Вопросы к экзамену по дисциплине «Архитектура, проектирование и разработка программных средств» 2022-23 учебный год

**1. Технология разработки программного обеспечения. Определение. Краткая характеристика.**

**Определение**: Технология разработки ПО – это совокупность методов, инструментов и процессов, используемых для проектирования, разработки, тестирования и сопровождения программного обеспечения.  
**Характеристика**: Включает этапы жизненного цикла (анализ требований, проектирование, кодирование, тестирование, внедрение и сопровождение). Основная цель – создание качественного, функционального и надежного программного обеспечения.

**2. Технология разработки программного обеспечения. Определение. Основные требования.**

**Основные требования**:

* Упорядоченность и формализация процессов.
* Использование современных инструментов автоматизации.
* Поддержка интеграции и совместимости компонентов.
* Адаптивность к изменениям требований.
* Обеспечение высокого уровня качества и безопасности ПО.

**3. Понятие качества программного обеспечения. Механизмы обеспечения.**

**Качество ПО** – это совокупность характеристик, которые определяют его способность удовлетворять установленные и предполагаемые требования пользователей.  
**Механизмы обеспечения**:

* Использование стандартов (например, ISO/IEC 25010).
* Ревизии и контроль качества на всех этапах разработки.
* Тестирование (unit-тесты, интеграционное, системное и регрессионное).
* Инструменты анализа кода и автоматизации тестирования.
* Внедрение CI/CD для постоянной проверки.

**4. Основные параметры качества в соответствии со стандартом ISO/IEC 25010:2011.**

Стандарт определяет восемь характеристик качества:

1. Функциональность.
2. Надежность.
3. Удобство использования (юзабилити).
4. Производительность (эффективность).
5. Совместимость.
6. Удобство сопровождения.
7. Безопасность.
8. Переносимость.

**5. Основные параметры качества (ISO/IEC 25010:2011). Функциональность.**

**Функциональность** – способность ПО предоставлять ожидаемые функции при определённых условиях.

* Подпараметры: полнота, корректность, пригодность, соблюдение стандартов.

**6. Цели и задачи технологий разработки ПО. Особенности современных крупных проектов ИС.**

**Цели и задачи**:

* Повышение производительности труда разработчиков.
* Сокращение времени разработки.
* Уменьшение количества ошибок.
* Обеспечение совместимости и масштабируемости.  
  **Особенности крупных проектов ИС**:
* Высокая сложность.
* Многокомандное участие.
* Большое количество требований.
* Непрерывная интеграция и развертывание.

**7. Основные параметры качества (ISO/IEC 25010:2011). Надежность.**

**Надежность** – способность ПО сохранять свою производительность в условиях определённых нагрузок.

* Подпараметры: устойчивость к сбоям, способность восстанавливаться, доступность.

**8. Основные параметры качества (ISO/IEC 25010:2011). Юзабилити.**

**Юзабилити** – удобство и лёгкость использования ПО для выполнения задач.

* Подпараметры: понятность, обучаемость, привлекательность интерфейса, возможность адаптации.

**9. Основные параметры качества (ISO/IEC 25010:2011). Эффективность.**

**Эффективность** – способность ПО обеспечивать высокий уровень производительности при минимальных затратах ресурсов.

* Подпараметры: время отклика, пропускная способность, оптимизация ресурсов.

**10. Основные параметры качества (ISO/IEC 25010:2011). Удобство сопровождения.**

**Удобство сопровождения** – лёгкость внесения изменений и обновлений в ПО.

* Подпараметры: модульность, повторное использование, тестируемость, анализируемость.

### 11. Основные параметры качества (ISO/IEC 25010:2011). Факторы, влияющие на качество ПО. Классификация факторов.

**Факторы, влияющие на качество ПО**:

* Организационные (команда, управление проектом).
* Технические (архитектура, выбор технологий).
* Экономические (бюджет, сроки).
* Внешние (изменение требований, рыночные условия).  
  **Классификация факторов**:

1. Внутренние – архитектура, модульность, стандарты кода.
2. Внешние – пользовательский опыт, совместимость с другими системами.
3. Процессные – качество управления проектом.

