестирование на основе спецификаций: Создание тестов на основе описанных требований и пользовательских сценариев.

Проверка пользовательского интерфейса: Оценка корректности взаимодействия пользователя с системой.

Преимущества:

Не требует знания внутренней структуры системы.

Независимость от языка программирования и архитектуры кода.

Подходит для тестирования с точки зрения конечного пользователя.

Недостатки:

Ограниченные возможности по выявлению дефектов в коде и внутренней логике.

Возможность пропустить дефекты, не связанные с внешним поведением.

**7. Виды тестирования в зависимости от степени автоматизации: ручное, автоматизированное тестирование.**

Ручное тестирование – это процесс проверки программного обеспечения вручную, без использования автоматизированных инструментов и скриптов.

Цель: Убедиться, что программное обеспечение работает в соответствии с требованиями, путем выполнения тестовых сценариев вручную.

Основные аспекты:

Создание тест-кейсов: Разработка подробных тестовых сценариев на основе требований и спецификаций.

Выполнение тестов: Тестировщик вручную выполняет тестовые сценарии и фиксирует результаты.

Анализ результатов: Сравнение фактических результатов с ожидаемыми и документирование выявленных дефектов.

Ретестирование: Проверка исправлений после внесения изменений в ПО.

Преимущества:

Гибкость в тестировании различных аспектов и пользовательских сценариев.

Легкость в адаптации тестов под изменяющиеся требования.

Возможность проведения тестирования на основе интуиции и опыта тестировщика.

Эффективно для тестирования UI и UX.

Недостатки:

Трудоемкость и затраты времени на выполнение тестов.

Возможность человеческих ошибок.

Неэффективность для выполнения повторяющихся и регрессионных тестов.

Ограниченная масштабируемость.

Автоматизированное тестирование – это процесс проверки программного обеспечения с использованием специальных инструментов и скриптов для автоматического выполнения тестов.

Цель: Повысить эффективность и точность тестирования за счет автоматизации повторяющихся и трудоемких тестов.

Основные аспекты:

Разработка тест-скриптов: Создание автоматизированных тестов с использованием языков программирования и специальных фреймворков.

Настройка среды: Подготовка тестовой среды и конфигурация инструментов для автоматизированного тестирования.

Выполнение тестов: Автоматическое выполнение тест-скриптов и сбор результатов.

Анализ результатов: Автоматический анализ и генерация отчетов о результатах тестирования.

Регрессионное тестирование: Частое использование автоматизированных тестов для проверки регрессий после изменений в коде.

Преимущества:

Высокая скорость выполнения тестов.

Повышенная точность и повторяемость результатов.

Возможность проведения тестирования 24/7 без участия человека.

Эффективность для регрессионного тестирования и тестирования производительности.

Легкость масштабирования и покрытия большого числа сценариев.

Недостатки:

Высокие первоначальные затраты на разработку и настройку тестов.

Требуется техническая экспертиза для создания и поддержания тест-скриптов.

Ограниченная гибкость в адаптации под новые требования без изменения скриптов.

Неэффективность для тестирования UI и UX аспектов, требующих человеческого восприятия.

**8. Виды тестирования в зависимости от степени изолированности тестируемых компонентов: модульное, интергационное, системное тестирование.**

Модульное тестирование – это процесс проверки отдельных компонентов или модулей программного обеспечения в изоляции от других частей системы.

Цель: Убедиться, что каждый отдельный модуль работает правильно согласно его спецификации.

Основные аспекты:

Изолированность: Тестирование проводится на уровне отдельного модуля, без взаимодействия с другими модулями.

Мокирование: Использование заглушек и мок-объектов для эмуляции поведения зависимых модулей.

Мелкие тесты: Создание мелких, четко определенных тестов для проверки отдельных функций и методов.

Автоматизация: Модульные тесты часто автоматизируются для обеспечения быстрой проверки изменений в коде.

Преимущества:

Раннее выявление дефектов.

Легкость в локализации и исправлении ошибок.

Поддержка рефакторинга кода.

Высокая точность тестирования отдельных компонентов.

Недостатки:

Ограниченная проверка взаимодействия между модулями.

Требует написания большого количества тестов.

Не выявляет интеграционные и системные проблемы.

Интеграционное тестирование – это процесс проверки взаимодействия между модулями или компонентами программного обеспечения.

Цель: Убедиться, что отдельные модули корректно взаимодействуют друг с другом и совместно выполняют требуемые функции.

Основные аспекты:

Тестирование интерфейсов: Проверка правильности взаимодействия через интерфейсы между модулями.

Инкрементальное тестирование: Постепенная интеграция и тестирование модулей по мере их готовности.

Системные интеграционные тесты: Проверка взаимодействия значимых подсистем или сервисов.

Использование моков и заглушек: Эмуляция поведения некоторых зависимостей для тестирования интеграции.

Преимущества:

Выявление проблем взаимодействия между модулями.

Проверка совместимости различных компонентов.

Обнаружение ошибок в интерфейсах и протоколах обмена данными.

Недостатки:

Сложность в определении источника ошибки.

Требует дополнительных ресурсов для настройки тестовой среды.

Не охватывает системные и эксплуатационные аспекты полностью.

Системное тестирование – это процесс проверки всей системы в целом, включая все модули и компоненты, для обеспечения соответствия функциональным и нефункциональным требованиям.

Цель: Убедиться, что вся система работает правильно и удовлетворяет требованиям заказчика.

Основные аспекты:

Функциональное тестирование: Проверка всех функциональных требований системы.

Нефункциональное тестирование: Оценка производительности, безопасности, совместимости и других нефункциональных аспектов.

Энд-ту-энд тестирование: Проверка системы от начала до конца, включая все процессы и потоки данных.

Реальные условия: Тестирование в условиях, максимально приближенных к реальной эксплуатации.

Преимущества:

Комплексная проверка всей системы.

Выявление дефектов, связанных с интеграцией и эксплуатацией.

Проверка соответствия системы требованиям заказчика.

Недостатки:

Высокая стоимость и длительность выполнения.

Требует полного развертывания системы.

Сложность в изоляции и исправлении обнаруженных дефектов.

**9. Виды тестирования в зависимости от подготовленности: интуитивное тестирование, исследовательское тестирование, тестирование по документации.**

Интуитивное тестирование – это неформальный метод тестирования, который проводится без предварительно разработанных тестовых сценариев и документации. Оно основано на интуиции, опыте и знаниях тестировщика.

Цель: Быстро обнаружить очевидные дефекты и ошибки в системе.

Основные аспекты:

Отсутствие структуры: Тестирование проводится спонтанно, без предварительной подготовки.

Использование опыта: Тестировщик полагается на свой опыт и понимание системы.

Гибкость: Возможность быстро переключаться между различными частями системы и методами тестирования.

Преимущества:

Быстрое выявление критических и очевидных ошибок.

Гибкость и возможность адаптации к изменениям в системе.

Не требует затрат на подготовку тестовой документации.

Недостатки:

Неполное покрытие функциональности.

Зависимость от навыков и опыта тестировщика.

Сложность в повторении тестов и документировании результатов.

Исследовательское тестирование – это метод тестирования, который сочетает в себе процесс тестирования с процессом изучения и обучения системе. Тестировщик активно изучает систему, разрабатывает и выполняет тесты на ходу.

Цель: Обнаружить дефекты и проблемы в системе путем активного изучения и анализа ее поведения.

Основные аспекты:

Активное изучение: Тестировщик исследует систему, чтобы понять ее поведение и выявить потенциальные дефекты.

Адаптация: Создание и выполнение тестов на основе полученной информации.

Документирование: Ведение записей о проведенных тестах и обнаруженных дефектах.

Преимущества:

Глубокое понимание системы и ее поведения.

Высокая вероятность выявления скрытых и сложных дефектов.

Возможность адаптации и изменения подхода в процессе тестирования.

Недостатки:

Требует высококвалифицированных тестировщиков.

Сложность в повторении тестов и стандартной отчетности.

Может быть трудоемким и затратным по времени.

Тестирование по документации – это метод тестирования, при котором тестировщик следует заранее разработанным тестовым сценариям и планам, описанным в документации.

Цель: Обеспечить систематическое и полное покрытие всех функциональных и нефункциональных требований системы.

Основные аспекты:

Тестовые сценарии: Использование детально разработанных тест-кейсов, основанных на требованиях и спецификациях.

План тестирования: Следование заранее разработанному плану тестирования.

Документирование: Тщательное документирование выполнения тестов и результатов.

Преимущества:

Систематичность и полнота тестирования.

Легкость в повторении тестов и верификации результатов.

Обеспечение соответствия требованиям и спецификациям.

Недостатки:

Меньшая гибкость в адаптации к изменениям.

Возможность пропуска незапланированных дефектов.

Затраты на разработку и поддержку тестовой документации.

**10. Виды тестирования в зависимости от места и времени проведения тестирования: приемочное тестирование, альфа-тестирование, бета- тестирование.**

Приемочное тестирование – это процесс проверки программного обеспечения на соответствие требованиям и критериям приемки заказчика перед его внедрением в эксплуатацию.

Цель: Убедиться, что программное обеспечение соответствует всем требованиям и может быть принято заказчиком.

Основные аспекты:

Тестирование по требованиям: Проверка соответствия функциональным и нефункциональным требованиям.

Участие заказчика: Заказчик или представители пользователей активно участвуют в тестировании.

Реальные сценарии: Тестирование проводится в условиях, максимально приближенных к реальным условиям эксплуатации.

Критерии приемки: Использование заранее определенных критериев для оценки готовности системы.

Преимущества:

Обеспечение соответствия требованиям заказчика.

Подтверждение готовности системы к эксплуатации.

Вовлечение конечных пользователей в процесс тестирования.

Недостатки:

Возможны задержки, если требования не полностью соответствуют ожиданиям.

Зависимость от наличия четких и полных требований.

Альфа-тестирование – это внутренняя проверка программного обеспечения, проводимая на стадии завершения разработки, но перед выпуском его на внешнее тестирование.

Цель: Обнаружить и исправить максимальное количество дефектов перед выпуском продукта на бета-тестирование или в эксплуатацию.

Основные аспекты:

Проведение внутри компании: Тестирование проводится разработчиками и внутренними тестировщиками.

Ограниченный доступ: Участие ограниченного числа пользователей, часто включающих сотрудников компании.

