**1)** **Тестирование: понятие, критерии. Требования к тестированию. Процесс проведения отладки.**

Тестирование: Это процесс проверки программы на ошибки с целью обеспечения её корректной работы.

Критерии тестирования:Полнота, эффективность, воспроизводимость, репрезентативность, автоматизируемость, интуитивность.

Требования к тестированию: Полнота покрытия, корректность, управление временем и ресурсами, отслеживаемость, актуальность.

Процесс отладки: Воспроизведение ошибки, изучение кода, выявление ошибки, исправление, повторное тестирование, документирование.

**2) Цели и основные принципы тестирования.**

Цели тестирования: Выявление дефектов, обеспечение качества, уверенность в работе, повышение удовлетворенности клиентов.

Основные принципы тестирования: Раннее начало, исчерпывающее покрытие, автоматизация, тестирование ошибок, приоритизация, непрерывность, отчетность.

**3) Стадии разработки программного обеспечения: описание.**

1. Сбор требований: Определение и документирование требований к продукту.

2. Проектирование:Создание архитектуры системы и проектирование компонентов.

3. Разработка: Написание кода программы в соответствии с требованиями.

4. Тестирование: Проверка качества программы на наличие ошибок и дефектов.

5. Внедрение: Развертывание и внедрение программы в среду заказчика.

6.Сопровождение: Поддержка, исправление ошибок и обновление программы после выпуска.

**4)** **Подходы к формированию тестовых наборов: описание, примеры.**

1.Метод эквивалентных классов: Разбиение входных данных на классы с одинаковым поведением.

2. Метод граничных значений: Тестирование граничных и близлежащих к ним значений.

3. Метод попарного тестирования: Проверка всех возможных комбинаций пар параметров.

4. Метод тестирования по принципу ошибок: Создание тестов на основе интуиции и опыта.

5. Метод тестирования по структуре: Тестирование внутренней структуры программного продукта.

6. Метод случайного тестирования: Создание тестов без четких критериев.

**5)** **Среда тестирования: понятие, настройка. Документированность процесса тестирования: тестовый план и отчёт.**

Среда тестирования: Это набор аппаратных и программных компонентов для проведения тестирования ПО.

Настройка среды тестирования: Включает установку и конфигурацию необходимых инструментов и данных для тестирования.

Документированность процесса тестирования:

- Тестовый план: Описывает подход к тестированию, цели, расписание, используемые ресурсы и критерии приемки.

- Отчет о тестировании: Содержит результаты тестов, найденные дефекты, статистику и анализ эффективности тестирования.

**6)** **Методология тестирования сложных систем. Понятие сложной системы.**

Сложная система - это система с множеством взаимосвязанных элементов, обладающих эмерджентными свойствами, которые не могут быть объяснены только суммой свойств отдельных элементов.

Методология тестирования сложных систем:

1. Системный подход: Тестирование рассматривается как часть общего процесса разработки, учитывая взаимодействие с другими процессами.

2. Архитектурное тестирование: Включает тестирование модулей, подсистем и интеграцию между ними.

3. Тестирование производительности: Важно проверять производительность и масштабируемость системы.

4. Тестирование в реальном времени: Проверка поведения системы в реальном времени или при взаимодействии с окружающей средой.

5. Тестирование безопасности: Обязательно проверять систему на устойчивость к атакам и уязвимости.

6. Тестирование в условиях неопределенности: Тестирование должно быть адаптивным и учитывать возможные изменения в процессе разработки и эксплуатации.

**7)** **Понятие жизненного цикла программного обеспечения. Стандарты и их классификация.**

Понятие жизненного цикла программного обеспечения (ЖЦ ПО):

Это последовательность этапов, через которые проходит программное обеспечение от создания до вывода из эксплуатации.

Стандарты и их классификация:

Стандарты в области ЖЦ ПО включают ISO/IEC 12207, ISO/IEC 15288, IEEE 12207 и CMMI. Они описывают процессы и практики управления разработкой и качеством программного обеспечения.

