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

**Тестирование** — это процесс исследования программного обеспечения с целью выявления ошибок, проверки соответствия требованиям и оценки качества продукта.  
**Критерии тестирования** включают:

* Полноту тестового покрытия
* Репродуцируемость
* Отсутствие двусмысленности
* Автоматизируемость

**Требования к тестированию:**

* Четкость тест-кейсов
* Использование актуальных данных
* Возможность повторного использования

**Процесс отладки** включает локализацию ошибки, анализ кода, исправление и регрессионное тестирование.

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

**Цели:**

* Выявление дефектов
* Предотвращение ошибок
* Повышение качества ПО
* Обеспечение соответствия требованиям

**Принципы тестирования (ISTQB):**

1. Тестирование показывает наличие дефектов, но не их отсутствие
2. Исчерпывающее тестирование невозможно
3. Раннее тестирование снижает стоимость исправлений
4. Сгущение дефектов (80/20)
5. Парадокс пестицида
6. Тестирование зависит от контекста
7. Заблуждение об отсутствии ошибок

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

 Анализ **требований**

* Определение целей системы.
* Сбор и документирование требований (функциональных и нефункциональных).
* Создание спецификаций (SRS – Software Requirements Specification).

 Проектирование **(Design)**

* Архитектурное проектирование системы.
* Разработка технических решений и схемы взаимодействия компонентов.
* Выбор технологий, инструментов, структур данных и алгоритмов.

 Разработка **(Coding, Implementation)**

* Написание программного кода на выбранном языке программирования.
* Интеграция компонентов системы.
* Версионный контроль и управление кодом.

 Тестирование **(Testing)**

* Юнит-тестирование, интеграционное тестирование, системное тестирование.
* Поиск и исправление багов (debugging).
* Регрессионное тестирование.

 Развертывание **(Deployment)**

* Установка программного обеспечения в продакшен-среде.
* Проведение финального тестирования на реальных данных.
* Мониторинг и поддержка.

 Сопровождение **и обслуживание (Maintenance & Support)**

* Обновления и патчи безопасности.
* Поддержка пользователей.
* Доработка функционала на основе отзывов.

### 4. Подходы к формированию тестовых наборов: описание, примеры.

#### **1. Эквивалентное разбиение (Equivalence Partitioning)**

* Входные данные делятся на классы эквивалентности.
* Пример: если ввод допускает числа 1-100, то тестируется одно число из каждого диапазона (1-49, 50, 51-100).

#### **2. Анализ граничных значений (Boundary Value Analysis, BVA)**

* Проверка границ допустимых значений.
* Пример: если поле принимает от 1 до 100, то тестируется 0, 1, 100, 101.

#### **3. Таблицы принятия решений (Decision Table Testing)**

* Используется для проверки сложной логики решений.
* Пример: система скидок в интернет-магазине при разных условиях (купон + сумма заказа + статус пользователя).

#### **4. Причина-следствие (Cause-Effect Graphing)**

* Построение графа зависимостей входных условий и ожидаемых результатов.

#### **5. Исследовательское тестирование (Exploratory Testing)**

* Используется, когда требования плохо задокументированы.
* Пример: тестировщик исследует поведение системы, находя неожиданные сценарии.

### 5. Среда тестирования: понятие, настройка. Документированность процесса тестирования: тестовый план и отчет.

**Среда тестирования** — это аппаратное и программное обеспечение, необходимое для тестов.  
Настройка включает установку ПО, конфигурацию серверов, настройку БД.

**Тестовый план** — документ, описывающий цели, объем тестирования, критерии завершения.  
**Тестовый отчет** — содержит результаты тестирования, обнаруженные дефекты и рекомендации.