Ранние версии ПО: Проверка ранних версий ПО, возможно с ограниченным функционалом.

Исправление дефектов: Активное выявление и исправление дефектов до выхода на бета-тестирование.

Преимущества:

Раннее выявление и исправление ошибок.

Обеспечение базовой стабильности системы перед внешним тестированием.

Улучшение качества перед выпуском на бета-тестирование.

Недостатки:

Ограниченное количество тестировщиков.

Может не выявить все проблемы, встречающиеся в реальных условиях эксплуатации.

Бета-тестирование – это внешняя проверка программного обеспечения, проводимая на стадии завершения разработки с участием реальных пользователей.

Цель: Получить обратную связь от реальных пользователей и выявить дефекты, которые не были обнаружены на предыдущих стадиях тестирования.

Основные аспекты:

Внешние пользователи: Участие реальных пользователей, которые не связаны с разработкой ПО.

Реальные условия эксплуатации: Тестирование проводится в реальных условиях, приближенных к эксплуатации.

Обратная связь: Сбор отзывов и предложений от пользователей для улучшения качества ПО.

Широкое покрытие: Охват различных сценариев использования и конфигураций систем.

Преимущества:

Получение реальной обратной связи от конечных пользователей.

Выявление дефектов и проблем, не обнаруженных в альфа-тестировании.

Улучшение качества и удовлетворенности пользователей.

Недостатки:

Возможность выявления критических ошибок на поздней стадии.

Требует координации и поддержки пользователей.

**11. Виды тестирования в зависимости от глубины тестового покрытия: Smoke, MAT, AT.**

Виды тестирования в зависимости от глубины тестового покрытия:

Smoke Testing:

Цель: Проверка базовой работоспособности системы.

Описание: Выполняется для быстрого подтверждения, что основная функциональность системы работает. Проверяются самые критичные пути и компоненты.

Пример: Запуск основного функционала приложения, такого как авторизация и главная страница.

Minimal Acceptance Testing (MAT):

Цель: Проверка минимального набора требований.

Описание: Проверяется, соответствует ли система основным критериям приемлемости, чтобы продолжить более детальное тестирование.

Пример: Проверка, что все ключевые функции и бизнес-процессы работают корректно согласно спецификациям.

Acceptance Testing (AT):

Цель: Подтверждение, что система удовлетворяет требованиям и готова к эксплуатации.

Описание: Проводится конечными пользователями или клиентами для оценки системы в реальных условиях. Это наиболее глубокое и всеобъемлющее тестирование.

Пример: Тестирование полного цикла бизнес-процессов и сценариев использования, имитирующих реальную работу пользователей с системой.

**12. Виды тестирования в зависимости от тестовых активностей: тестирование новых функциональностей, регрессионное тестирование, валидация дефектов.**

Виды тестирования в зависимости от тестовых активностей:

Тестирование новых функциональностей:

Цель: Проверка новых функций и модулей системы.

Описание: Проводится для обеспечения корректной работы недавно разработанных или изменённых функций. Включает тестирование согласно спецификациям и требованиям.

Пример: После добавления нового модуля отчётности, тестирование его функциональности, пользовательского интерфейса и интеграции с другими частями системы.

Регрессионное тестирование:

Цель: Обнаружение ошибок, введённых в систему после изменений.

Описание: Повторное тестирование ранее протестированных частей системы для подтверждения, что изменения (исправления багов, добавление новых функций) не нарушили существующую функциональность.

Пример: После исправления дефекта в платёжной системе, повторное тестирование всего процесса оплаты для проверки, что не возникли новые ошибки.

Валидация дефектов:

Цель: Подтверждение исправления ранее обнаруженных дефектов.

Описание: Проводится после исправления дефектов для проверки, что проблемы действительно устранены и система работает корректно в этих местах.

Пример: После фиксации ошибки в модуле авторизации, проверка работы всех сценариев входа в систему, чтобы убедиться в отсутствии первоначальной ошибки.

**13. Понятие требования, значение требований. Этапы работы над требованиями.**

Требование — это документированное представление того, что система должна делать или как она должна работать. Требования определяют функциональные и нефункциональные аспекты системы и служат основой для её разработки и тестирования.

Значение требований

Основа для разработки: Требования предоставляют чёткое описание задач и функций, которые система должна выполнять, что направляет процесс разработки.

Критерии для тестирования: Определение успешного выполнения требований позволяет тестировщикам проверять корректность системы.

Коммуникация с заинтересованными сторонами: Требования служат средством общения между клиентами, разработчиками и тестировщиками, помогая обеспечить понимание и согласие всех сторон.

Управление изменениями: Позволяют управлять изменениями в проекте и их влиянием на систему.

Этапы работы над требованиями

Сбор требований:

Выявление потребностей и ожиданий пользователей и других заинтересованных сторон.

Методы: интервью, опросы, наблюдения, анализ существующей документации.

Анализ требований:

Оценка собранных требований на полноту, согласованность, выполнимость и тестируемость.

Определение приоритетов и устранение противоречий.

Документирование требований:

Формализация требований в виде спецификаций.

Использование стандартов и шаблонов для однозначности и полноты описания.

Проверка и утверждение требований:

Проведение ревью и согласование требований с заинтересованными сторонами.

Утверждение финального набора требований, которые будут использованы для разработки и тестирования.

Управление требованиями:

Отслеживание изменений требований и их влияния на проект.

Поддержание актуальности требований на всех этапах жизненного цикла проекта.

Валидация требований:

Подтверждение, что требования правильно отражают потребности пользователей и могут быть реализованы в системе.

**14. Уровни требований: уровень бизнес-требований, уровень пользовательских требований, уровень продуктных требований (функциональные и нефункциональные требования).**

Уровень бизнес-требований

Описание: Высокоуровневые цели и задачи организации, которые система должна поддерживать.

Цель: Определение стратегических целей и выгод для бизнеса от внедрения системы.

Пример: Увеличение доли рынка на 10%, улучшение клиентского обслуживания, сокращение времени обработки заказов.

Уровень пользовательских требований

Описание: Подробное описание того, что пользователи ожидают от системы, как они будут взаимодействовать с ней.

Цель: Удовлетворение потребностей и ожиданий конечных пользователей системы.

Пример: Возможность пользователю создавать и управлять профилем, доступ к истории заказов, удобный интерфейс для поиска продуктов.

Уровень продуктных требований

Описание: Конкретные характеристики и функции, которые должна иметь система для выполнения бизнес- и пользовательских требований.

Цель: Детальное определение того, как система должна работать.

Функциональные требования

Описание: Описывают конкретные функции и действия, которые должна выполнять система.

Пример: Система должна позволять пользователям регистрироваться и входить в систему, обрабатывать платежи, генерировать отчёты.

Нефункциональные требования

Описание: Описывают характеристики системы, такие как производительность, надежность, безопасность, удобство использования.

Пример: Система должна поддерживать до 10,000 одновременных пользователей, время отклика должно быть менее 2 секунд, данные пользователей должны быть зашифрованы.

**15. Вариант использования (Use Case): понятие, назначение, элементы, сценарии.**

Понятие: Вариант использования — это описание взаимодействия между пользователем (или другой системой) и системой для выполнения конкретной задачи. Это один из методов моделирования требований к системе, который помогает определить, как система будет использоваться.

Назначение:

Определение функциональных требований к системе.

Обеспечение понимания всех возможных сценариев использования системы.

Служит основой для разработки тестовых сценариев и системной документации.

Улучшение коммуникации между аналитиками, разработчиками и пользователями.

Элементы варианта использования

Акторы:

Описание: Внешние сущности, которые взаимодействуют с системой. Актором может быть человек, другая система или внешний процесс.

Пример: Пользователь, администратор, внешняя платёжная система.

Цель:

Описание: Конкретный результат, который должен быть достигнут в результате выполнения варианта использования.

Пример: Пользователь успешно завершил процесс покупки.

Предусловия:

Описание: Условия, которые должны быть выполнены перед началом сценария.

Пример: Пользователь зарегистрирован и вошел в систему.

Сценарий:

Основной сценарий:

Описание: Последовательность шагов, ведущая к достижению цели при нормальных условиях.

Пример: Пользователь добавляет товар в корзину, переходит к оформлению заказа, вводит платёжные данные, подтверждает покупку.

Альтернативные сценарии:

Описание: Дополнительные или альтернативные пути выполнения сценария.

Пример: Пользователь изменяет количество товаров в корзине, выбирает другой способ оплаты.

Исключительные ситуации:

Описание: Описание шагов в случае возникновения ошибок или исключений.

Пример: Платёж не прошел из-за недостатка средств, пользователю показывается сообщение об ошибке и предлагается ввести другие платёжные данные.

Постусловия:

Описание: Условия, которые должны быть выполнены после завершения сценария.

Пример: Заказ успешно создан и отправлен на обработку.

Сценарии

Основной сценарий:

Пошаговая инструкция по достижению цели варианта использования при нормальном ходе событий.

Альтернативные сценарии:

Описание возможных отклонений от основного сценария, которые приводят к достижению той же цели другим путем.

Исключительные сценарии:

Описание шагов в случае возникновения ошибок или исключений, которые препятствуют достижению основной цели.

**16. Требования к качеству функционирования сложных программных комплексов, требования к характеристикам качества сложных программных комплексов.**

Требования к качеству функционирования сложных программных комплексов

Качество функционирования сложных программных комплексов определяется набором характеристик, которые обеспечивают их надежность, производительность, безопасность и удобство использования. Основные требования к качеству включают:

Надежность:

Цель: Обеспечение безотказной работы системы в течение заданного времени.

Ключевые показатели:

Доступность: Система должна быть доступна для использования в течение максимального времени.

Устойчивость к отказам: Способность системы продолжать работу в случае отказа одного или нескольких компонентов.

Восстанавливаемость: Способность системы быстро восстановиться после сбоя.

Производительность:

Цель: Обеспечение своевременной обработки данных и отклика системы.

Ключевые показатели:

Время отклика: Время, необходимое для выполнения определённого действия.

Пропускная способность: Количество операций, выполняемых системой за единицу времени.

Масштабируемость: Способность системы увеличивать производительность при увеличении нагрузки.

Безопасность:

Цель: Защита данных и функций системы от несанкционированного доступа и атак.

Ключевые показатели:

Конфиденциальность: Защита данных от несанкционированного доступа.

Целостность: Обеспечение точности и полноты данных.