**8)** **Каскадная модель жизненного цикла. Последовательность этапов. Особенности разработки по каскадной модели.**

Каскадная модель жизненного цикла:

- Последовательность этапов:

1. Анализ требований

2. Проектирование

3. Реализация

4. Тестирование

5. Внедрение

6. Сопровождение

Особенности разработки:

- Линейный характер:

- Жесткая фиксация требований:

- Отсутствие итераций:

- Недостаточная гибкость:

- Подходит для стабильных требований:

- Высокие требования к документации:

**9)** **Спиральная модель жизненного цикла. Особенности и проблемы разработки по спиральной модели.**

Спиральная модель жизненного цикла:

Особенности:

- Итерационный подход с оценкой рисков.

- Гибкость в изменении требований.

- Прозрачность и обратная связь.

Проблемы:

- Сложное управление и ресурсозатратность.

- Необходимость экспертизы для управления рисками.

- Риск зацикливания и потери времени.

**10)** **Каскадная и спиральная модель: сходства и различия.**

Сходства:

- Обе модели управляют жизненным циклом программного обеспечения.

- Имеют последовательные этапы разработки.

- Цель - разработать качественное программное обеспечение.

- Подходят для проектов с четко определенными требованиями.

Различия:

- Каскадная - линейная, спиральная - итеративная.

- Каскадная модель не учитывает риски, в то время как спиральная - ориентирована на их управление.

- Каскадная менее гибкая, изменения дорогостоящи и затруднительны.

- Спиральная модель гибкая, позволяет наиболее эффективно реагировать на изменения требований.

- Управление проектом в каскадной - линейно, в спиральной - итеративно с оценкой рисков.

- Каскадная часто используется в проектах с жесткими требованиями, спиральная - в проектах с изменяющимися требованиями и управлением рисками.

**11)** **Характеристики программного продукта. Этапы решения задач. Понятие алгоритма и его свойства.**

Характеристики программного продукта:

1. Функциональность: Способность выполнения задач.

2. Надежность: Устойчивость к сбоям и ошибкам.

3. Эффективность: Экономичное использование ресурсов.

4. Удобство использования: Простота и удобство интерфейса.

5. Поддерживаемость: Возможность обслуживания и модификации.

6. Портируемость: Способность к работе на разных платформах.

Этапы решения задач:

1. Понимание проблемы

2. Проектирование

3. Реализация

4. Тестирование

5. Внедрение

6. Сопровождение

Алгоритм:

Последовательность шагов для решения задачи.

- Свойства:

1. Понятность

2. Определенность

3. Эффективность

4. Верифицируемость

5. Универсальность

6. Ограниченность

**12)** **Классификация видов тестирования.**

Классификация видов тестирования:

1. Функциональное тестирование: Проверка функций программы.

2. Нефункциональное тестирование: Тестирование аспектов, как производительность, безопасность.

3. Модульное тестирование: Проверка отдельных модулей.

4. Дымовое тестирование: Проверка основных функций программы.

5. Санитарное тестирование: Проверка стабильности программы.

6. Интеграционное тестирование: Проверка взаимодействия компонентов.

7. Тестирование методом "черного ящика": Тестирование по спецификации.

8. Тестирование методом "белого ящика": Тестирование с знанием внутренней реализации.

9. Тестирование методом "серого ящика": Комбинация методов "черного" и "белого" ящиков.

10. Регрессионное тестирование: Проверка на наличие новых ошибок после изменений.

11. Юзабилити тестирование: Оценка удобства использования интерфейса.

12. Автоматизированное тестирование: Использование специальных инструментов.

13. Ручное тестирование: Проверка программы вручную.

14. Экспертное тестирование: Проведение специалистами.

15. Пользовательское тестирование: Тестирование конечными пользователями.

**13)** **Функциональное тестирование. Понятие, основные функции, условия ошибки, режим выполнения. Примеры функционального тестирования.**

Функциональное тестирование:

Понятие: Проверка соответствия программы функциональным требованиям.

Основные функции:

1. Проверка функциональности

2. Проверка корректности

3. Проверка пользовательского взаимодействия

Условия ошибки:

1. Неверный результат

2. Непредусмотренное поведение

3. Отсутствие функциональности

Примеры:

1. Регистрация пользователя

2. Поиск в веб-приложении

3. Оформление заказа в интернет-магазине

**14)** **Нефункциональное тестирование. Понятие, основные функции, условия ошибки, режим выполнения. Примеры нефункционального тестирования.**

Нефункциональное тестирование:

Понятие: Проверка аспектов программы, кроме её функциональности, например, производительности и безопасности.