### 6. Методология тестирования сложных систем. Понятие сложной системы.

**Сложная система** — это система с высокой взаимосвязанностью модулей, сложной логикой и многокомпонентной архитектурой.  
Методы тестирования:

* **Модульное тестирование**
* **Интеграционное тестирование**
* **Нагрузочное тестирование**
* **Автоматизация тестирования**

### 7. Понятие жизненного цикла программного обеспечения. Стандарты и их классификация.

Жизненный цикл ПО (SDLC) — это процесс разработки от идеи до поддержки.  
**Стандарты:**

* **ISO/IEC 9126** — качество ПО
* **ISO/IEC 25010** — характеристики качества
* **ISTQB** — стандарты тестирования

### 8. Каскадная модель жизненного цикла. Особенности разработки по каскадной модели.

**Каскадная модель (Waterfall)** — последовательная разработка, где этапы идут строго один за другим.  
**Особенности:**

* Четко определенные фазы
* Минимизация изменений
* Долгий цикл разработки
* Высокая формализация документации

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

**Спиральная модель жизненного цикла программного обеспечения (ПО) — это процесс разработки, сочетающий в себе как итеративность, так и этапность. Циклический подход к разработке, в котором процесс проходит через несколько итераций, каждая из которых завершается оценкой рисков и принятием решения о дальнейшей работе.**

**Особенности:**

* Итерационный процесс
* Анализ рисков перед каждой итерацией
* Подходит для сложных проектов  
  **Проблемы:**
* Высокие затраты
* Сложность управления

### 10. Каскадная и спиральная модель: сходства и различия.

**Сходства:**

* Фазовый подход
* Документированность

**Различия:**

| **Каскадная** | **Спиральная** |
| --- | --- |
| Линейная | Итерационная |
| Минимум изменений | Гибкость |
| Долгий цикл | Быстрая адаптация |

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

#### **Характеристики программного продукта**

Ключевые характеристики:

* **Функциональность** – выполняет ли ПО свои функции.
* **Надежность** – устойчивость к сбоям.
* **Производительность** – скорость и эффективность работы.
* **Юзабилити** – удобство для пользователя.
* **Сопровождаемость** – легкость модификации.

#### **Этапы решения задач**

1. Постановка задачи.
2. Анализ требований и формулировка условий.
3. Разработка алгоритма.
4. Кодирование (программирование).
5. Тестирование.
6. Внедрение и эксплуатация.

#### **Понятие алгоритма и его свойства**

**Алгоритм** – конечная последовательность шагов для решения задачи.

Основные свойства:

* **Дискретность** – шаги четко определены.
* **Определенность** – нет двусмысленностей.
* **Конечность** – завершается за конечное число шагов.
* **Результативность** – дает нужный результат.
* **Массовость** – применим к множеству задач одного типа.

### 12. Классификация видов тестирования.

 По **уровню тестирования**

* **Модульное (Unit Testing)** – проверка отдельных функций и модулей.
* **Интеграционное (Integration Testing)** – взаимодействие модулей.
* **Системное (System Testing)** – полное тестирование всей системы.
* **Приемочное (Acceptance Testing)** – соответствие требованиям заказчика.

 По **методу тестирования**

* **Белый ящик** – тестирование с доступом к коду.
* **Черный ящик** – проверка функционала без знания кода.
* **Серый ящик** – частичный доступ к внутренней логике.

 По **степени автоматизации**

* **Ручное (Manual Testing)**.
* **Автоматизированное (Automated Testing)**.

 По **цели тестирования**

* **Функциональное тестирование** – проверка соответствия требованиям.
* **Нагрузочное тестирование** – проверка работы под нагрузкой.
* **Стресс-тестирование** – поведение при экстремальных условиях.
* **Регрессионное тестирование** – проверка после внесения изменений.

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

#### **Понятие:**

Функциональное тестирование (Functional Testing) — это вид тестирования ПО, направленный на проверку соответствия функциональности системы требованиям, указанным в спецификации.