Доступность: Обеспечение доступности данных и функций для авторизованных пользователей.

Удобство использования:

Цель: Обеспечение легкости и удобства взаимодействия пользователей с системой.

Ключевые показатели:

Узнаваемость: Легкость освоения системы новыми пользователями.

Эффективность: Способность пользователей выполнять задачи быстро и без усилий.

Удовлетворенность: Уровень удовлетворенности пользователей при взаимодействии с системой.

Поддерживаемость:

Цель: Обеспечение возможности легко изменять и обновлять систему.

Ключевые показатели:

Модульность: Структурная организация системы, позволяющая легко вносить изменения.

Понятность: Легкость понимания и использования кода и документации.

Тестируемость: Способность системы поддерживать процесс тестирования.

Требования к характеристикам качества сложных программных комплексов

Функциональность:

Система должна выполнять все заявленные функции корректно и полно.

Соответствие требованиям и спецификациям.

Взаимодействие с другими системами и компонентами.

Эффективность:

Оптимальное использование ресурсов (память, процессорное время).

Минимизация затрат на выполнение операций.

Совместимость:

Способность системы работать в различных средах (аппаратных, программных, сетевых).

Поддержка стандартов и протоколов.

Переносимость:

Способность системы работать на разных платформах и в различных средах.

Легкость адаптации к новым условиям.

Адаптируемость:

Способность системы адаптироваться к изменениям в требованиях и окружающей среде.

Гибкость и расширяемость функциональности.

**17. Требования к эффективности использования ресурсов ЭВМ программным комплексом в реальном времени. Проверка корректности функциональных требований к сложным комплексам программ.**

Требования к эффективности использования ресурсов ЭВМ программным комплексом в реальном времени

Для программных комплексов, работающих в реальном времени, важно эффективно использовать ресурсы ЭВМ, чтобы обеспечить своевременное выполнение задач. Основные требования включают:

Производительность:

Время отклика: Система должна быстро реагировать на запросы и события, обеспечивая минимальные задержки.

Пропускная способность: Система должна обрабатывать максимальное количество операций в единицу времени.

Оптимизация использования ресурсов:

Процессорное время: Система должна эффективно использовать процессорное время, избегая перегрузок.

Память: Оптимизация использования оперативной памяти для предотвращения утечек памяти и обеспечении её достаточного объёма для всех процессов.

Дисковое пространство: Эффективное управление дисковым пространством для хранения данных и журналов.

Реализация приоритетов:

Обработка приоритетных задач: Система должна уметь обрабатывать задачи с разными приоритетами, обеспечивая своевременное выполнение критичных задач.

Непрерывность работы:

Устойчивость к отказам: Система должна продолжать функционировать при отказах отдельных компонентов.

Восстанавливаемость: Способность быстро восстанавливаться после сбоев.

Энергопотребление:

Система должна быть энергоэффективной, особенно в случае встроенных или мобильных приложений, чтобы продлить срок службы батарей и снизить эксплуатационные расходы.

Проверка корректности функциональных требований к сложным комплексам программ

Для проверки корректности функциональных требований необходимо использовать следующие методы:

Анализ требований:

Цель: Обеспечение полноты, однозначности и выполнимости требований.

Методы: Ревью требований, анализ требований на согласованность и отсутствие противоречий.

Моделирование и прототипирование:

Цель: Проверка жизнеспособности и корректности требований на ранних стадиях разработки.

Методы: Создание моделей системы и прототипов, проведение сценариев использования.

Верификация требований:

Цель: Подтверждение того, что требования правильно отражают потребности пользователей и спецификации.

Методы: Ревью, инспекции и согласование требований с заинтересованными сторонами.

Тестирование:

Цель: Убедиться, что реализованная система соответствует функциональным требованиям.

Методы:

Юнит-тестирование: Проверка отдельных компонентов системы.

Интеграционное тестирование: Проверка взаимодействия между компонентами.

Системное тестирование: Полная проверка всей системы на соответствие функциональным требованиям.

Приемочное тестирование: Проверка соответствия системы ожиданиям и требованиям конечных пользователей.

Трассируемость требований:

Цель: Обеспечение возможности прослеживания реализации каждого требования на всех этапах разработки.

Методы: Создание матриц трассируемости, которые связывают требования с соответствующими элементами дизайна, кода и тестов.

**18. Понятие качества ПО. Стандарт определения качества ПО. Основные аспекты качества ПО по ISO 9126.**

Качество программного обеспечения (ПО) – это степень, до которой программное обеспечение удовлетворяет установленным требованиям, отвечает ожиданиям и нуждам пользователей, и не содержит дефектов. Качество ПО включает как функциональные аспекты, такие как правильность выполнения функций, так и нефункциональные аспекты, такие как производительность, надежность, удобство использования и безопасность.

Стандарт определения качества ПО

ISO/IEC 9126 – это международный стандарт для оценки качества программного обеспечения. Он определяет модель качества ПО и предоставляет метрики для его оценки. Стандарт разделен на четыре части:

Качество модели

Внешние метрики

Внутренние метрики

Качество в использовании

Основные аспекты качества ПО по ISO 9126

ISO 9126 определяет шесть основных характеристик качества программного обеспечения:

Функциональность (Functionality)

Соответствие требованиям (Suitability): Способность ПО предоставлять функции, соответствующие спецификациям и требованиям.

Точность (Accuracy): Способность ПО предоставлять правильные результаты с требуемой степенью точности.

Взаимодействие (Interoperability): Способность ПО взаимодействовать с другими системами или компонентами.

Безопасность (Security): Способность ПО защищать данные и предотвратить несанкционированный доступ.

Соответствие (Compliance): Способность ПО соответствовать стандартам, нормативам и правилам.

Надежность (Reliability)

Зрелость (Maturity): Способность ПО предотвращать дефекты при эксплуатации.

Отказоустойчивость (Fault Tolerance): Способность ПО поддерживать заданный уровень производительности при возникновении отказов.

Восстановимость (Recoverability): Способность ПО восстанавливаться после отказов и сбоев.

Удобство использования (Usability)

Понятность (Understandability): Легкость, с которой пользователь может понять, как использовать ПО.

Обучаемость (Learnability): Легкость, с которой пользователь может научиться использовать ПО.

Управляемость (Operability): Удобство использования ПО в рабочей среде.

Эффективность (Efficiency)

Производительность (Time Behaviour): Время отклика ПО и его производительность при выполнении задач.

Ресурсоемкость (Resource Utilisation): Способность ПО эффективно использовать ресурсы системы, такие как память и процессорное время.

Сопровождаемость (Maintainability)

Анализируемость (Analyzability): Легкость, с которой можно анализировать ПО для обнаружения дефектов или причин ошибок.

Изменяемость (Changeability): Легкость, с которой можно вносить изменения в ПО.

Стабильность (Stability): Способность ПО минимизировать неожиданные эффекты от изменений.

Тестируемость (Testability): Легкость, с которой можно тестировать ПО для обеспечения его правильности.

Переносимость (Portability)

Адаптируемость (Adaptability): Способность ПО быть адаптированным к различным средам эксплуатации.

Устанавливаемость (Installability): Легкость, с которой можно установить ПО в различных средах.

Соответствие (Conformance): Способность ПО соответствовать стандартам переносимости.

Заменяемость (Replaceability): Способность ПО быть замененным другим ПО с минимальными затратами.

**19. Характеристики и атрибуты качества ПО по ISO 9126 и их описание.**

ISO 9126 определяет шесть основных характеристик качества программного обеспечения, каждая из которых включает в себя несколько атрибутов. Эти характеристики и атрибуты помогают оценить и улучшить качество ПО. Ниже приводится описание каждой характеристики и соответствующих атрибутов.

1. Функциональность (Functionality)

Функциональность описывает способность программного обеспечения выполнять требуемые функции и задачи, соответствующие установленным требованиям и ожиданиям пользователей.

Соответствие требованиям (Suitability): Способность ПО предоставлять функции, соответствующие спецификациям и требованиям пользователей.

Точность (Accuracy): Способность ПО предоставлять правильные и точные результаты.

Взаимодействие (Interoperability): Способность ПО взаимодействовать с другими системами, обмениваться и использовать информацию.

Безопасность (Security): Способность ПО предотвращать несанкционированный доступ и защищать данные.

Соответствие (Compliance): Способность ПО соответствовать стандартам, законодательным и нормативным требованиям.

2. Надежность (Reliability)

Надежность характеризует способность ПО функционировать корректно в течение определенного времени и при определенных условиях.

Зрелость (Maturity): Способность ПО избегать дефектов и обеспечивать стабильную работу.

Отказоустойчивость (Fault Tolerance): Способность ПО поддерживать функциональность при возникновении сбоев.

Восстановимость (Recoverability): Способность ПО восстанавливаться после сбоев и обеспечивать восстановление данных.

3. Удобство использования (Usability)

Удобство использования оценивает, насколько легко и эффективно пользователи могут использовать программное обеспечение.

Понятность (Understandability): Легкость понимания назначения ПО и его функциональности пользователями.

Обучаемость (Learnability): Легкость, с которой пользователи могут научиться использовать ПО.

Управляемость (Operability): Удобство использования ПО, включая интерфейс и взаимодействие с системой.

4. Эффективность (Efficiency)

Эффективность характеризует производительность ПО и его способность эффективно использовать ресурсы системы.

Производительность (Time Behaviour): Время отклика и скорость выполнения функций ПО.

Ресурсоемкость (Resource Utilisation): Способность ПО эффективно использовать ресурсы системы, такие как память и процессорное время.

5. Сопровождаемость (Maintainability)

Сопровождаемость описывает способность ПО быть изменяемым и поддерживаемым с минимальными усилиями.

Анализируемость (Analyzability): Легкость, с которой можно анализировать ПО для обнаружения причин ошибок и дефектов.

Изменяемость (Changeability): Легкость, с которой можно вносить изменения в ПО.

Стабильность (Stability): Способность ПО минимизировать неожиданные эффекты от изменений.

Тестируемость (Testability): Легкость, с которой можно тестировать ПО для обеспечения его правильности после изменений.

6. Переносимость (Portability)

Переносимость характеризует способность ПО быть перенесенным и установленным в различных операционных средах.

Адаптируемость (Adaptability): Способность ПО адаптироваться к различным операционным средам с минимальными изменениями.