### 12. Системный подход к разработке ПО. Понятие «Эвроритм». Структура системного подхода.

**Эвроритм** – это методология, описывающая последовательные циклы анализа, синтеза и проверки решений при разработке систем.  
**Структура системного подхода**:

* Анализ требований.
* Построение модели системы.
* Проектирование и разработка.
* Тестирование и валидация.

### 13. Системный подход к разработке ПО. Компонентный анализ.

**Компонентный анализ**: исследование и разбиение системы на составные части (модули/компоненты), анализ их функциональности, взаимодействия и повторного использования.

### 14. Системный подход к разработке ПО. Структурный анализ.

**Структурный анализ**: процесс декомпозиции сложной системы на подсистемы с выделением их взаимосвязей, потоков данных и иерархии. Использует диаграммы потоков данных (DFD).

### 15. Системный подход к разработке ПО. Функциональный анализ.

**Функциональный анализ**: определение задач, функций и действий системы, которые удовлетворяют пользовательским требованиям.

### 16. Системный подход к разработке ПО. Параметрический анализ.

**Параметрический анализ**: оценка характеристик системы (производительность, надёжность, масштабируемость) с целью выбора оптимальных параметров.

### 17. Системный подход к разработке ПО. Генетический анализ.

**Генетический анализ**: исследование эволюции системы, анализ прошлых версий для улучшения текущей разработки.

### 18. Каскадная модель жизненного цикла ПО.

**Описание**: линейная модель разработки, в которой фазы выполняются последовательно (анализ → проектирование → разработка → тестирование → сопровождение).  
**Плюсы**: простота, четкая структура.  
**Минусы**: отсутствие гибкости, сложность адаптации к изменениям.

### 19. Паттерн ****Bridge**** (Мост).

**Описание**: разделяет абстракцию и её реализацию, позволяя им изменяться независимо.  
**Структура**: абстракция, конкретная абстракция, интерфейс реализации, конкретная реализация.  
**Пример**: Рендеринг (2D или 3D) графических фигур (кругов, квадратов).

### 20. Паттерн ****Decorator**** (Декоратор).

**Описание**: динамически добавляет объекту новые обязанности.  
**Структура**: компонент, конкретный компонент, декоратор, конкретный декоратор.  
**Пример**: добавление функций к пользовательскому интерфейсу (например, скроллинг).

### 21. Паттерн ****Facade**** (Фасад).

**Описание**: упрощает доступ к сложной системе через единый интерфейс.  
**Структура**: фасад, подсистемы.  
**Пример**: мультимедийный плеер, скрывающий сложность работы кодеков.

### 22. Паттерн ****Proxy**** (Заместитель).

**Описание**: представляет объект, предоставляя к нему контролируемый доступ.  
**Структура**: клиент, прокси, реальный объект.  
**Пример**: кэширование данных для удалённого доступа.

### 23. Паттерн ****Command**** (Команда).

**Описание**: инкапсулирует запрос как объект.  
**Структура**: команда, получатель, клиент, инициатор.  
**Пример**: действия меню в GUI (Undo, Redo).

### 24. Паттерн ****Memento**** (Хранитель).

**Описание**: сохраняет состояние объекта для его восстановления.  
**Структура**: объект, хранитель, хранитель состояния.  
**Пример**: функции сохранения/загрузки.

### 25. Паттерн ****Iterator**** (Итератор).

**Описание**: обеспечивает доступ к элементам коллекции без раскрытия её внутренней структуры.  
**Пример**: перебор элементов массива или списка.

### 26. Паттерн ****Mediator**** (Посредник).

**Описание**: координирует взаимодействие между объектами.  
**Пример**: чаты, где посредник – сервер.

### 27. Паттерн ****Strategy**** (Стратегия).

**Описание**: определяет семейство алгоритмов и делает их взаимозаменяемыми.  
**Пример**: сортировка с разными алгоритмами (QuickSort, MergeSort).