#### **Основные функции:**

* Проверка корректности работы функций приложения
* Оценка соответствия требованиям заказчика
* Проверка работы пользовательского интерфейса
* Обнаружение дефектов, связанных с логикой работы системы

#### **Условия ошибки:**

* Некорректные или отсутствующие функциональные элементы
* Несоответствие требованиям спецификации
* Ошибки при вводе и обработке данных

#### **Режим выполнения:**

Может выполняться вручную (manual) или автоматически (automated).

#### **Примеры функционального тестирования:**

* Проверка входа в систему
* Тестирование корзины интернет-магазина
* Проверка отправки формы на сайте

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

#### **Понятие:**

Нефункциональное тестирование (Non-Functional Testing) оценивает характеристики ПО, не связанные напрямую с его функциональностью, такие как производительность, безопасность, удобство использования и т. д.

#### **Основные функции:**

* Проверка скорости работы системы
* Оценка удобства использования интерфейса
* Анализ защищенности данных

#### **Условия ошибки:**

* Длительное время отклика системы
* Уязвимости, позволяющие несанкционированный доступ
* Сложность навигации по интерфейсу

#### **Режим выполнения:**

Чаще всего автоматизированный, особенно для нагрузочного и стресс-тестирования.

#### **Примеры нефункционального тестирования:**

* Тестирование скорости загрузки веб-сайта
* Анализ безопасности платежной системы
* Тестирование удобства интерфейса мобильного приложения

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

#### **Понятие:**

Модульное тестирование (Unit Testing) проверяет отдельные модули (функции, классы, методы) программы изолированно от других компонентов.

#### **Основные методы:**

* **White-box testing** (тестирование с доступом к коду)
* **Mocking** (использование заглушек для внешних зависимостей)

#### **Алгоритм выполнения:**

1. Выбор модуля для тестирования
2. Создание тестовых случаев
3. Запуск тестов
4. Анализ результатов

#### **Преимущества:**

* Раннее выявление ошибок
* Облегчение рефакторинга кода
* Автоматизация тестирования

#### **Недостатки:**

* Высокие затраты на написание тестов
* Ограниченность в проверке взаимодействий между модулями

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

#### **Понятие:**

Дымовое тестирование (Smoke Testing) — это первичная проверка работоспособности основных функций приложения после сборки новой версии ПО.

#### **Основные методы:**

* Ручное тестирование
* Автоматизированное тестирование (с помощью скриптов)

#### **Алгоритм выполнения:**

1. Сборка новой версии
2. Запуск базовых тестов
3. Оценка работоспособности системы
4. Принятие решения о дальнейшем тестировании

#### **Преимущества:**

* Быстрое выявление критических ошибок
* Минимальные затраты на тестирование
* Помогает определить готовность к более глубокому тестированию

#### **Примеры дымового тестирования:**

* Проверка запуска приложения
* Проверка входа в систему
* Проверка загрузки главной страницы

#### **Отличия от санитарного тестирования:**

* Дымовое тестирование охватывает всю систему, а санитарное — только исправленные баги
* Дымовое тестирование проводится перед детальными проверками, санитарное — после фиксов

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

#### **Понятие:**

Санитарное тестирование (Sanity Testing) проверяет исправленные баги и их влияние на другие части системы.

#### **Основные методы:**

* Локальное тестирование исправленных дефектов
* Частичное регрессионное тестирование

#### **Алгоритм выполнения:**

1. Определение исправленных ошибок
2. Создание тест-кейсов
3. Запуск тестов
4. Анализ влияния исправлений

#### **Преимущества:**

* Ускоряет процесс тестирования
* Позволяет убедиться в исправлении ошибок
* Экономит ресурсы перед полным регрессионным тестированием

#### **Примеры санитарного тестирования:**

* Проверка исправленного бага с авторизацией
* Тестирование исправленной функции расчета цен

#### **Отличия от дымового тестирования:**

* Санитарное тестирование направлено на проверку конкретных исправлений, а дымовое — на базовую работоспособность системы
* Санитарное тестирование проводится после исправления багов, а дымовое — перед основными тестами.