Устанавливаемость (Installability): Легкость, с которой можно установить ПО в различных средах.

Соответствие (Conformance): Способность ПО соответствовать стандартам переносимости.

Заменяемость (Replaceability): Способность ПО быть замененным другим ПО с минимальными затратами и усилиями.

**20. Метрики оценки качества ПО и их характеристики.**

Метрики оценки качества программного обеспечения (ПО) представляют собой количественные показатели, используемые для измерения и оценки различных аспектов качества ПО. Эти метрики помогают разработчикам, тестировщикам и менеджерам понять текущий уровень качества ПО и выявить области, требующие улучшения. Ниже представлены основные метрики оценки качества ПО и их характеристики.

1. Функциональные метрики

Функциональные метрики измеряют соответствие ПО требованиям и ожиданиям пользователей.

Процент покрытия требований (Requirements Coverage): Процент функциональных требований, которые были реализованы и протестированы.

Формула:

Процент покрытия=(Количество реализованных требований/Общее количество требований) \* 100

Количество дефектов (Defect Count): Общее количество выявленных дефектов в ПО.

Формула: Простое подсчитывание всех зарегистрированных дефектов.

Процент закрытых дефектов (Defect Closure Rate): Процент исправленных дефектов от общего количества выявленных.

Формула:

Процент закрытых дефектов=(Количество исправленных дефектов/Общее количество дефектов) \* 100

2. Метрики надежности

Метрики надежности оценивают способность ПО функционировать корректно в течение определенного времени.

Среднее время между отказами (Mean Time Between Failures, MTBF): Среднее время работы ПО между отказами.

Формула:

MTBF=Общее время работы/Количество отказов

Среднее время восстановления (Mean Time to Repair, MTTR): Среднее время, необходимое для восстановления ПО после отказа.

Формула:

MTTR=Общее время на восстановление/Количество отказов

Процент отказоустойчивости (Fault Tolerance Rate): Процент успешных операций, несмотря на возникшие сбои.

Формула:

Процент отказоустойчивости=(Количество успешных операций/Общее количество операций) \* 100

3. Метрики удобства использования

Метрики удобства использования измеряют насколько легко и эффективно пользователи могут использовать ПО.

Время на выполнение задачи (Task Completion Time): Среднее время, необходимое пользователю для выполнения определенной задачи с помощью ПО.

Формула: Среднее значение времени, затраченного пользователями на выполнение задачи.

Процент завершенных задач (Task Success Rate): Процент успешно выполненных пользователями задач.

Формула:

Процент завершенных задач=(Количество успешно завершенных задач/Общее количество задач) \* 100

Количество ошибок пользователей (User Error Rate): Количество ошибок, совершенных пользователями при использовании ПО.

Формула: Простое подсчитывание всех ошибок, совершенных пользователями.

4. Метрики эффективности

Метрики эффективности оценивают производительность ПО и его способность эффективно использовать ресурсы системы.

Время отклика (Response Time): Среднее время отклика системы на запросы пользователей.

Формула: Среднее значение времени отклика на запросы пользователей.

Использование ресурсов (Resource Utilization): Процент использования различных ресурсов системы (ЦП, память, диск и т.д.).

Формула:

Использование ресурсов=(Используемые ресурсы/Общее количество доступных ресурсов) \* 100

Производительность системы (Throughput): Количество операций или транзакций, выполненных системой за определенное время.

Формула: Количество операций/транзакций в единицу времени (например, операций в секунду).

5. Метрики сопровождаемости

Метрики сопровождаемости измеряют способность ПО быть изменяемым и поддерживаемым.

Сложность кода (Code Complexity): Уровень сложности кода, который влияет на его сопровождение.

Формула: Часто измеряется с использованием метрик цикломатической сложности (количество независимых путей через программу).

Процент изменяемого кода (Changeability Rate): Процент кода, который может быть изменен с минимальными усилиями.

Формула:

Процент изменяемого кода=(Количество измененных строк кода/Общее количество строк кода) \* 100

Время на исправление дефекта (Defect Fix Time): Среднее время, необходимое для исправления дефекта.

Формула: Среднее значение времени, затраченного на исправление каждого дефекта.

6. Метрики переносимости

Метрики переносимости оценивают способность ПО быть перенесенным и установленным в различных операционных средах.

Время установки (Installation Time): Среднее время, необходимое для установки ПО в различных средах.

Формула: Среднее значение времени, затраченного на установку ПО.

Процент успешных установок (Installation Success Rate): Процент успешных установок ПО в различных средах.

Формула:

Процент успешных установок=(Количество успешных установок/Общее количество установок) \* 100

Количество сред, поддерживаемых ПО (Supported Environments Count): Общее количество различных операционных сред, в которых ПО может работать.

Формула: Простое подсчитывание поддерживаемых сред.

**21. Классификация методов контроля качества.**

Методы контроля качества программного обеспечения (ПО) включают различные техники и процессы, используемые для обеспечения того, чтобы ПО соответствовало установленным требованиям и стандартам. Эти методы можно классифицировать по нескольким критериям: по этапам разработки, по способу проведения, по степени формальности и по цели. Ниже приводится классификация методов контроля качества.

1. По этапам разработки

Контроль качества на стадии проектирования:

Рецензирование (Review): Проверка проектной документации на соответствие требованиям и стандартам.

Аудит (Audit): Независимая проверка процесса проектирования и документации.

Контроль качества на стадии кодирования:

Код-ревью (Code Review): Проверка исходного кода разработчиками или специалистами по качеству.

Статический анализ кода (Static Code Analysis): Автоматическая проверка кода на предмет ошибок, соответствия стандартам кодирования и других проблем без выполнения программы.

Контроль качества на стадии тестирования:

Модульное тестирование (Unit Testing): Проверка отдельных модулей или компонентов ПО на корректность работы.

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

Системное тестирование (System Testing): Комплексная проверка всей системы на соответствие требованиям.

Приемочное тестирование (Acceptance Testing): Проверка ПО на соответствие требованиям заказчика перед его внедрением в эксплуатацию.

2. По способу проведения

Статические методы:

Рецензирование (Review): Формальная или неформальная проверка документации и исходного кода.

Анализ метрик (Metrics Analysis): Использование количественных показателей для оценки качества ПО.

Проверка соответствия стандартам (Standards Compliance Check): Проверка соответствия процессов и продуктов установленным стандартам.

Динамические методы:

Тестирование (Testing): Выполнение ПО для проверки его поведения и выявления дефектов.

Инспекция (Inspection): Формальный процесс проверки документации и исходного кода с целью выявления дефектов и нарушений стандартов.

Верификация и валидация (Verification and Validation): Процессы, направленные на подтверждение того, что ПО соответствует требованиям и ожиданиям пользователей.

3. По степени формальности

Формальные методы:

Инспекция (Inspection): Строго формализованный процесс проверки документации и исходного кода.

Аудит (Audit): Официальная проверка процессов и продуктов на соответствие стандартам и требованиям.

Неформальные методы:

Рецензирование (Review): Меньше формализованный процесс проверки документации и исходного кода, может включать обсуждение и обмен мнениями.

Walkthrough: Пошаговое обсуждение документации или кода с целью выявления дефектов и улучшения качества.

4. По цели

Профилактические методы (Preventive Methods):

Планирование качества (Quality Planning): Разработка планов и стратегий для обеспечения качества на всех этапах жизненного цикла ПО.

Тренинг и обучение (Training and Education): Подготовка специалистов по вопросам качества и стандартам разработки ПО.

Корректирующие методы (Corrective Methods):

Анализ причин дефектов (Root Cause Analysis): Определение причин возникновения дефектов и разработка мер по их устранению.

Корректирующие действия (Corrective Actions): Внедрение изменений, направленных на устранение выявленных дефектов и предотвращение их повторного возникновения.

**22. Понятие исключения, примеры исключений.**

Исключение в программировании представляет собой событие, которое возникает во время выполнения программы и нарушает её нормальное течение. Исключения могут быть вызваны различными ошибками, такими как попытка деления на ноль, обращение к несуществующему элементу массива, неверный тип данных, проблемы с доступом к файлам и т.д. Когда происходит исключение, программа может быть остановлена, если исключение не будет обработано специально разработанным кодом.

DivideByZeroException

Описание: Возникает при попытке деления на ноль.

try {

int result = 10 / 0;

} catch (DivideByZeroException e) {

Console.WriteLine("Ошибка: " + e.Message);

}

IndexOutOfRangeException

Описание: Возникает при попытке доступа к несуществующему индексу массива.

try {

int[] myArray = {1, 2, 3};

int element = myArray[5];

} catch (IndexOutOfRangeException e) {

Console.WriteLine("Ошибка: " + e.Message);

}

NullReferenceException

Описание: Возникает при попытке доступа к методу или свойству через null-объект.

try {

string str = null;

int length = str.Length;

} catch (NullReferenceException e) {

Console.WriteLine("Ошибка: " + e.Message);

}

FileNotFoundException

Описание: Возникает при попытке открытия несуществующего файла.

try {

using (StreamReader sr = new StreamReader("non\_existent\_file.txt")) {

string content = sr.ReadToEnd();

}

} catch (FileNotFoundException e) {

Console.WriteLine("Ошибка: " + e.Message);

}

InvalidCastException

Описание: Возникает при неправильном приведении типов.

try {

object obj = 100;

string str = (string) obj;

} catch (InvalidCastException e) {

Console.WriteLine("Ошибка: " + e.Message);

}

**23. Особенности функционального тестирования программного обеспечения (тестирования «черного ящика»). Ошибки, выявляемые при функциональном тестировании**

Функциональное тестирование программного обеспечения (ПО) относится к методам тестирования, при которых проверяется корректность выполнения функций системы, определенных в спецификациях и требованиях. Данный вид тестирования также известен как тестирование «черного ящика», поскольку тестировщики не используют знания о внутренней структуре или реализации ПО. Они сосредотачиваются исключительно на проверке внешнего поведения системы.

Основные особенности функционального тестирования

Тестирование на основе требований:

Функциональное тестирование проверяет выполнение ПО на соответствие функциональным требованиям и спецификациям. Оно подтверждает, что система делает то, что от нее ожидается.

Независимость от внутренней реализации:

Тестировщики рассматривают ПО как "черный ящик", то есть не учитывают внутреннюю структуру кода и реализации. Они работают исключительно с интерфейсом и функциональностью системы.