### 28. Паттерн ****Chain of Responsibility**** (Цепочка обязанностей).

**Описание**: передача запроса по цепочке обработчиков до его выполнения.  
**Пример**: обработка исключений в приложении.

### 29. CASE-технология. CASE-средства. CASE-системы.

**Описание**: компьютеризированные инструменты для автоматизации разработки ПО.  
**CASE-средства**: инструменты проектирования, моделирования, управления проектами.  
**CASE-системы**: комплекс интегрированных средств для полного цикла разработки.

### 30. Области применения CASE-технологий. Классификация технологий разработки ИС.

**Области применения**: проектирование баз данных, моделирование процессов, тестирование.  
**Классификация технологий**:

* Традиционные (каскадная, спиральная).
* Гибкие (Agile, Scrum).
* Совмещённые.

### 31. Базовые принципы объектно-ориентированной методологии проектирования

**Принципы**:

1. **Инкапсуляция** – объединение данных и методов, работающих с ними, в одном объекте.
2. **Наследование** – создание новых классов на основе существующих с добавлением или переопределением функций.
3. **Полиморфизм** – возможность объектов разных классов использовать одинаковый интерфейс.
4. **Абстракция** – выделение значимых характеристик объекта и игнорирование несущественных.

### 32. Основные понятия методологии проектирования ИС. Этапы развития технологий проектирования ИС

**Основные понятия**:

* Информационная система (ИС): совокупность методов, процессов, и технологий для обработки данных.
* Моделирование: создание моделей, отражающих реальные процессы.
* Жизненный цикл: этапы создания и сопровождения ИС.

**Этапы развития**:

1. Ручное проектирование.
2. Структурный подход.
3. Объектно-ориентированный подход.
4. Гибкие методологии (Agile).

### 33. Жизненный цикл ИС. Модели жизненного цикла

**Жизненный цикл ИС** – последовательность этапов создания и поддержки системы.  
**Модели ЖЦ**:

1. **Каскадная модель** – линейное выполнение этапов.
2. **Поэтапная модель с промежуточным контролем** – проверка завершения каждого этапа перед переходом к следующему.
3. **Спиральная модель** – разработка через последовательные итерации с анализом рисков.
4. **Agile** – гибкая модель с акцентом на итеративное развитие и взаимодействие с пользователями.

### 34. Методология объектно-ориентированного программирования

**Принципы ООП**:

1. **Абстракция** – выделение только значимой информации.
2. **Инкапсуляция** – защита данных от прямого доступа.
3. **Наследование** – создание иерархии классов.
4. **Полиморфизм** – выполнение однотипных операций над объектами разных классов.

### 35. Методология объектно-ориентированного анализа и проектирования. Концептуализация предметной области

**Концептуализация** – процесс определения основных сущностей, их свойств и связей, отражающих предметную область, для построения моделей системы.

### 36. Методология объектно-ориентированного анализа и проектирования. Роль архитектора программного обеспечения

**Роль архитектора ПО**:

* Разработка архитектуры системы.
* Выбор технологий и инструментов.
* Контроль соответствия системы требованиям.
* Решение сложных технических задач.

### 37. CRC-карточки (Component, Responsibility, Collaborator)

**CRC-карточки**: метод проектирования, где для каждого класса описываются:

1. **Component (компонент)** – название класса.
2. **Responsibility (обязанности)** – задачи, которые выполняет класс.
3. **Collaborator (сотрудники)** – другие классы, с которыми взаимодействует данный.

### 38. Современные CASE-средства. Классификация и характеристики

**CASE-средства** – инструменты для автоматизации разработки ПО.  
**Классификация**:

1. По этапам ЖЦ (анализ, проектирование, тестирование).
2. По интеграции (локальные, интегрированные).
3. По функциональности (моделирование, управление проектами).  
   **Характеристики**: визуализация, поддержка командной работы, интеграция с другими инструментами.