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

#### **Понятие:**

Интеграционное тестирование (Integration Testing) проверяет взаимодействие между модулями системы.

#### **Необходимость интеграционного тестирования:**

* Выявление проблем в коммуникации между модулями
* Обеспечение корректной работы системы в целом

#### **Основные методы:**

* **Big Bang** (все модули тестируются одновременно)
* **Incremental** (модули тестируются поэтапно)
* **Top-Down** (тестирование от верхнего уровня системы)
* **Bottom-Up** (тестирование с нижнего уровня)

#### **Критерии входа и выхода:**

**Вход:**

* Завершено модульное тестирование
* Подготовлена тестовая среда

**Выход:**

* Все интерфейсы между модулями протестированы
* Все критические ошибки устранены

**19–21. Тестирование «черного», «белого» и «серого ящика».**

| **Метод** | **Описание** | **Примеры** |
| --- | --- | --- |
| Черный ящик | Без знания кода | UI-тестирование |
| Белый ящик | С анализом кода | Unit-тесты |
| Серый ящик | Комбинация методов | API-тестирование |

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

**Понятие**: Регрессивное тестирование (Regression Testing) – это повторное тестирование уже протестированной системы после внесения изменений (исправления багов, обновления кода), чтобы убедиться, что новые изменения не привели к сбоям в существующем функционале.

**Необходимость**: Любое изменение в коде может повлиять на другие части системы. Регрессивное тестирование необходимо, чтобы предотвратить появление новых дефектов после обновлений.

**Методы**:

* **Полное регрессивное тестирование** – все тест-кейсы проверяются заново.
* **Частичное регрессивное тестирование** – проверяются только критически важные модули.
* **Селективное регрессивное тестирование** – выполняются только измененные тест-кейсы.
* **Автоматизированное регрессивное тестирование** – используется при частых релизах для ускорения тестирования.

**Выбор тест-кейсов**:

* Критически важные функции.
* Сценарии, связанные с измененными модулями.
* Часто используемые пользователями функции.

**Различие между повторным тестированием и регрессивным**:

* **Повторное тестирование (Re-testing)** – проверка исправленного бага на том же тесте.
* **Регрессивное тестирование** – проверка, что исправление не повлияло на другие части системы.

**Проблемы**:

* Длительность выполнения при большом количестве тест-кейсов.
* Затраты на автоматизацию.
* Определение приоритетов тест-кейсов.

### 23. Определение и цель ручного тестирования. Преимущества и недостатки ручного тестирования. Типы тестирования.

**Определение**: Процесс тестирования, выполняемый вручную без автоматизированных инструментов.

**Цель**: найти дефекты в ПО путем проверки функционала и поведения системы.

**Преимущества**:

* Гибкость.
* Возможность обнаружения UX-проблем.
* Не требует затрат на разработку тестов.

**Недостатки**:

* Затратно по времени.
* Высокая вероятность человеческих ошибок.
* Сложность повторения тестов.

**Типы тестирования**:

* Функциональное
* Интеграционное
* Регрессионное
* Приемочное
* Исследовательское

### 24. Юзабилити тестирование: понятие, основные подходы планирования и разработки программных продуктов.

**Понятие**: Проверка удобства использования ПО пользователями.

**Основные подходы**:

1. **Аналитический** – анализируется структура интерфейса.
2. **Экспертный** – оценка интерфейса специалистами.
3. **Пользовательский** – тестирование реальными пользователями.

### 25. Экспертный подход к юзабилити-тестированию: критерии. Преимущества и недостатки подхода.

**Критерии**:

* Соответствие гайдлайнам (например, Material Design).
* Эстетика и эргономика.
* Простота навигации.

**Преимущества**:

* Быстрое выявление очевидных ошибок.
* Экономия времени.

**Недостатки**:

* Нет реальных пользователей.
* Возможна субъективность.