Использование тестовых сценариев:

Тестировщики разрабатывают тестовые сценарии и тест-кейсы на основе спецификаций требований. Эти сценарии включают входные данные и ожидаемые результаты, которые проверяются при выполнении тестов.

Разнообразие техник тестирования:

В функциональном тестировании используются различные техники, такие как эквивалентное разбиение, анализ граничных значений, таблицы решений, переходы состояний и т.д.

Проверка всех функциональных аспектов:

Тестируются все аспекты функциональности системы, включая корректность ввода и вывода данных, обработку ошибок, управление данными и их целостность, а также взаимодействие с другими системами.

Ошибки, выявляемые при функциональном тестировании

Функциональное тестирование направлено на выявление ошибок, связанных с выполнением функций системы. Основные типы ошибок включают:

Ошибки в логике и алгоритмах:

Некорректная реализация логики или алгоритмов, приводящая к неверным результатам или поведению системы.

Ошибки в управлении данными:

Неправильная обработка, хранение или передача данных, что может привести к потерям или искажениям данных.

Ошибки в пользовательском интерфейсе:

Некорректное отображение данных, неработающие элементы интерфейса, ошибки в навигации и взаимодействии с пользователем.

Ошибки в обработке ошибок:

Неправильное или неполное управление исключительными ситуациями и ошибками, что может привести к сбоям или аварийным завершениям программы.

Ошибки в интеграции:

Проблемы, возникающие при взаимодействии с другими системами или компонентами, такие как несовместимость интерфейсов или протоколов.

Ошибки в управлении доступом и безопасностью:

Недостатки в механизмах аутентификации, авторизации и защиты данных, которые могут привести к уязвимостям безопасности.

Ошибки в требованиях и спецификациях:

Несоответствие функциональности требованиям и спецификациям, что может быть результатом неправильного понимания или реализации требований.

**24. Задачи, решаемые при функциональном тестировании.**

Функциональное тестирование направлено на проверку и верификацию функциональности программного обеспечения (ПО) в соответствии с требованиями и спецификациями. Оно играет ключевую роль в процессе обеспечения качества ПО и решает несколько важных задач:

1. Проверка соответствия требованиям

Задача: Убедиться, что ПО выполняет все функции, описанные в спецификациях и требованиях.

Описание: Тестировщики создают тестовые сценарии и кейсы, которые проверяют выполнение каждой функции системы. Это помогает удостовериться, что ПО соответствует ожиданиям заказчика и конечных пользователей.

2. Выявление дефектов и ошибок

Задача: Обнаружить и задокументировать ошибки и дефекты, которые возникают при выполнении функциональных сценариев.

Описание: Тестирование помогает найти ошибки в логике программы, неправильные результаты, сбои в работе и другие дефекты, которые могут повлиять на работу системы.

3. Проверка пользовательского интерфейса (UI)

Задача: Убедиться, что интерфейс пользователя работает корректно и соответствует требованиям.

Описание: Функциональное тестирование проверяет элементы интерфейса, такие как кнопки, формы, меню и другие взаимодействия, чтобы удостовериться в их правильной работе и удобстве использования.

4. Проверка обработок данных

Задача: Убедиться, что ПО корректно обрабатывает входные и выходные данные.

Описание: Тестировщики проверяют, что данные правильно принимаются, обрабатываются и сохраняются в системе, а также корректно отображаются пользователям.

5. Проверка интеграции компонентов

Задача: Проверить взаимодействие между различными модулями и компонентами системы.

Описание: Тестирование интеграции помогает удостовериться, что компоненты системы корректно взаимодействуют друг с другом и обмениваются данными.

6. Проверка обработки ошибок и исключений

Задача: Убедиться, что система правильно обрабатывает ошибки и исключительные ситуации.

Описание: Тестировщики проверяют, как система реагирует на различные виды ошибок, такие как неправильный ввод данных, сбои в соединениях и другие нестандартные ситуации.

7. Проверка соответствия стандартам и нормативам

Задача: Убедиться, что ПО соответствует внутренним и внешним стандартам и нормативным требованиям.

Описание: Функциональное тестирование помогает проверить соответствие ПО стандартам безопасности, защиты данных, пользовательского интерфейса и другим нормативным требованиям.

8. Поддержка регрессионного тестирования

Задача: Обеспечить, что изменения в коде не нарушили существующую функциональность.

Описание: Функциональные тесты используются в регрессионном тестировании для проверки, что новые изменения или исправления не привели к появлению новых ошибок в ранее работающих частях системы.

9. Проверка локализации и интернационализации

Задача: Убедиться, что ПО корректно работает на различных языках и в различных региональных настройках.

Описание: Тестировщики проверяют, что система поддерживает несколько языков и адаптируется к региональным стандартам (например, формат даты, валюты и т.д.).

**25. Методы тестирования по принципу «черного ящика».**

Тестирование по принципу «черного ящика» включает методы, при которых тестировщики проверяют функциональность программного обеспечения без знания его внутренней структуры и кода. Они фокусируются на входных данных и ожидаемых выходных результатах, основываясь на спецификациях требований. Вот основные методы тестирования «черного ящика»:

1. Эквивалентное разбиение (Equivalence Partitioning)

Описание: Этот метод разделяет входные данные на группы (классы эквивалентности), где все значения внутри одной группы предполагают одинаковое поведение системы.

Пример: При тестировании функции, принимающей возраст, классы эквивалентности могут быть:

Возраст < 0 (недопустимое значение)

0 ≤ Возраст < 18 (несовершеннолетний)

18 ≤ Возраст < 65 (взрослый)

Возраст ≥ 65 (пожилой)

2. Анализ граничных значений (Boundary Value Analysis)

Описание: Этот метод фокусируется на тестировании границ классов эквивалентности, так как ошибки часто происходят на границах.

Пример: При тестировании функции, принимающей возраст от 1 до 100, нужно проверить:

Граничные значения: 1, 100

Значения за границами: 0, 101

3. Метод таблиц принятия решений (Decision Table Testing)

Описание: Этот метод использует таблицы решений для систематизации тестирования сложных логических условий. В таблице решений перечисляются все возможные комбинации условий и действий.

Пример: Для системы скидок на основе членства и суммы покупки таблица решений может включать комбинации «Членство: Да/Нет» и «Сумма покупки: <50 / ≥50» с соответствующими скидками.

4. Метод состояний и переходов (State Transition Testing)

Описание: Этот метод проверяет систему на корректность переходов между состояниями в ответ на различные события или условия.

Пример: Для банкомата:

Состояния: «Готов к работе», «Запрос ПИН-кода», «Выдача наличных»

Переходы: «Ввод карты» (переход к «Запрос ПИН-кода»), «Ввод ПИН-кода» (переход к «Выдача наличных»)

5. Метод использования диаграмм (Use Case Testing)

Описание: Этот метод основан на проверке функциональности системы в соответствии с описанными сценариями использования (use cases), которые описывают взаимодействие пользователя с системой.

Пример: Для интернет-магазина:

Use case «Оформление заказа»: Включает шаги «Выбор товара», «Добавление в корзину», «Ввод данных доставки», «Оплата».

6. Метод тестирования на основе требований (Requirements-Based Testing)

Описание: Этот метод тестирования фокусируется на проверке требований к системе. Каждый тест-кейс разрабатывается для проверки соответствия требованиям.

Пример: Требование «Система должна отправлять подтверждение заказа по электронной почте» тестируется проверкой отправки электронной почты после оформления заказа.

7. Метод тестирования случайным образом (Random Testing)

Описание: В этом методе входные данные выбираются случайным образом из возможных значений, чтобы проверить поведение системы в случайных условиях.

Пример: При тестировании формы ввода пользовательских данных можно генерировать случайные строки, числа и символы для проверки валидации полей.

**26. Методы тестирования по принципу «белого ящика».**

Тестирование по принципу «белого ящика», также известное как структурное или стеклянное тестирование, предполагает проверку внутренней структуры и реализации кода программы. Тестировщики используют знание кода для создания тестов, которые охватывают все возможные пути выполнения программы. Основные методы тестирования по принципу «белого ящика» включают:

1. Покрытие операторов (Statement Coverage)

Описание: Этот метод проверяет, что каждый оператор в программе был выполнен хотя бы один раз.

Пример: Если программа состоит из условий и циклов, тесты должны быть написаны так, чтобы каждая строка кода была выполнена хотя бы раз.

2. Покрытие ветвей (Branch Coverage)

Описание: Этот метод проверяет, что каждая ветвь (true/false) в условных операторах (например, if-else) была выполнена.

Пример: В случае if-else конструкции, тесты должны убедиться, что оба пути выполнения (ветвь if и ветвь else) были протестированы.

3. Покрытие условий (Condition Coverage)

Описание: Этот метод проверяет, что каждое отдельное логическое условие в условных операторах было проверено на истинность и ложность.

Пример: В выражении if (A && B), необходимо проверить случаи, когда A и B принимают значения true и false.

4. Покрытие пути (Path Coverage)

Описание: Этот метод проверяет, что все возможные пути выполнения программы были пройдены.

Пример: Для сложных программ с многочисленными ветвлениями и циклами тесты должны охватывать все возможные пути выполнения.

5. Покрытие многократных условий (Multiple Condition Coverage)

Описание: Этот метод проверяет все возможные комбинации условий в сложных логических выражениях.

Пример: В выражении if (A || (B && C)), необходимо проверить все возможные комбинации значений A, B и C (true/false).

6. Тестирование циклов (Loop Testing)

Описание: Этот метод фокусируется на проверке правильности выполнения циклов.

Пример: Для каждого цикла необходимо проверить выполнение 0 раз (пропуск цикла), 1 раз, и множество раз (состояние выхода из цикла).

7. Тестирование на основе данных (Data Flow Testing)

Описание: Этот метод анализирует поток данных внутри программы, проверяя пути, по которым данные перемещаются от их создания до использования и уничтожения.

Пример: Проверка, что переменная была инициализирована перед использованием, и что данные корректно передаются между функциями.

8. Тестирование управления (Control Flow Testing)

Описание: Этот метод проверяет логическую структуру программы, фокусируясь на контрольных точках и управлении потоком выполнения.

Пример: Использование графов потока управления для идентификации и тестирования всех возможных маршрутов выполнения.