Функции:

1. Производительность: Оценка скорости работы программы.

2. Безопасность: Проверка защищенности от атак.

3. Совместимость: Проверка работы на разных платформах.

4. Надежность: Оценка стабильности.

5. Удобство использования: Оценка удобства интерфейса.

Условия ошибки:

1. Медленная работа.

2. Уязвимости.

3. Несовместимость.

4. Частые сбои.

5. Неудобный интерфейс.

Примеры:

1. Тестирование производительности веб-страниц.

2. Проверка безопасности веб-приложения.

3. Тестирование совместимости мобильного приложения.

4. Оценка надежности сервера.

5. Тестирование удобства использования интерфейса.

**15)** **Модульное тестирование. Понятие, основные методы, алгоритм выполнения, преимущества и недостатки.**

Модульное тестирование:

Понятие: Проверка отдельных модулей программы в изоляции.

Методы:

1. Черный ящик: Тестируются входные и выходные данные.

2. Белый ящик: Тестируются внутренние структуры и логика модуля.

3. Серый ящик: Комбинированный метод.

Алгоритм:

1. Выбор модуля.

2. Разработка тестовых случаев.

3. Запуск тестов.

4. Анализ результатов.

5. Фиксация ошибок.

Преимущества:

- Раннее обнаружение ошибок.

- Легкая локализация ошибок.

- Возможность автоматизации.

Недостатки:

- Ограниченность в оценке взаимодействия модулей.

- Недостаточное покрытие системы.

- Затраты на поддержку.

**16) Дымовое тестирование. Понятие, основные методы, алгоритм выполнения, преимущества, примеры дымового тестирования, отличия от санитарного тестирования.**

Дымовое тестирование - проверка работы системы без углубления в её детали. Основные методы: генерация входных данных, запуск тестовых сценариев, анализ реакции системы. Преимущества: обнаружение скрытых дефектов, повышение уверенности в качестве продукта. Примеры: тестирование взаимодействия компонентов, проверка совместимости. Отличие от санитарного тестирования: фокус на системе в целом, не на отдельных компонентах.

**17) Санитарное тестирование. Понятие, основные методы, алгоритм выполнения, преимущества, примеры санитарного тестирования, отличия от дымового тестирования.**

Санитарное тестирование - проверка отдельных компонентов или функций системы.

Основные методы: тестирование интерфейсов, модульное тестирование, функциональное тестирование.

Алгоритм выполнения:

1. Определение целей тестирования.

2. Создание тестовых случаев для каждого компонента или функции.

3. Запуск тестов и анализ результатов.

Преимущества: Обнаружение дефектов в отдельных частях системы, более простая настройка и выполнение тестов.

Примеры санитарного тестирования: Проверка работы отдельных функций приложения, тестирование API.

Отличие от дымового тестирования: Фокус на отдельных компонентах или функциях, а не на системе в целом.

**18) Интеграционное тестирование. Понятие, необходимость интеграционного тестирования, основные методы, критерии входа и выхода для интеграционного тестирования.**

Интеграционное тестирование - проверка взаимодействия компонентов ПО. Необходимо для обнаружения ошибок в их интеграции.

Основные методы: "сверху вниз" и "снизу вверх".

Критерии входа: Готовность компонентов к интеграции, наличие интерфейсных спецификаций.

Критерии выхода: Успешная интеграция всех компонентов, отсутствие критических ошибок, соответствие ожидаемому поведению системы.

**19) Тестирование методом «черного ящика». Типы и область тестирования, примеры.**

Тестирование методом "черного ящика" основано на проверке функциональности программы без знания внутренней реализации. Типы включают функциональное, пользовательское, атрибутное и др. Область тестирования включает в себя входные данные, выходные данные, интерфейсы, исключения и др. Примеры включают тестирование графического интерфейса, тестирование API, тестирование пользовательского опыта и др.