### 26. Пользовательский подход к юзабилити-тестированию: критерии, методы измерения. Преимущества и недостатки подхода.

**Критерии**:

* Время выполнения задач.
* Число ошибок.
* Оценка удовлетворенности.

**Методы измерения**:

* A/B тестирование.
* Eye-tracking.

**Преимущества**:

* Реальная обратная связь.
* Высокая достоверность.

**Недостатки**:

* Дорого.
* Требует подготовки тест-группы.

### 27. Автоматизация тестирования: понятие, типы, принципы. Требования к тестировщику. Преимущества и недостатки автоматизации тестирования.

**Понятие**: Использование инструментов для автоматического выполнения тестов.

**Типы**:

* UI-тестирование
* API-тестирование
* Нагрузочное тестирование

**Принципы**:

* Регулярность выполнения тестов.
* Поддержка тестов.
* Минимизация ручного труда.

**Требования к тестировщику**:

* Знание инструментов (Selenium, JUnit).
* Навыки программирования.

**Преимущества**:

* Ускорение тестирования.
* Высокая точность.

**Недостатки**:

* Высокая стоимость внедрения.
* Неэффективность для сложных UX-тестов.

### 28. Понятие утверждений и их параметры. Модели и основные виды утверждений. Понятие и категории директив.

**Понятие утверждений**: Условие, которое проверяется в тестах (assert).

**Виды утверждений**:

* Проверка равенства.
* Проверка логических условий.

**Директивы**: Инструкции для компилятора или интерпретатора, например, #define.

### 29. Определение, цель и этапы тестирования производительности. Критерии для измеряемых метрик.

**Определение**: Анализ быстродействия системы.

**Цель**: определить, справляется ли система с нагрузками.

**Этапы**:

1. Планирование тестов.
2. Запуск тестов.
3. Анализ результатов.

**Критерии метрик**:

* Время отклика.
* Пропускная способность.
* Количество запросов в секунду.

### 1. ****Структура и виды CASE-средств. Критерии при выборе.****

**CASE-средства** (Computer-Aided Software Engineering) — инструменты для автоматизации разработки ПО.

#### Виды:

* **Верхнего уровня (Upper CASE)** – поддержка ранних этапов (анализ, проектирование).
* **Нижнего уровня (Lower CASE)** – реализация, тестирование, отладка.
* **Интегрированные (I-CASE)** – объединяют оба уровня.

#### Критерии выбора:

* Поддержка необходимых методологий (например, **RUP, Agile**).
* Интеграция с другими инструментами (**Git, CI/CD**).
* Юзабилити и документация.

### 2. ****Структуры среды разработки. Факторы при выборе.****

Среда разработки (IDE) включает редактор кода, компилятор, отладчик и инструменты сборки.

#### Варианты структур:

* **Локальная (Standalone IDE)** – на одном устройстве (**JetBrains, Eclipse**).
* **Облачная (Cloud IDE)** – веб-доступ (**GitHub Codespaces**).
* **Модульная (Plugin-based)** – расширяемая (например, **VS Code**).

#### Факторы выбора:

* Язык программирования.
* Совместимость с платформой.
* Инструменты тестирования и отладки.

### 3. ****Характеристика основных возможностей среды разработки.****

Основные функции:

* Подсветка синтаксиса и автодополнение.
* Инструменты отладки (watch, breakpoints).
* Система контроля версий (интеграция с **Git**).
* Автоматизация сборки (**Gradle, Maven**).

### 4. ****Основные инструменты среды для создания, исполнения и управления ИС.****

* **Редактор кода** – Visual Studio Code, JetBrains IDEA.
* **Средства тестирования** – JUnit, Selenium.
* **CI/CD-системы** – GitLab CI, Jenkins.
* **Мониторинг и логирование** – Prometheus, Grafana.

### 5. ****Средства обработки информации: определение, виды.****

**Определение**: ПО для сбора, хранения, обработки данных.

#### Виды:

* **Базы данных** (**SQL, NoSQL**).
* **ETL-инструменты** (Extract, Transform, Load) – Apache Nifi.
* **BI-системы** (**Power BI, Tableau**).