**27. Понятие системного тестирования. Цели и задачи системного тестирования.**

Системное тестирование — это этап тестирования программного обеспечения, на котором тестируется полностью интегрированная система для проверки её соответствия заданным требованиям. На этом этапе тестируется система в целом, включая все её компоненты, модули и взаимодействие с внешними системами и пользователями.

Цели системного тестирования

Проверка соответствия требованиям:

Убедиться, что система выполняет все функции, описанные в спецификациях и требованиях.

Обнаружение дефектов:

Найти и устранить ошибки и дефекты, которые не были выявлены на предыдущих этапах тестирования (модульное и интеграционное тестирование).

Проверка взаимодействия компонентов:

Убедиться, что все компоненты системы корректно взаимодействуют друг с другом.

Оценка поведения системы:

Проверить, как система работает в различных условиях эксплуатации, включая нормальные и экстремальные условия.

Верификация пользовательских сценариев:

Убедиться, что система соответствует ожиданиям и требованиям конечных пользователей.

Системное тестирование решает следующие ключевые задачи:

Функциональное тестирование:

Проверка, что система выполняет все функции, описанные в требованиях.

Нефункциональное тестирование:

Оценка таких характеристик системы, как производительность, безопасность, удобство использования, надежность и масштабируемость.

Тестирование интеграции:

Проверка взаимодействия между различными компонентами системы и их совместной работы.

Тестирование интерфейсов:

Проверка корректности работы интерфейсов между компонентами системы, а также внешними системами и пользователями.

Тестирование на соответствие стандартам и нормативам:

Убедиться, что система соответствует отраслевым стандартам, нормативным требованиям и корпоративным политикам.

Тестирование совместимости:

Проверка, что система корректно работает на различных платформах, устройствах, операционных системах и браузерах.

Тестирование отказоустойчивости и восстановления:

Проверка поведения системы при возникновении отказов и её способности восстанавливаться после них.

Регрессионное тестирование:

Проверка, что изменения в системе (исправления ошибок, добавление новых функций) не привели к появлению новых ошибок в ранее работающих частях системы.

**28. Типы системного тестирования. Этапы системного тестирования.**

Системное тестирование включает различные типы тестирования, каждый из которых направлен на проверку определённых аспектов системы. Основные типы системного тестирования включают:

Функциональное тестирование (Functional Testing):

Цель: Проверка того, что система выполняет свои функции в соответствии с требованиями.

Примеры: Проверка обработки транзакций, авторизации пользователей, генерации отчетов.

Нефункциональное тестирование (Non-Functional Testing):

Цель: Проверка нефункциональных характеристик системы, таких как производительность, безопасность и удобство использования.

Примеры:

Тестирование производительности (Performance Testing): Проверка времени отклика и пропускной способности.

Тестирование надежности (Reliability Testing): Оценка способности системы функционировать без отказов.

Тестирование безопасности (Security Testing): Проверка защиты данных и устойчивости к атакам.

Тестирование удобства использования (Usability Testing): Оценка удобства взаимодействия пользователей с системой.

Регрессионное тестирование (Regression Testing):

Цель: Обнаружение ошибок, введённых в систему после изменений или исправлений.

Примеры: Повторное тестирование основных функций системы после обновления программного обеспечения.

Интеграционное тестирование (Integration Testing):

Цель: Проверка взаимодействия между различными модулями и компонентами системы.

Примеры: Проверка взаимодействия между базой данных и пользовательским интерфейсом.

Тестирование совместимости (Compatibility Testing):

Цель: Проверка работы системы в различных средах и на различных устройствах.

Примеры: Проверка работы веб-приложения в разных браузерах и операционных системах.

Тестирование на основе сценариев (Scenario Testing):

Цель: Проверка системы с использованием реалистичных сценариев использования, отражающих поведение пользователей.

Примеры: Проверка полного процесса заказа товара в интернет-магазине от выбора товара до оплаты и получения подтверждения.

Этапы системного тестирования

Системное тестирование проходит через несколько этапов, каждый из которых включает определённые задачи и цели:

Планирование тестирования (Test Planning):

Описание: Разработка стратегии и плана тестирования.

Задачи:

Определение целей и объёма тестирования.

Назначение ролей и ответственности.

Определение ресурсов и инструментов, необходимых для тестирования.

Создание расписания тестирования.

Разработка тестов (Test Design):

Описание: Создание тестовых сценариев и случаев на основе требований и спецификаций.

Задачи:

Анализ требований и спецификаций.

Определение тестовых данных и условий.

Разработка тестовых сценариев и случаев.

Подготовка тестовой среды (Test Environment Setup):

Описание: Настройка необходимой среды для проведения тестирования.

Задачи:

Установка программного обеспечения и конфигурация оборудования.

Подготовка тестовых данных.

Проверка готовности тестовой среды.

Выполнение тестов (Test Execution):

Описание: Проведение тестирования согласно разработанным тестовым сценариям и случаям.

Задачи:

Выполнение тестовых сценариев.

Регистрация результатов тестирования.

Сообщение об обнаруженных дефектах.

Анализ и отчётность (Test Analysis and Reporting):

Описание: Анализ результатов тестирования и подготовка отчётов.

Задачи:

Анализ обнаруженных дефектов и проблем.

Оценка степени покрытия тестов.

Подготовка отчётов о тестировании.

Завершение тестирования (Test Closure):

Описание: Окончание процесса тестирования и оценка его результатов.

Задачи:

Проведение итогового анализа и оценка выполнения целей тестирования.

Документация результатов и выводов.

Архивирование тестовых артефактов и данных.

Проведение ретроспективы и выявление улучшений для будущих тестовых циклов.

**29. Достоинства и недостатки системного тестирования.**

Достоинства системного тестирования

Целостное тестирование системы (Holistic Testing):

Преимущество: Позволяет проверить всю систему в целом, включая все её компоненты и модули.

Результат: Уверенность в том, что система работает как единое целое.

Выявление интеграционных дефектов (Detection of Integration Issues):

Преимущество: Помогает выявить проблемы взаимодействия между различными компонентами и модулями.

Результат: Обеспечивает совместимость и корректное взаимодействие всех частей системы.

Проверка выполнения требований (Verification of Requirements Fulfillment):

Преимущество: Проверяет, что система соответствует всем функциональным и нефункциональным требованиям.

Результат: Уверенность в том, что система выполняет все заявленные функции и обладает необходимыми характеристиками.

Реалистичные сценарии (Realistic Scenarios):

Преимущество: Использует реальные сценарии использования, что помогает выявить дефекты, которые могут возникнуть в реальных условиях эксплуатации.

Результат: Обеспечивает адекватное поведение системы в реальной среде.

Обнаружение критических дефектов (Detection of Critical Issues):

Преимущество: Позволяет обнаружить дефекты, которые могут быть критичными для работы всей системы.

Результат: Снижает риск сбоев и серьезных проблем на этапе эксплуатации.

Недостатки системного тестирования

Высокая стоимость (High Cost):

Недостаток: Системное тестирование требует значительных затрат на ресурсы, время и усилия.

Результат: Повышенные затраты на проведение тестирования, особенно в больших и сложных проектах.

Продолжительность тестирования (Time-Consuming):

Недостаток: Поскольку системное тестирование охватывает всю систему, оно может занимать много времени.

Результат: Задержки в графике проекта, особенно если обнаруживаются серьёзные дефекты, требующие значительных исправлений.

Сложность тестирования (Complexity):

Недостаток: Тестирование больших и сложных систем требует тщательной подготовки, планирования и координации.

Результат: Высокая вероятность ошибок в процессе тестирования и необходимость в высококвалифицированных специалистах.

Трудности в изоляции дефектов (Difficulty in Isolating Issues):

Недостаток: Обнаруженные дефекты могут быть сложно изолировать и идентифицировать конкретный компонент или модуль, вызывающий проблему.

Результат: Увеличение времени на анализ и исправление дефектов.

Невозможность полного охвата (Impossibility of Full Coverage):

Недостаток: Невозможно протестировать все возможные сценарии и условия эксплуатации системы.

Результат: Некоторые дефекты могут оставаться невыявленными до этапа реальной эксплуатации.

**30. Понятие нагрузочного тестирования, приложения, виртуального пользователя, итерации, нагрузки, производительности, масштабируемости приложения, профиля нагрузки, нагрузочной точки, теста производительности.**

Нагрузочное тестирование (Load Testing) - это вид тестирования, направленный на проверку производительности системы при увеличении нагрузки. Целью является определение, как система справляется с большим количеством пользователей, запросов или данных, а также выявление узких мест и точек отказа.

Приложение (Application):

Описание: Программное обеспечение, которое тестируется на предмет производительности под нагрузкой.

Пример: Веб-сайт, мобильное приложение, база данных.

Виртуальный пользователь (Virtual User):

Описание: Симулированный пользователь, который выполняет действия в системе во время нагрузочного тестирования.

Пример: Сценарий входа в систему, добавления товара в корзину, выполнения поиска.

Итерация (Iteration):

Описание: Однократное выполнение тестового сценария виртуальным пользователем.

Пример: Один цикл выполнения сценария покупки в интернет-магазине.

Нагрузка (Load):

Описание: Количество запросов, пользователей или операций, одновременно обрабатываемых системой.

Пример: 1000 одновременных пользователей, отправляющих запросы к серверу.

Производительность (Performance):

Описание: Способность системы быстро и эффективно обрабатывать запросы под нагрузкой.

Пример: Время отклика сервера, количество запросов в секунду.

Масштабируемость приложения (Scalability of Application):

Описание: Способность системы увеличивать свою производительность с увеличением ресурсов.

Пример: Увеличение числа серверов, что приводит к пропорциональному увеличению обрабатываемых запросов.

Профиль нагрузки (Load Profile):

Описание: Описание характера нагрузки, включающее количество пользователей, типы запросов и их частоту.

Пример: В течение часа пиковая нагрузка составляет 5000 пользователей, каждый из которых выполняет 10 запросов в минуту.

Нагрузочная точка (Load Point):

Описание: Конкретное значение нагрузки, при которой проводятся измерения производительности системы.

Пример: Точка нагрузки в 3000 одновременных пользователей.

Тест производительности (Performance Test):

Описание: Тест, направленный на измерение характеристик производительности системы под различной нагрузкой.

Пример: Измерение времени отклика системы при различных уровнях нагрузки, выявление узких мест.