**20)** **Тестирование методом «белого ящика». Типы и область тестирования, примеры. Преимущества и недостатки метода.**

Тестирование "белого ящика" проверяет внутреннюю структуру программы. Примеры: тестирование блоков кода, интеграции компонентов. Преимущества: обнаружение ошибок на ранних этапах, повышенная точность. Недостатки: требует доступа к коду, может быть сложным и не обнаруживает некоторые типы ошибок.

**21)** **Тестирование методом «серого ящика». Стратегия, преимущества и проблемы тестирования. Область тестирования. Этапы выполнения тестирования.**

Тестирование методом "серого ящика":

- Стратегия: Частичное знание внутренней структуры программы.

- Преимущества: Объединяет преимущества тестирования "белого" и "черного ящиков".

- Проблемы: Требует более сложной подготовки тестов.

- Область тестирования: Включает проверку интерфейсов, алгоритмов и обработки данных программы.

- Этапы выполнения: Подготовка, выполнение, анализ, отчетность.

**22)** **Регрессивное тестирование. Понятие, необходимость, методы проведения регрессивного тестирования. Выбор регрессивного теста и тестовых случаев. Различия между повторным тестированием и регрессивным. Проблемы регрессивного тестирования.**

Регрессивное тестирование:

- Понятие: Повторное тестирование после изменений для обнаружения новых ошибок.

- Необходимость: Гарантия работоспособности программы после изменений.

- Методы: Автоматизация, выбор ключевых тестов.

- Выбор тестов: Основан на критических функциях и предыдущих ошибках.

- Различия: Повторное - без изменений, регрессивное - после изменений.

- Проблемы: Недостаточное покрытие, затраты времени и ресурсов.

**23)** **Определение и цель ручного тестирования. Преимущества и недостатки ручного тестирования. Типы тестирования.**

Ручное тестирование:

- Определение и цель: Выполнение тестов вручную для проверки функциональности и качества программы.

- Преимущества: Гибкость, возможность обнаружения неожиданных проблем.

- Недостатки: Трудоемкость, возможность человеческих ошибок.

- Типы тестирования: Функциональное, пользовательское, регрессионное, нагрузочное

**24)** **Интеграционное тестирование. Понятие, цель и объект тестирования. Методы и инструменты тестирования.**

Интеграционное тестирование:

- Понятие и цель: Проверка взаимодействия компонентов после их объединения. Цель - обнаружение ошибок в интеграции и проверка работы системы.

- Объект тестирования: Взаимодействие компонентов или модулей программы.

- Методы и инструменты: Тестирование "сверху вниз" и "снизу вверх", использование автоматизации (JUnit, TestNG, Selenium).

**25)** **Юзабилити тестирование: понятие, основные подходы планирования и разработки программных продуктов.**

Юзабилити тестирование:

- Понятие: Проверка удобства использования программного продукта.

- Подходы:

1. Исследовательский: Изучение потребностей пользователей и конкурентов.

2. Проектирование с учетом пользователя: Создание удобного интерфейса.

3. Тестирование с участием пользователей: Проверка на ранних этапах итераций.

4. Обратная связь и улучшение: Использование обратной связи для постоянного совершенствования.

**26)** **Экспертный подход к юзабилити-тестированию: критерии. Преимущества и недостатки подхода.**

Экспертный подход к юзабилити-тестированию:

Критерии:

- Экспертное мнение специалистов по юзабилити.

- Соблюдение стандартов и принципов дизайна интерфейса.

- Оценка удобства использования с точки зрения опыта и знаний экспертов.

Преимущества:

- Быстрота и эффективность оценки.

- Меньшая стоимость по сравнению с тестированием с участием пользователей.

- Возможность выявления проблем на ранних стадиях разработки.

Недостатки:

- Отсутствие реальной обратной связи от конечных пользователей.

- Возможность пропуска некоторых проблем, которые могут быть замечены только пользователями.

- Ограниченная перспектива экспертов может привести к упущению некоторых аспектов.

**27)** **Пользовательский подход к юзабилити-тестированию: критерии, методы измерения. Преимущества и недостатки подхода**

Пользовательский подход к юзабилити-тестированию:

Критерии:

- Задачи пользователей.

- Эффективность выполнения задач.

- Удовлетворенность пользователей.

Методы измерения:

- Тестирование сценариев использования.

- Сбор обратной связи.

- Использование метрик, например, время выполнения задач и количество ошибок.