### 6. ****Организация работы в команде разработчиков. Виды моделей и роли.****

#### Модели работы:

* **Agile (Scrum, Kanban)**.
* **Waterfall** (каскадная) – последовательная модель.
* **DevOps** – автоматизированный процесс.

#### Роли:

* **Project Manager** – управление проектом.
* **Product Owner** – формирование требований.
* **Developer** – написание кода.

### 7. ****Система контроля версий: совместимость, установка, настройка.****

* Совместимость с ОС (**Git, Mercurial**).
* Установка через пакетные менеджеры (**apt, brew, choco**).
* Настройка .gitconfig, ключи SSH.

### 8. Характеристика методов обеспечения кроссплатформенности информационной системы.

* **Виртуализация** – Docker, Kubernetes.
* **Кроссплатформенные фреймворки** – Qt, Xamarin.
* **Мультиплатформенные языки** – Java, Python.

### 9. Особенности сервисно-ориентированной архитектуры, основные принципы.

* **Разделение сервисов** – модульность.
* **Использование API** – REST, gRPC.
* **Слабая связанность** – минимизация зависимостей.

### 10. Интегрированные среды разработки для создания независимых программ: назначение, сфера применения.

Применение:

* Разработка десктопных и мобильных приложений (**Xcode, Android Studio**).
* Веб-разработка (**WebStorm, PyCharm**).

### 11. ****Особенности объектно-ориентированных языков.****

* **Инкапсуляция** – скрытие данных.
* **Наследование** – переиспользование кода.
* **Полиморфизм** – один интерфейс, разное поведение.

Примеры: **Java, C++, Python**.

### 12. Диаграммы UML вариантов использования, последовательности, кооперации: цель построения и синтаксис.

#### **Варианты использования (Use Case Diagram)**

* Показывает взаимодействие акторов с системой.
* **Синтаксис**: актеры (stickman), кейсы (овалы).

#### **Диаграмма последовательности (Sequence Diagram)**

* Определяет порядок вызова объектов.
* **Синтаксис**: участники (lifeline), сообщения (стрелки).

#### **Диаграмма кооперации (Collaboration Diagram)**

* Показывает связи между объектами.

### 13. Диаграммы UML развертывания, компонентов, потоков данных: цель построения и синтаксис.

#### **Диаграмма развертывания (Deployment Diagram)**

* Определяет размещение компонентов на узлах.

#### **Диаграмма компонентов (Component Diagram)**

* Показывает структуру ПО (библиотеки, модули).

#### **Диаграмма потоков данных (DFD)**

* Описывает движение информации в системе.

### 14. ****Архитектура информационной системы: виды, типы групп, применение архитектурных описаний.****

Виды:

* **Монолитная** – единая структура.
* **Микросервисная** – независимые сервисы.
* **Клиент-серверная** – разделение ролей.

Применение:

* **Монолит** – простые системы.
* **Микросервисы** – масштабируемые приложения.

### 15. Определение конфигурации информационной системы. Этапы, методы и средства конфигурирования.

#### **Этапы:**

1. Определение требований.
2. Выбор технологий.
3. Настройка компонентов.
4. Тестирование и развертывание.

#### **Методы:**

* **Шаблоны конфигурации** – Ansible, Terraform.
* **Автоматическое развертывание** – Docker Compose.

### ****16. Критерии выбора технических средств проекта информационной системы****

Выбор оборудования зависит от следующих критериев:

* **Производительность** – соответствие вычислительным требованиям ИС.
* **Надежность** – уровень отказоустойчивости (RAID, резервные копии).
* **Масштабируемость** – возможность увеличения ресурсов.
* **Совместимость** – поддержка ОС, драйверов, API.
* **Энергоэффективность** – снижение затрат на эксплуатацию.