### 39. UML как инструмент моделирования

**UML (Unified Modeling Language)** – язык для визуализации, проектирования и документирования систем.  
**Примеры диаграмм**:

* Диаграммы классов.
* Диаграммы последовательностей.
* Диаграммы случаев использования.

### 40. Современные CASE-технологии

**Назначение**: автоматизация процессов анализа, проектирования, разработки и сопровождения ИС.  
**Функции**: моделирование, генерация кода, управление проектами, тестирование.  
**Классификация**:

1. По функциональности (аналитические, проектные, тестовые).
2. По интеграции (универсальные, специализированные).

### 41. Концептуальная модель UML

**Концептуальная модель UML** – это описание сущностей и связей языка UML, позволяющее моделировать системы.  
**Строительные блоки UML**:

1. **Основные элементы**: диаграммы, сущности, связи.
2. **Группирующие элементы**: пакеты.
3. **Анотационные элементы**: комментарии.

**Правила UML**:

* Определяют допустимые связи и структуру диаграмм.

**Общие механизмы UML**:

* Стереотипы, метки, ограничения.

**Архитектура и жизненный цикл ПО**:

* UML охватывает все этапы жизненного цикла ПО, от анализа требований до тестирования и поддержки.

### 42. Языки моделирования и их реализация в современных CASE-средствах

**Языки моделирования**:

* **UML** – для объектно-ориентированных систем.
* **BPMN** – для бизнес-процессов.
* **ERD** – для проектирования баз данных.

**Реализация в CASE-средствах**:

* **Enterprise Architect**, **Rational Rose** – поддерживают моделирование на UML.
* **ARIS**, **Bizagi** – для BPMN.

### 43. Язык UML. Словарь и предметы в UML

**Словарь UML**: включает термины и определения, используемые при моделировании, например, "класс", "объект", "ассоциация".  
**Предметы UML**:

* Статические элементы: классы, объекты, пакеты.
* Динамические элементы: события, состояния.

### 44. Понятие архитектуры информационной системы

**Архитектура ИС**: структурированное описание компонентов системы, их взаимосвязей, взаимодействия с пользователями и внешней средой.

### 45. Архитектура информационной системы. Формальные методы описания

**Формальные методы**:

1. **Диаграммы UML** – для структурного и динамического описания.
2. **Языки ADL** (Architecture Description Language) – для формализации архитектуры.

### 46. Архитектурные структуры и представления. Понятие "представление"

**Представление** – это взгляд на архитектуру ИС, акцентирующий внимание на отдельных аспектах (например, логическая структура, физическая инфраструктура).

### 47. Архитектурные структуры и представления. Классификация

**Классификация архитектурных структур**:

1. **Логические** – описание функций и компонентов.
2. **Физические** – топология размещения оборудования.
3. **Процессные** – потоки данных, процессы.
4. **Разработки** – структура программного кода.

### 48. Структурное проектирование и программирование. Модульная структура программы

**Структурное проектирование**: декомпозиция системы на функциональные модули.  
**Модульная структура**:

* Линейная.
* Иерархическая.
* Взаимозависимая.

### 49. Архитектуры: уровни абстракций, порты, потоки данных

**Архитектуры**:

1. **Уровни абстракций** – многоуровневые системы (например, клиент-сервер).
2. **Порты** – интерфейсы взаимодействия между компонентами.
3. **Потоки данных** – архитектура, основанная на передаче данных через каналы.

### 50. Методы формального функционального проектирования

**Методы**:

1. **DFD (Data Flow Diagrams)** – анализ потоков данных.
2. **FSM (Finite State Machines)** – представление поведения систем.
3. **Petri Nets** – моделирование процессов с параллелизмом.

### 51. Роль и место проектирования в жизненном цикле ИС

**Роль проектирования**:  
Проектирование является центральным этапом ЖЦ ИС, определяющим структуру, поведение и взаимодействие компонентов системы. Это мост между анализом требований и реализацией.

**Место в ЖЦ**:

* Следует за анализом требований.
* Предшествует разработке и тестированию.