Преимущества:

- Реальная обратная связь от пользователей.

- Обнаружение реальных проблем и потребностей пользователей.

- Улучшение удовлетворенности и лояльности пользователей.

Недостатки:

- Затраты времени и ресурсов.

- Не всегда представительная выборка пользователей.

- Невозможность выявить все проблемы на ранних этапах.

**28)** **Автоматизация тестирования: понятие, типы, принципы. Требования к тестировщику. Преимущества и недостатки автоматизации тестирования.**

Автоматизация тестирования:

- Понятие: Использование программных инструментов для выполнения тестовых сценариев без участия человека.

- Типы: Функциональное, регрессионное, нагрузочное тестирование и другие.

- Принципы: Повторяемость, надежность, эффективность.

- Требования к тестировщику: Понимание основ тестирования, навыки программирования, разработка эффективных тестовых сценариев, аналитические и коммуникационные навыки.

- Преимущества: Увеличение скорости, повышение точности, экономия времени и ресурсов.

- Недостатки: Затраты на разработку и поддержку, невозможность полного замещения ручного тестирования, ограничения в тестировании некоторых аспектов.

**29)** **Понятие утверждений и их параметры. Модели и основные виды утверждений. Понятие и категории директив.**

Утверждения:

- Понятие: Это проверяемые высказывания в тестировании ПО.

- Параметры: Ожидаемое и фактическое значения, точность сравнения, состояние системы.

Модели и основные виды:

- Модели: "Given-When-Then", "Arrange-Act-Assert".

- Виды: Сравнение значений, проверка состояния, сравнение структуры данных и другие.

Директивы:

- Понятие: Инструкции для тестирования.

- Категории: Описательные, действенные, ожидаемые результаты.

**30)** **Определение, цель и этапы тестирования производительности. Критерии для измеряемых метрик.**

Тестирование производительности:

- Определение: Оценка производительности программного продукта.

- Цель: Определить скорость и эффективность системы.

- Этапы: Планирование, создание тестов, запуск и анализ.

- Критерии: Время отклика, пропускная способность, загрузка CPU и памяти, использование сети, количество пользователей.  
**31)** **Структура и виды CASE-средств. Критерии при выборе.**

CASE-средства (Computer-Aided Software Engineering) - это инструменты, помогающие автоматизировать процессы разработки программного обеспечения.

Структура CASE-средств может включать:

* Инструменты для моделирования и проектирования.
* Инструменты для генерации кода.
* Инструменты для управления проектом.
* Инструменты для анализа и тестирования.

Критерии при выборе CASE-средства:

* Функциональность.
* Интеграция.
* Простота использования.
* Поддержка.
* Совместимость.
* Стоимость.

Выбор CASE-средства зависит от конкретных потребностей и характеристик проекта.

**32)** **Структуры среды разработки. Факторы при выборе.**Структура среды разработки включает:

* Текстовый редактор.
* Компилятор/Интерпретатор.
* Отладчик.
* Средства управления версиями.
* Инструменты сборки.

Факторы при выборе среды разработки:

* Поддержка языков программирования.
* Функциональность.
* Интеграция с другими инструментами.
* Совместимость.
* Производительность.
* Поддержка и сообщество.
* Цена и лицензирование

*Основные возможности среды разработки включают:*

Текстовый редактор.

Компилятор/Интерпретатор.

Отладчик.

Управление версиями.

Инструменты сборки.

Автодополнение и подсказки.

Интеграция с внешними инструментами.

Анализ кода.

Графическое моделирование.

Тестирование.

**33) Характеристика основных возможности среды разработки.**

Основные возможности среды разработки включают текстовый редактор, компилятор/интерпретатор, отладчик, управление версиями, инструменты сборки, а также функции автодополнения, интеграции с внешними инструментами, анализа кода и тестирования.

**34) Основные инструменты среды для создания, исполнения и управления информационной системой.**

Основные инструменты среды разработки для создания, исполнения и управления информационной системой включают текстовый редактор для написания кода, компилятор/интерпретатор для преобразования кода в исполняемый формат, отладчик для поиска и исправления ошибок, инструменты управления версиями для контроля изменений, а также инструменты сборки для автоматизации процесса сборки программы.