Пример: Для высоконагруженных ИС выбирают **серверы с многоядерными процессорами (Intel Xeon, AMD EPYC), NVMe SSD** и поддержкой **виртуализации (VMware, KVM)**.

### ****17. Репозиторий проекта информационной системы: понятие и процесс создания****

**Репозиторий** – централизованное хранилище кода, конфигураций и документации.

**Процесс создания:**

1. Выбор системы контроля версий (**Git, Mercurial**).
2. Инициализация (git init).
3. Добавление файлов (git add .).
4. Создание первого коммита (git commit -m "Initial commit").
5. Подключение к удаленному репозиторию (git remote add origin <URL>).
6. Отправка (git push origin main).

Пример: Использование **GitHub, GitLab или Bitbucket** для хранения репозитория.

### ****18. Алгоритм определения уровня доступа к репозиторию в системе контроля версий****

1. **Определение ролей** (администратор, разработчик, тестировщик).
2. **Настройка прав** (чтение, запись, администрирование).
3. **Применение политики защиты веток** (запрет прямого пуша в main).
4. **Настройка аутентификации** (SSH-ключи, OAuth, 2FA).
5. **Мониторинг активности** (git log, веб-интерфейс GitHub/GitLab).

Пример:

sh

КопироватьРедактировать

git config --global user.name "Developer"

git config --global user.email "developer@example.com"

В GitLab права задаются через **Members → Role (Owner, Maintainer, Developer, Reporter, Guest)**.

### 19. Процесс настройки среды разработки информационной системы: описание установки предпочитаемых инструментов и языков кода.

1. **Установка IDE** (**JetBrains IDEA, VS Code, Eclipse**).
2. **Выбор языка программирования** (Python, Java, C#).
3. **Настройка компилятора/интерпретатора** (python --version, javac -version).
4. **Добавление зависимостей** (pip install, npm install).
5. **Настройка системы контроля версий** (git clone).
6. **Конфигурация среды выполнения** (установка JVM, Node.js, .NET).

Пример:

sh

КопироватьРедактировать

sudo apt update && sudo apt install openjdk-17-jdk

### ****20. Инструменты мониторинга разработки проекта****

* **Система контроля версий** – Git, GitLab CI/CD.
* **Трекеры задач** – Jira, Trello.
* **Мониторинг логов** – ELK Stack (Elasticsearch, Logstash, Kibana).
* **CI/CD-инструменты** – Jenkins, GitHub Actions.

Пример: Настройка **Prometheus + Grafana** для мониторинга серверов разработки.

### ****21. Требования к пользовательскому интерфейсу ИС****

* **Юзабилити** – удобство и интуитивность.
* **Доступность** – поддержка разных устройств.
* **Скорость отклика** – минимизация задержек.
* **Кроссбраузерность** – работа в Chrome, Firefox, Edge.
* **Соблюдение UX-гайдлайнов** (Material UI, Apple HIG).

Пример: Форма входа должна включать **автозаполнение, двухфакторную аутентификацию**.

### ****22. Принципы создания графического интерфейса (GUI)****

* **Простота** – минимум элементов.
* **Контрастность** – удобочитаемость.
* **Единообразие** – повторяющиеся паттерны.
* **Адаптивность** – поддержка мобильных устройств.
* **Логичность навигации** – структура по принципу **"3 клика"**.

Пример: Использование **React, Vue.js, Qt, Tkinter** для разработки GUI.

### 23. Понятие и формы спецификации языка программирования для разработки информационной системе.

**Спецификация** – описание синтаксиса и семантики языка.

Формы спецификаций:

* **Официальная документация** (Python PEP, Java JLS).
* **EBNF (Extended Backus-Naur Form)** – формальное описание грамматики.
* **ISO-стандарты** (C++ ISO/IEC 14882, SQL ISO/IEC 9075).