### 52. Архитектурные структуры и представления. Структуры «компонент и соединитель». Классификация

**Структуры «компонент и соединитель»** описывают компоненты системы и их взаимодействия.

**Классификация**:

1. **Поток управления** – взаимодействие компонентов через вызовы методов.
2. **Обмен данными** – взаимодействие через потоки данных.
3. **Общий ресурс** – взаимодействие через общие базы данных или файловую систему.

### 53. Архитектурные структуры и представления. Структуры распределения

**Структуры распределения** описывают, как компоненты системы распределяются по узлам сети.

**Классификация**:

1. **Клиент-сервер** – разделение на клиентские и серверные компоненты.
2. **Распределенные системы** – децентрализованная обработка данных.
3. **Облачные структуры** – размещение компонентов в облачных средах.

### 54. Варианты архитектур программных систем. Архитектуры, основанные на уровнях абстракций

**Уровни абстракций**:

* Деление системы на логические слои (например, уровень представления, бизнес-логики, данных).
* Примеры: архитектура трехуровневых систем, MVC (Model-View-Controller).

### 55. Варианты архитектур программных систем. Архитектуры, основанные на портах

**Архитектуры на портах**:

* Определяют взаимодействие компонентов через четко описанные интерфейсы (порты).
* Пример: микросервисные архитектуры, где порты определяют API для взаимодействия.

### 56. Варианты архитектур программных систем. Архитектуры, основанные на потоках данных

**Архитектуры потоков данных**:

* Основываются на передаче данных между компонентами через каналы.
* Примеры: конвейерные системы, системы обработки данных в реальном времени.

### 57. Архитектурно-экономический цикл. Влияние на архитектуру ПО

**Архитектурно-экономический цикл** – процесс выбора архитектурных решений с учетом их стоимости и выгод.  
**Влияние**:

* Архитектура должна обеспечивать баланс между затратами на разработку и эксплуатацию и выгодами от использования.

### 58. Выявление и разработка требований к программному обеспечению

**Выявление требований**:

* Методы: интервью, наблюдение, мозговой штурм, анализ документации.

**Разработка требований**:

* Составление спецификаций: функциональных, нефункциональных и ограничений.

### 59. Анализ или оценка архитектуры. Методы анализа архитектуры

**Методы анализа архитектуры**:

1. **ATAM (Architecture Tradeoff Analysis Method)** – анализ компромиссов.
2. **SAAM (Software Architecture Analysis Method)** – оценка пригодности архитектуры.
3. **CBAM (Cost Benefit Analysis Method)** – оценка стоимости и выгод архитектурных решений.

### 60. Метод анализа стоимости и эффективности (CBAM)

**CBAM (Cost Benefit Analysis Method)** – метод анализа, направленный на оценку выгод и затрат архитектурных решений.

**Этапы**:

1. Определение целей.
2. Сбор данных о затратах и выгодах.
3. Моделирование и расчет показателей.
4. Выбор решений на основе анализа.

4o

### 61. Метод анализа компромиссных архитектурных решений (ATAM)

**ATAM** – метод оценки архитектуры программных систем для выявления компромиссов между качественными атрибутами (надежность, производительность, масштабируемость).

**Основные этапы**:

1. Определение целей и сценариев использования.
2. Идентификация архитектурных решений.
3. Анализ взаимодействия решений с атрибутами качества.
4. Выявление компромиссов.
5. Представление результатов анализа.

### 62. Архитектура программных систем. Архитектурный образец

**Понятие**: Архитектурный образец (pattern) – это проверенное решение общих задач проектирования систем.  
**Назначение**: Упрощение проектирования за счет использования стандартных подходов.  
**Примеры**: MVC, клиент-сервер, микросервисы.

### 63. Архитектура программных систем. Эталонная модель

**Понятие**: Эталонная модель описывает стандартный набор компонентов и их взаимодействие.  
**Назначение**: Обеспечение единообразия, упрощение интеграции и стандартизации.  
**Взаимосвязь**: Эталонные модели формируют базу для архитектурных образцов.