**35) Средства обработки информации: определение, виды.**  
Средства обработки информации - это инструменты и технологии, используемые для сбора, хранения, обработки, анализа и передачи данных. Виды средств обработки информации включают программное обеспечение для баз данных, текстовых редакторов, электронных таблиц, систем управления контентом, систем управления проектами, а также аппаратное обеспечение, такое как компьютеры, серверы, сетевое оборудование и устройства хранения данных.

**36) Организация работы в команде разработчиков. Виды моделей и роли.**

Организация работы в команде разработчиков включает различные модели и роли:

Модели работы:

* Каскадная модель разработки.
* Итеративная модель разработки.
* Agile (гибкая) модель разработки.
* DevOps.

Роли в команде:

* Проектный менеджер.
* Аналитик.
* Разработчик.
* Тестировщик.
* Архитектор.
* Дизайнер пользовательского интерфейса (UI/UX).
* Системный администратор.
* Коммуникационный специалист (или DevOps инженер в DevOps модели).

**37) Система контроля версий: совместимость, установка, настройка.**

Система контроля версий (СКВ) позволяет отслеживать изменения в исходном коде и координировать работу разработчиков.

Совместимость: Ведущие СКВ, такие как Git, Subversion (SVN), Mercurial, совместимы с различными операционными системами (Windows, macOS, Linux) и популярными IDE.

Установка: Обычно системы контроля версий могут быть установлены через пакетные менеджеры (например, apt, yum, brew), скачав исполняемые файлы с официальных сайтов или установив через графические установщики.

Настройка: После установки, систему контроля версий следует настроить для работы с конкретным проектом. Это включает настройку имени пользователя и email (для Git), создание репозитория, настройку удаленных репозиториев (если требуется) и определение файлов, игнорируемых системой контроля версий.

**38) Характеристика методов обеспечения кроссплатформенности информационной системы.**

Методы обеспечения кроссплатформенности информационной системы включают:

* Использование кроссплатформенных языков программирования: Написание кода на языках, которые поддерживаются на различных платформах, таких как Java, Python, JavaScript.
* Использование кроссплатформенных фреймворков и библиотек: Использование фреймворков и библиотек, которые предоставляют абстракции для работы с операционной системой и обеспечивают кроссплатформенную совместимость, например, Qt, Electron.
* Виртуализация: Запуск приложения в виртуальной среде, которая может быть установлена на различных операционных системах.
* Облачные сервисы: Разработка и развертывание приложения в облаке, что позволяет обеспечить доступ к приложению через веб-браузер с любой платформы.
* Адаптивный дизайн: Разработка пользовательского интерфейса, который адаптируется к разным размерам экранов и устройствам, что позволяет обеспечить кроссплатформенную совместимость приложения.

**39) Особенности сервисно-ориентированной архитектуры, основные принципы.**

* Особенности сервисно-ориентированной архитектуры (SOA) и основные принципы:
* Модульность: Система разбивается на независимые сервисы, каждый из которых выполняет конкретную функцию.
* Интероперабельность: Сервисы могут взаимодействовать друг с другом независимо от используемых технологий и платформ.
* Переиспользование: Сервисы могут быть использованы многократно в различных контекстах и приложениях.
* Легкая интеграция: Сервисы могут быть легко интегрированы друг с другом и существующими системами благодаря использованию стандартизированных протоколов и интерфейсов.
* Автономность: Каждый сервис функционирует независимо и может быть развернут, масштабирован и обновлен отдельно от других сервисов.
* Отказоустойчивость: Сервисы должны быть способными обрабатывать ошибки и отказы, не приводящие к сбоям системы в целом.
* Гибкость и масштабируемость: Архитектура SOA позволяет гибко масштабировать и модифицировать систему путем добавления, изменения или удаления сервисов.
* Ориентация на бизнес-процессы: Сервисы ориентированы на предоставление функциональности, поддерживающей бизнес-процессы организации.
* Эти принципы позволяют создавать гибкие, модульные и легко масштабируемые информационные системы.

**40) Интегрированные среды разработки для создания независимых программ: назначение, сфера применения.**

Интегрированные среды разработки (IDE) предназначены для создания программного обеспечения, включая независимые программы. Они обеспечивают инструменты для написания, отладки, тестирования и сборки программ. Сфера применения IDE охватывает различные области, включая разработку приложений для настольных компьютеров, мобильных устройств, веб-приложений, игр и других программных продуктов.