### ****24. Методы организации ввода-вывода информации в информационной системе****

Методы организации ввода-вывода в информационной системе включают:

1. **Синхронный ввод-вывод** – процесс ожидает завершения операции.
2. **Асинхронный ввод-вывод** – выполнение программы продолжается независимо от операции ввода-вывода.
3. **Буферизированный ввод-вывод** – данные временно хранятся в буфере для оптимизации работы.
4. **Прямой доступ к устройствам (DMA)** – передача данных в память без участия процессора.
5. **Потоковый ввод-вывод** – последовательная обработка данных, например, через API.
6. **Файловый ввод-вывод** – чтение/запись данных в файлы.
7. **Сетевой ввод-вывод** – обмен данными через протоколы (HTTP, TCP/IP).

Пример: база данных использует буферизацию при записи данных на диск.

### ****25. Спецификация настроек типовой информационной системы****

Спецификация включает:

1. **Архитектурные параметры** – клиент-серверная модель, облачные технологии.
2. **Требования к аппаратному обеспечению** – процессор, ОЗУ, диск.
3. **Программные требования** – ОС, СУБД, сервер приложений.
4. **Конфигурация пользователей** – роли, права доступа.
5. **Настройки безопасности** – шифрование, аутентификация.
6. **Параметры интеграции** – API, поддерживаемые форматы данных.
7. **Логирование и мониторинг** – инструменты наблюдения за системой.

Пример: CRM-система требует MySQL, 8 ГБ ОЗУ, ролей администратора и менеджера.

### ****26. Сообщения между модулями: назначения, вход и разделы модуля****

**Назначение** – обмен данными между модулями, синхронизация процессов.  
**Вход** – входные параметры API, очереди сообщений, вызовы функций.  
**Разделы модуля**:

* **Интерфейсный слой** – принимает запросы.
* **Логика обработки** – выполняет вычисления.
* **Модуль данных** – обращается к БД.

Пример: модуль авторизации отправляет токен в модуль проверки прав доступа.

### ****27. Приложения для моделирования процессов: виды моделей, требования****

**Виды моделей**:

1. **Графические** – диаграммы BPMN, UML.
2. **Имитационные** – Arena, AnyLogic.
3. **Алгебраические** – математическое моделирование.
4. **Агентные** – модели взаимодействия объектов.

**Требования**:

* Поддержка нотаций BPMN, UML.
* Визуальное представление процессов.
* Интеграция с БД.
* Генерация отчетов.

Пример: BPMN-диаграммы используются для моделирования бизнес-процессов в Camunda.

### ****28. Разработка приложений: виды, этапы разработки****

**Виды**:

* Веб-приложения.
* Мобильные приложения.
* Десктопные приложения.
* Встроенные системы.

**Этапы**:

1. Анализ требований.
2. Проектирование архитектуры.
3. Программирование.
4. Тестирование.
5. Внедрение.
6. Поддержка.

Пример: при разработке интернет-магазина сначала создается ТЗ, затем пишется код, проводится тестирование.

### ****29. Отладка приложения: определение, существующие методы****

**Определение** – процесс поиска и исправления ошибок.

**Методы**:

1. **Логирование** – запись действий программы.
2. **Интерактивная отладка** – использование отладчика (например, GDB).
3. **Анализ дампов памяти** – изучение состояния программы при сбоях.
4. **Тестирование** – юнит-тесты, интеграционные тесты.
5. **Инструменты профилирования** – анализ производительности.

Пример: если сайт падает при загрузке, изучается лог ошибок сервера.

### ****30. Интеграция модуля: определение, возможность, способы интеграции****

**Определение** – процесс объединения модуля в систему.

**Возможность интеграции**:

* Открытые API.
* Поддержка стандартных форматов (JSON, XML).
* Гибкость архитектуры (микросервисы).

**Способы**:

1. **API-интеграция** – REST, SOAP.
2. **Сообщения в шине данных (ESB, Kafka)**.
3. **Базы данных** – общий доступ.
4. **Файловый обмен** – CSV, XML.

Пример: платежный модуль подключается к сайту через REST API.