### 64. Характеристики "хорошей" архитектуры

1. **Модульность** – легко заменяемые компоненты.
2. **Масштабируемость** – адаптация к увеличению нагрузки.
3. **Надежность** – устойчивость к сбоям.
4. **Понятность** – простота восприятия.
5. **Гибкость** – возможность изменения без значительных затрат.

### 65. Детальное проектирование. Проектирование классов

**Понятия**:

* **Активный класс** – класс, имеющий свой поток управления.
* **Пассивный класс** – класс, используемый другими, без собственного потока.

### 66. Проектирование классов

**Общий подход**:

* Определение обязанностей и взаимодействий классов.
* Минимизация связности и увеличение независимости.

**Требования к именованию**:

* Классы: понятные, отражающие назначение (например, UserManager).
* Методы: ясные, описывающие действия (например, getUserData).

**Диаграммы UML**:

* Диаграммы классов.
* Диаграммы последовательностей.

### 67. Сценарий использования

**Назначение**: Описание взаимодействия пользователей с системой для достижения цели.  
**Диаграмма вариантов использования**:

* Представляет акторов и их взаимодействия с системой через прецеденты.

### 68. Связи как механизм коммуникации

**Диаграмма связи UML**:

* Описывает взаимодействие объектов в контексте выполнения задач.
* Элементы: объекты, сообщения, ассоциации.

### 69. Диаграмма классов

**Описание**:

* Отображает классы, их атрибуты, методы и связи между ними.
* Связи: ассоциации, агрегации, композиции, зависимости.

**Пример**:  
Класс Order связан с классом Customer через ассоциацию.

### 70. Шаблоны проектирования

**Назначение**: Повышение эффективности разработки за счет использования готовых решений.  
**Классификация**:

1. **Порождающие** – Factory, Singleton.
2. **Структурные** – Adapter, Decorator.
3. **Поведенческие** – Observer, Strategy.

**Примеры**:

* **Factory** – создание объектов.
* **Observer** – уведомление подписчиков о событиях.

### 71. Функционально-ориентированное (структурное) проектирование программного обеспечения

**Описание**:  
Методология, сосредоточенная на разделении программы на функциональные блоки, основанные на задачах, которые должна выполнять система.

**Основные принципы**:

* Декомпозиция функций системы на иерархию.
* Четкое определение входных и выходных данных.
* Разработка диаграмм потоков данных (DFD).

### 72. Структурные методологии и подходы к анализу и проектированию

**Методологии**:

1. **SADT (Structured Analysis and Design Technique)**:
   * Построение диаграмм потоков данных и контекстных диаграмм.
2. **DFD (Data Flow Diagram)**:
   * Графическое представление потоков данных в системе.

**Подходы**:

* Декомпозиция: деление системы на отдельные модули.
* Иерархическая структура: выстраивание взаимосвязей между уровнями.

### 73. Паттерн Memento (Хранитель)

**Описание**:  
Паттерн для сохранения и восстановления состояния объекта без нарушения его инкапсуляции.

**Структура**:

* **Originator** – объект, состояние которого нужно сохранять.
* **Memento** – объект, хранящий состояние.
* **Caretaker** – управляет сохранением и восстановлением.

**Пример**:  
Сохранение позиции в текстовом редакторе для функции «Отменить».

### 74. Паттерн Iterator (Итератор)

**Описание**:  
Паттерн для последовательного доступа к элементам коллекции без раскрытия внутреннего устройства.

**Структура**:

* **Iterator** – интерфейс для итерации.
* **ConcreteIterator** – конкретная реализация итератора.
* **Aggregate** – коллекция.
* **ConcreteAggregate** – реализация коллекции.

**Пример**:  
Итерация по элементам списка или массива.

### 75. Паттерн Mediator (Посредник)

**Описание**:  
Паттерн, который централизует взаимодействие между объектами, уменьшая их связность.

**Структура**:

* **Mediator** – интерфейс для посредников.
* **ConcreteMediator** – реализация посредника.
* **Colleague** – участники взаимодействия.