**41) Особенности объектно-ориентированных языков программирования.**

* Особенности объектно-ориентированных языков программирования:
* Использование объектов: Программы строятся вокруг объектов, которые представляют сущности реального мира и обладают свойствами и методами.
* Инкапсуляция: Сокрытие внутренней реализации объекта и предоставление доступа к нему только через установленные интерфейсы.
* Наследование: Возможность создания новых классов на основе существующих с возможностью наследования и расширения функциональности.
* Полиморфизм: Возможность использования одного и того же метода для разных типов данных или объектов, что облегчает повторное использование кода и упрощает его понимание.
* Абстракция: Возможность определения общих характеристик объектов и их взаимодействия без уточнения конкретной реализации.
* Классы и объекты: Программы организуются вокруг классов, которые являются шаблонами для создания объектов с общими характеристиками.

Эти особенности позволяют создавать более гибкие, модульные и легко поддерживаемые программы.

**42) Диаграммы UML вариантов использования, последовательности, кооперации: цель построения и синтаксис.**

Диаграммы UML (Unified Modeling Language):

1. Диаграмма вариантов использования (Use Case):

* Цель: Показать функциональные требования системы и её взаимодействие с внешними акторами.
* Синтаксис: Включает акторов (людей, другие системы), варианты использования (use cases) и их связи.

1. Диаграмма последовательности (Sequence):

* Цель: Показать последовательность взаимодействия между объектами в определенном сценарии использования.
* Синтаксис: Представляет объекты в виде вертикальных линий, а сообщения между объектами в виде горизонтальных стрелок.

1. Диаграмма кооперации (Collaboration):

* Цель: Показать взаимодействие между объектами и их связи в системе.
* Синтаксис: Представляет объекты в виде прямоугольников с указанием их ролей и связи между объектами.

**43) Диаграммы UML развертывания, компонентов, потоков данных: цель построения и синтаксис.**

Диаграммы UML:

1. Диаграмма развертывания (Deployment):

* Цель: Показать физическую конфигурацию системы и размещение компонентов на аппаратном обеспечении.
* Синтаксис: Включает узлы (например, серверы), компоненты и связи между ними, а также артефакты (например, исполняемые файлы).

1. Диаграмма компонентов (Component):

* Цель: Показать структуру и зависимости компонентов системы.
* Синтаксис: Включает компоненты, интерфейсы, зависимости между компонентами и артефакты, которые реализуют компоненты.

1. Диаграмма потоков данных (Data Flow):

* Цель: Показать потоки данных и процессы, которые их обрабатывают в системе.
* Синтаксис: Включает процессы (функции, операции), потоки данных, хранилища данных и внешние сущности, а также связи между ними.

**44) Архитектура информационной системы: виды, типы групп, применение архитектурных описаний.**

Архитектура информационной системы (ИС):

1. Виды:

- Многоуровневая: Разделение системы на слои, каждый из которых отвечает за определенный функциональный аспект.

- Клиент-сервер: Разделение системы на клиентские и серверные компоненты, обеспечивающие взаимодействие между ними.

- Микросервисная: Разделение системы на небольшие, независимые сервисы, которые взаимодействуют между собой через сеть.

2. Типы групп:

- Логическая группировка: Связанные компоненты или функциональные модули, которые объединены по логическим критериям.

- Физическая группировка: Компоненты, размещенные на одном устройстве или в одной локации.

3. Применение архитектурных описаний:

- Планирование и проектирование: Описание структуры и взаимодействия компонентов системы.

- Документация: Создание документации для разработчиков, администраторов и пользователей.

- Анализ и моделирование: Использование описаний для анализа производительности, безопасности и других аспектов системы.

- Управление изменениями: Использование архитектурных описаний для оценки влияния изменений в системе.

**45) Определение конфигурации информационной системы. Этапы, методы и средства конфигурирования.**

Конфигурация информационной системы (ИС) - это процесс управления и контроля изменений в компонентах ИС, включая программное обеспечение, аппаратное обеспечение, настройки и документацию.