**31. Цели нагрузочного тестирования.**

Нагрузочное тестирование проводится для достижения нескольких важных целей, которые помогают обеспечить надёжность, производительность и масштабируемость системы. Основные цели нагрузочного тестирования включают:

Проверка производительности под нагрузкой:

Цель: Определить, как система справляется с определённой нагрузкой (количеством пользователей, запросов или данных).

Описание: Измерение времени отклика, пропускной способности и других показателей производительности при различных уровнях нагрузки.

Выявление узких мест и точек отказа:

Цель: Обнаружить компоненты системы, которые становятся «узкими местами» при увеличении нагрузки.

Описание: Определение модулей или ресурсов (например, процессор, память, дисковая подсистема), которые ограничивают производительность системы.

Оценка масштабируемости:

Цель: Определить способность системы увеличивать производительность при увеличении ресурсов (например, добавлении серверов).

Описание: Тестирование с увеличивающейся нагрузкой для оценки, как система масштабируется и где возникают ограничения.

Проверка стабильности и надежности:

Цель: Убедиться, что система может работать стабильно под нагрузкой в течение длительного времени.

Описание: Проведение длительных тестов, чтобы выявить потенциальные проблемы, такие как утечки памяти или деградация производительности со временем.

Проверка соответствия SLA (Service Level Agreement):

Цель: Убедиться, что система соответствует оговорённым уровням обслуживания.

Описание: Проверка, что система выполняет свои функции в рамках заданных показателей производительности, таких как время отклика или пропускная способность.

Подготовка к пиковым нагрузкам:

Цель: Убедиться, что система сможет справиться с ожидаемыми пиковыми нагрузками.

Описание: Моделирование пиковых условий (например, во время распродаж, крупных событий) и проверка, как система справляется с резким увеличением нагрузки.

Оценка эффективности оптимизаций:

Цель: Проверить, как внесенные изменения или оптимизации влияют на производительность системы.

Описание: Сравнение результатов тестов до и после изменений для оценки их влияния на производительность.

Обеспечение качественного пользовательского опыта:

Цель: Убедиться, что пользователи не столкнутся с проблемами производительности при использовании системы.

Описание: Проверка, что время отклика и другие параметры находятся на приемлемом уровне для обеспечения положительного пользовательского опыта.

**32. Виды нагрузочного теста.**

Нагрузочное тестирование включает различные виды тестов, каждый из которых имеет свои цели и методики. Основные виды нагрузочного тестирования включают:

Нагрузочное тестирование (Load Testing):

Цель: Определить, как система справляется с определённой рабочей нагрузкой.

Описание: Проверка производительности системы при увеличении количества пользователей или запросов до нормального или ожидаемого уровня. Этот вид тестирования помогает определить, как система будет работать при стандартной эксплуатационной нагрузке.

Стресс-тестирование (Stress Testing):

Цель: Оценить пределы производительности системы и её поведение при экстремальных нагрузках.

Описание: Создание условий, превышающих обычные рабочие нагрузки, чтобы выявить, как система справляется с перегрузками и каковы её точки отказа. Этот вид тестирования помогает определить устойчивость системы и её способность восстанавливаться после сбоев.

Тестирование пиковых нагрузок (Spike Testing):

Цель: Оценить, как система справляется с внезапными и кратковременными пиками нагрузки.

Описание: Симулирование резкого увеличения нагрузки за короткий период времени и наблюдение за реакцией системы. Этот вид тестирования важен для систем, которые могут столкнуться с внезапными всплесками активности, например, при запуске рекламных кампаний или распродаж.

Тестирование устойчивости (Soak Testing или Endurance Testing):

Цель: Проверить, как система справляется с длительными нагрузками.

Описание: Применение умеренной постоянной нагрузки на продолжительное время (часами или днями) для выявления проблем, таких как утечки памяти, деградация производительности и другие долгосрочные эффекты. Этот вид тестирования помогает оценить стабильность и надежность системы в течение длительного периода эксплуатации.

Тестирование масштабируемости (Scalability Testing):

Цель: Определить способность системы масштабироваться при увеличении нагрузки.

Описание: Проверка, как увеличение ресурсов (например, добавление серверов) влияет на производительность системы. Этот вид тестирования помогает определить, насколько эффективно система может увеличивать свою производительность при увеличении нагрузки.

Тестирование конфигурации (Configuration Testing):

Цель: Оценить влияние различных конфигураций системы на её производительность.

Описание: Тестирование системы при различных конфигурациях оборудования и программного обеспечения, чтобы определить оптимальные настройки для наилучшей производительности. Этот вид тестирования помогает найти наиболее эффективные конфигурации для работы системы.

Тестирование конкурентного доступа (Concurrency Testing):

Цель: Проверить, как система справляется с одновременным доступом множества пользователей.

Описание: Моделирование большого количества пользователей, одновременно выполняющих различные операции, чтобы выявить проблемы, связанные с блокировками, состояниями гонки и другими проблемами конкурентного доступа. Этот вид тестирования важен для систем, где много пользователей взаимодействуют с одной и той же базой данных или ресурсами одновременно.

**33. Понятие стресс-тестирования. Цели стресс-тестирования.**

Стресс-тестирование (Stress Testing) - это тип нагрузочного тестирования, целью которого является проверка поведения системы при экстремальных нагрузках и условиях, превышающих её нормальные эксплуатационные параметры. Стресс-тестирование помогает выявить пределы производительности системы и её устойчивость к отказам.

Цели стресс-тестирования

Определение пределов производительности системы:

Цель: Узнать максимальные возможности системы при экстремальных условиях.

Описание: Проверка, как система справляется с увеличением нагрузки до точек, где она начинает деградировать или отказывать. Это помогает понять максимальные нагрузки, которые система может выдержать.

Выявление точек отказа и узких мест:

Цель: Определить, где и когда система начинает проявлять нестабильность или отказывать.

Описание: Стресс-тестирование помогает обнаружить компоненты, которые становятся «узкими местами» при экстремальных нагрузках, и точки, где система выходит из строя.

Проверка устойчивости системы к отказам:

Цель: Оценить, как система справляется с экстремальными условиями и восстановлением после сбоев.

Описание: Тестирование показывает, насколько система устойчива к нагрузкам, превышающим нормальные условия эксплуатации, и как она восстанавливается после отказов.

Оценка поведения системы под перегрузкой:

Цель: Понять, как система ведет себя при перегрузке.

Описание: Стресс-тестирование позволяет оценить, как система реагирует на значительные перегрузки, включая возможные деградации производительности, замедление отклика, ошибки и сбои.

Проверка систем на устойчивость к нештатным ситуациям:

Цель: Проверить, как система справляется с неожиданными и экстремальными ситуациями.

Описание: Создание сценариев, имитирующих нештатные и аварийные ситуации (например, резкое увеличение числа пользователей, отказ серверов или сетевых компонентов) для оценки реакции системы.

Проверка долговременной устойчивости:

Цель: Оценить, как система работает под высокой нагрузкой в течение длительного времени.

Описание: Выявление проблем, таких как утечки памяти и другие ресурсоемкие ошибки, которые проявляются только при длительных высоких нагрузках.

Оценка отказоустойчивости и восстановления:

Цель: Оценить, насколько быстро и эффективно система может восстановиться после сбоя.

Описание: Проверка механизмов резервирования, отказоустойчивости и восстановления, чтобы убедиться, что система может быстро вернуться к нормальной работе после сбоя.

**34. Преимущества и недостатки стресс-тестирования.**

Преимущества стресс-тестирования

Выявление пределов производительности:

Преимущество: Помогает определить максимальные возможности системы и её поведение при экстремальных нагрузках.

Результат: Информация о предельных условиях эксплуатации, что позволяет лучше подготовиться к неожиданным пиковым нагрузкам.

Обнаружение узких мест и точек отказа:

Преимущество: Позволяет выявить компоненты, которые становятся узкими местами или точками отказа под высокими нагрузками.

Результат: Возможность оптимизировать систему и устранить выявленные проблемы до их появления в реальных условиях.

Проверка устойчивости к сбоям:

Преимущество: Оценка, как система реагирует на сбои и её способность восстанавливаться после них.

Результат: Повышение надежности и устойчивости системы, что важно для критически важных приложений.

Повышение качества и надежности системы:

Преимущество: Обеспечивает уверенность в том, что система способна работать в условиях экстремальных нагрузок.

Результат: Более высокая степень надежности и стабильности системы в реальных эксплуатационных условиях.

Подготовка к реальным экстремальным условиям:

Преимущество: Моделирование реальных ситуаций с экстремальными нагрузками, такими как распродажи, крупные события или аварийные ситуации.

Результат: Система лучше подготовлена к внезапным пиковым нагрузкам и нештатным ситуациям.

Недостатки стресс-тестирования

Высокие затраты на проведение:

Недостаток: Стресс-тестирование требует значительных ресурсов, времени и усилий.

Результат: Повышенные затраты на проведение тестов, особенно для сложных и больших систем.

Сложность моделирования экстремальных условий:

Недостаток: Трудности в точном воспроизведении реальных экстремальных условий и сценариев.

Результат: Ограниченная точность тестов и возможные расхождения с реальными условиями эксплуатации.

Риск повреждения системы:

Недостаток: Проведение стресс-тестов может привести к сбоям и повреждениям системы.

Результат: Возможность временного вывода системы из строя или необходимости её восстановления после тестов.

Не всегда отражает реальную эксплуатацию:

Недостаток: Экстремальные условия, создаваемые в процессе стресс-тестирования, могут не всегда полностью соответствовать реальным условиям эксплуатации.

Результат: Результаты тестирования могут быть частично нерелевантны для обычных условий использования системы.

Трудности в интерпретации результатов:

Недостаток: Результаты стресс-тестирования могут быть сложны для интерпретации и анализа.

Результат: Необходимость в высокой квалификации специалистов для правильной оценки результатов и принятия решений по улучшению системы.

**35. Виды стресс-тестов. Виды результатов стресс-тестов. Этапы проведения тест-тестирования.**

Виды стресс-тестов

Классическое стресс-тестирование (Classical Stress Testing):

Описание: Постепенное увеличение нагрузки на систему до тех пор, пока она не начнет давать сбои или достигнет своих пределов производительности.

Пример: Постепенное увеличение количества пользователей или запросов до тех пор, пока система не начнет замедляться или отказывать.

Тестирование на предельную нагрузку (Load Limit Testing):

Описание: Определение максимальной нагрузки, которую система может выдержать без серьезных сбоев или деградации производительности.