Этапы конфигурирования:

1. Идентификация компонентов: Определение всех элементов, входящих в состав ИС.

2. Контроль версий: Управление изменениями компонентов и документации.

3. Конфигурационное управление: Определение и управление зависимостями между компонентами.

4. Тестирование и верификация: Проверка корректности и работоспособности ИС после изменений.

5. Учет и аудит: Отслеживание всех изменений и обеспечение соответствия стандартам безопасности и требованиям.

Методы и средства конфигурирования:

1. Системы управления конфигурацией (СКУК): Например, Git, SVN, Mercurial.

2. Инструменты автоматизации сборки и развертывания: Например, Jenkins, Ansible.

3. Инструменты отслеживания ошибок и задач: Например, Jira, Redmine.

4. Инструменты документирования: Например, Wiki, Confluence.

5. Инструменты тестирования: Например, JUnit, Selenium.

Эффективное конфигурирование помогает обеспечить стабильность и надежность работы информационной системы.

**46)Критерии выбора технических средств проекта информационной системы:**

**Требования проекта.**

Совместимость с существующей инфраструктурой.

Производительность.

Безопасность.

**47)Репозиторий проекта информационной системы:**

Понятие: это централизованное хранилище для всех файлов проекта, включая исходный код, документацию и тесты.

Процесс создания включает создание репозитория, добавление файлов, управление доступом и резервное копирование.

**48)Алгоритм определения уровня доступа к репозиторию в системе контроля версий:**

Определение прав доступа на основе ролей и обязанностей пользователей.

Установка доступа к определенным веткам или файлам.

Использование аутентификации и авторизации.

**49)Процесс настройки среды разработки информационной системы:**

Установка предпочитаемых инструментов и языков кода, конфигурация среды разработки, подключение к репозиторию.

**50)Инструменты мониторинга разработки проекта:**

Отслеживание прогресса задач, управление версиями, отладка и профилирование кода.

**51)Требования к пользовательскому интерфейсу информационной системы:**

Удобство использования, соответствие корпоративному стилю, адаптивность под разные устройства.

**52)Принципы создания графического пользовательского интерфейса (GUI):**

Простота, согласованность, интуитивность, эффективное использование места на экране.

**53)Понятие и формы спецификации языка программирования для разработки информационной системы:**

Спецификация - это описание синтаксиса и семантики языка.

Формы: стандарты языка, документация, справочные материалы.

**54)Методы организации ввода-вывода информации в информационной системе:**

Формы и диалоговые окна, командная строка, веб-интерфейс, API.

**55)Спецификация настроек типовой информационной системы:**

Описание параметров и настроек системы для ее корректной работы.

**56)Сообщения между модулями:**

Назначение: обмен информацией для выполнения задач.

Вход: данные для обработки.

Разделы модуля: функциональные блоки.

**57)Приложения для моделирования процессов:**

Виды моделей: процессные диаграммы, BPMN, UML и др.

Требования: визуальное представление, анализ и оптимизация процессов.

**58)Разработка приложений:**

Виды: десктопные, веб, мобильные.

Этапы разработки.

Этапы разработки могут включать в себя:

Анализ требований: сбор и анализ требований к приложению.

Проектирование: разработка архитектуры, дизайна и интерфейса приложения.

Реализация: написание и тестирование кода.

Тестирование: проверка функциональности, исправление ошибок.

Внедрение: установка и запуск приложения в рабочей среде.

Сопровождение: поддержка и обновление приложения после внедрения.

**59)Отладка приложения:**

Идентификация и исправление ошибок в коде.

Методы: отладчик, логирование, тестирование.

**60)Интеграция модуля:**

Определение: это процесс объединения отдельных модулей или компонентов в единую систему с целью обеспечения их взаимодействия и совместной работы.

Возможности интеграции:

Интеграция на уровне данных: обмен информацией между модулями через общую базу данных или API.

Интеграция на уровне интерфейса: взаимодействие модулей через стандартизированные интерфейсы и протоколы обмена данными.

Интеграция на уровне бизнес-логики: согласование и совмещение бизнес-процессов и правил, реализованных в различных модулях.

Способы интеграции:

API (предоставление набора функций и методов для взаимодействия с модулем. HTTP, REST, SOAP и т.д.