Пример: Определение максимального количества одновременных пользователей, которых система может обслуживать без значительного увеличения времени отклика.

Тестирование на деградацию (Degradation Testing):

Описание: Проверка, как система справляется с постепенной деградацией ресурсов (например, уменьшение памяти или пропускной способности).

Пример: Постепенное уменьшение доступной памяти или полосы пропускания сети и наблюдение за реакцией системы.

Тестирование отказа компонентов (Component Failure Testing):

Описание: Симуляция отказа отдельных компонентов системы (например, серверов, баз данных) и проверка реакции системы.

Пример: Отключение одного из серверов в кластере и наблюдение за тем, как оставшиеся серверы справляются с увеличенной нагрузкой.

Тестирование на пиковую нагрузку (Spike Testing):

Описание: Проверка реакции системы на внезапные и кратковременные пики нагрузки.

Пример: Внезапное увеличение количества пользователей на короткий период времени и наблюдение за реакцией системы.

Виды результатов стресс-тестов

Пределы производительности (Performance Limits):

Описание: Максимальные значения нагрузки, при которых система продолжает работать без серьезных сбоев.

Пример: Максимальное количество запросов в секунду, которое система может обработать без деградации времени отклика.

Узкие места (Bottlenecks):

Описание: Компоненты или процессы, которые ограничивают производительность системы под нагрузкой.

Пример: Выявление базы данных, как узкого места, из-за длительного времени выполнения запросов при высокой нагрузке.

Пределы отказоустойчивости (Fault Tolerance Limits):

Описание: Способность системы продолжать работу при отказе одного или нескольких её компонентов.

Пример: Проверка, что система продолжает функционировать при отказе одного из серверов.

Сценарии отказа (Failure Scenarios):

Описание: Условия и ситуации, при которых система начинает давать сбои.

Пример: Сценарии, при которых система перестает отвечать на запросы при достижении определенного уровня нагрузки.

Поведение системы под нагрузкой (System Behavior Under Load):

Описание: Характеристики и поведение системы при различных уровнях нагрузки.

Пример: Наблюдение за тем, как изменяется время отклика и пропускная способность при увеличении числа пользователей.

Этапы проведения стресс-тестирования

Планирование (Planning):

Действие: Определение целей стресс-тестирования, выбор метрик, создание сценариев тестирования и планирование ресурсов.

Результат: Подробный план стресс-тестирования с определением всех необходимых шагов и критериев.

Подготовка среды (Environment Preparation):

Действие: Настройка тестовой среды, установка и конфигурация инструментов для стресс-тестирования.

Результат: Готовая к тестированию среда, которая максимально точно воспроизводит реальную эксплуатационную систему.

Создание тестовых сценариев (Test Script Creation):

Действие: Разработка сценариев тестирования, которые будут использоваться для симуляции нагрузки.

Результат: Набор тестовых сценариев, готовых к выполнению.

Проведение тестов (Test Execution):

Действие: Запуск стресс-тестов и мониторинг системы в процессе выполнения.

Результат: Сбор данных о производительности и поведении системы под нагрузкой.

Анализ результатов (Results Analysis):

Действие: Анализ собранных данных, идентификация узких мест и точек отказа.

Результат: Отчет о результатах стресс-тестирования с рекомендациями по улучшению производительности и устойчивости системы.

Отчетность и рекомендации (Reporting and Recommendations):

Действие: Создание детального отчета о проведенном тестировании и предоставление рекомендаций по оптимизации системы.

Результат: Документированный отчет с анализом результатов и предложениями по улучшению.

Корректирующие действия (Corrective Actions):

Действие: Внесение изменений и улучшений в систему на основе результатов стресс-тестирования.

Результат: Оптимизированная и улучшенная система, готовая к реальной эксплуатации.

**36. Понятие интеграционного тестирования. Методы интеграционного тестирования.**

Интеграционное тестирование (Integration Testing) — это этап тестирования программного обеспечения, при котором отдельные модули или компоненты объединяются и тестируются как единое целое. Цель интеграционного тестирования — выявить дефекты, связанные с взаимодействием между модулями, и убедиться, что интегрированные компоненты работают корректно вместе.

Методы интеграционного тестирования

Большой взрыв (Big Bang Integration):

Описание: Все модули или компоненты интегрируются одновременно, а затем тестируются как единое целое.

Преимущества: Простой метод, не требующий сложного планирования интеграции.

Недостатки: Трудности в выявлении источника ошибок, если тест не проходит, поскольку все компоненты интегрированы сразу. Может быть трудно изолировать и отладить проблемы.

Нисходящее интеграционное тестирование (Top-Down Integration Testing):

Описание: Начинается с интеграции верхнего уровня модулей (или главного модуля) и постепенно интегрируются модули более низкого уровня. Модули нижнего уровня заменяются заглушками (stubs), пока они не будут готовы.

Преимущества: Облегчает обнаружение ошибок в высокоуровневом дизайне и архитектуре. Функциональность верхнего уровня может быть протестирована рано.

Недостатки: Требует создания множества заглушек, что может быть трудоемким и времязатратным.

Восходящее интеграционное тестирование (Bottom-Up Integration Testing):

Описание: Начинается с интеграции модулей нижнего уровня, которые тестируются первыми, а затем постепенно интегрируются модули более высокого уровня. Модули верхнего уровня заменяются драйверами, пока они не будут готовы.

Преимущества: Легче изолировать ошибки и определить их источник. Нижние уровни могут быть тщательно протестированы перед интеграцией с верхними уровнями.

Недостатки: Требует создания множества драйверов. Высокоуровневая функциональность не может быть протестирована на ранних стадиях.

Интеграционное тестирование на основе функциональных групп (Functional Group Integration):

Описание: Модули, связанные общей функциональностью, интегрируются и тестируются вместе. После успешного тестирования этих групп, группы объединяются друг с другом и тестируются.

Преимущества: Обеспечивает более целенаправленный подход к тестированию интеграции. Упрощает выявление проблем в конкретных функциональных группах.

Недостатки: Может быть сложно идентифицировать все взаимосвязанные модули. Тестирование всех функциональных групп требует тщательного планирования.

Инкрементное интеграционное тестирование (Incremental Integration Testing):

Описание: Модули интегрируются и тестируются поэтапно, один за другим, в определенном порядке.

Преимущества: Облегчает выявление и устранение ошибок на ранних этапах интеграции. Тестирование проводится постепенно, что упрощает управление процессом.

Недостатки: Процесс может занять больше времени по сравнению с большим взрывом. Требует тщательного планирования последовательности интеграции.

Тестирование на основе интерфейсов (Interface Integration Testing):

Описание: Фокусируется на тестировании взаимодействия между интерфейсами модулей или компонентов.

Преимущества: Позволяет проверить, что интерфейсы правильно обрабатывают данные и взаимодействуют друг с другом. Выявляет ошибки в передаче данных между модулями.

Недостатки: Требует подробного понимания интерфейсов и их взаимодействий. Может быть сложно тестировать в изоляции от функциональности модулей.

**37. Структура итогового отчета о результатах тестирования.**

Итоговый отчет можно разделить на части с соответствующей информацией:

1 Общая информация.

2 Сведения о том, кто и когда тестировал программный продукт.

3 Тестовое окружение.

4 Общая оценка качества приложения.

5 Обоснование выставленного качества.

6 Графическое представление результатов тестирования.

7 Детализированный анализ качества по модулям.

8 ТОП-5 самых критичных дефектов.

9 Рекомендации.

**38. Содержание разделов «Общая информация», «Тестовое окружение» итогового отчета о результатах тестирования.**

Общая информация включает:

* название проекта;
* номер сборки;
* модули, которые подверглись тестированию (в случае, если тестировался не весь проект);
* виды тестов по глубине покрытия (Smoke Test, Minimal Acceptance Test, Acceptance Test), тестовые активности (New Feature Test, Regression Testing, Defect Validation);
* количество обнаруженных дефектов;
* вид рабочей тестовой документации (Acceptance Sheet, Test Survey, Test Cases).

Тестовое окружение содержит: ссылку на проект, браузер, операционную систему и другую информацию, конкретизирующую особенности конфигурации.

**39. Содержание раздела «Детализированный анализ качества» отчета о результатах тестирования. отчета о результатах тестирования.**

В данной части отчета описывается более подробная информация о проверенных частях функционала, устанавливается качество каждой проверенной части функционала (модуля) в отдельности, дается аргументация выставленного уровня качества. Как правило, данный раздел отчета представляется в табличной форме. В зависимости от вида проводимых тестовых активностей, эта часть отчета будет отличаться.

При оценке качества функционала на уровне Smoke теста, оно может быть либо Приемлемым (Acceptable), либо Неприемлемым (Unacceptable).

Если все наиболее важные функции работают корректно, то качество всего функционала на уровне Smoke может быть оценено, как Приемлемое. Если это релизная или предрелизная сборка, то для выставления Приемлемого качества на уровне Smoke не должно быть найдено функциональных дефектов.

В части о детализированной информации качества сборки следует более подробно описать проблемы, которые были найдены во время теста.

При оценке качества функционала на уровне Defect Validation указывается качество о проведении валидации дефектов, а именно:

* общее количество всех дефектов, поступивших на проверку;
* количество неисправленных дефектов и их процент от общего количества;
* список дефектов, которые не были проверены и причины, по которым этого не было сделано;
* наглядная таблица с неисправленными дефектами.

По вышеуказанным результатам выставляется качество теста. Если процент неисправленных дефектов &lt; 10%, то качество Приемлемое (Acceptable), если &gt; 10%, то качество Неприемлемое (Unacceptable). При оценке качества функционала на уровне New Feature Test (полный тест нового функционала) качество отдельно проверенного функционала может быть: Высокое (High), Среднее (Medium), Низкое (Low).

Важно отдельно указывать информацию о качестве каждого модуля нового функционала с аргументацией выставленной оценки.

При оценке качества функционала на уровне Regression Testing нужно анализировать динамику изменения качества проверенной функциональности в сравнении с более ранними версиями сборки. Для этого приводится сравнительная характеристика каждой из частей функционала в сравнении с предыдущими версиями сборки, даются ясные пояснения о выставлении соответствующего качества каждой функции в отдельности. Также, как и у предыдущего вида тестов, качество этих может быть: Высокое (High), Среднее (Medium), Низкое (Low).