**Пример**:  
Чат-приложение, где сервер выступает посредником между пользователями.

### 76. Паттерн Strategy (Стратегия)

**Описание**:  
Паттерн для выбора и замены алгоритмов во время выполнения.

**Структура**:

* **Strategy** – интерфейс для алгоритмов.
* **ConcreteStrategy** – реализация конкретного алгоритма.
* **Context** – клиент, использующий стратегию.

**Пример**:  
Разные методы сортировки массива (по возрастанию, по убыванию).

### 77. Паттерн Chain of Responsibility (Цепочка обязанностей)

**Описание**:  
Паттерн для передачи запроса через цепочку обработчиков, где каждый обработчик решает, обрабатывать запрос или передать дальше.

**Структура**:

* **Handler** – интерфейс для обработчиков.
* **ConcreteHandler** – конкретный обработчик.
* **Client** – отправитель запроса.

**Пример**:  
Система обработки запросов в техподдержке (оператор → специалист → руководитель).

### 78. CASE-технология. CASE-средства. CASE-системы

**CASE (Computer-Aided Software Engineering)** – технологии автоматизации процессов анализа, проектирования и разработки ПО.

**CASE-средства**:

* **Upper CASE**: для анализа и проектирования (UML).
* **Lower CASE**: для реализации (IDE).
* **Integrated CASE**: объединение всех этапов.

**Примеры систем**: Rational Rose, Enterprise Architect.

### 79. Области применения CASE-технологий. Классификация технологий разработки информационных систем

**Области применения**:

* Автоматизация проектирования и моделирования.
* Генерация исходного кода.
* Управление требованиями и тестированием.

**Классификация технологий**:

1. **Методологии**: водопадная, спиральная, Agile.
2. **Инструменты**: UML, BPMN.
3. **Автоматизация процессов**: CI/CD.

### 80. Базовые принципы объектно-ориентированной методологии проектирования

1. **Инкапсуляция** – сокрытие внутренней реализации класса.
2. **Наследование** – использование свойств базового класса в дочерних.
3. **Полиморфизм** – единый интерфейс для работы с разными объектами.
4. **Абстракция** – выделение ключевых характеристик объекта.

### 81. Основные понятия методологии проектирования ИС. Этапы развития технологий проектирования ИС

**Методология проектирования ИС** – совокупность методов, правил, средств и приемов, используемых при создании информационных систем (ИС).

**Этапы развития технологий проектирования ИС**:

1. **Ручное проектирование** – создание спецификаций без автоматизации.
2. **Структурный подход** – акцент на функциональность и процессы.
3. **Объектно-ориентированный подход** – моделирование данных и поведения.
4. **Компонентно-ориентированное проектирование** – создание модульных систем.
5. **Современные методологии** – Agile, DevOps, CI/CD.

### 82. Жизненный цикл ИС. Модели жизненного цикла.

**Жизненный цикл ИС** – процесс создания, эксплуатации и завершения использования системы.

**Модели ЖЦ**:

1. **Каскадная модель**: линейный процесс от анализа к внедрению.
2. **Поэтапная модель с промежуточным контролем**: добавление этапов тестирования между шагами.
3. **Спиральная модель**: итеративный процесс с учетом рисков.
4. **Agile**: гибкая модель с итерациями.

### 83. Понятие бизнес-процесса. Классификация бизнес-процессов (по уровню значимости, структуре, назначению)

**Бизнес-процесс** – последовательность действий, направленных на достижение бизнес-целей.

**Классификация**:

* **По уровню значимости**: стратегические, основные, вспомогательные.
* **По структуре**: линейные, разветвленные, циклические.
* **По назначению**: производственные, управленческие, обслуживающие.

### 84. Понятие бизнес-процесса. Классификация бизнес-процессов (по отношению к клиентам, уровню подробности рассмотрения, уровню сложности)

**Классификация**:

* **По отношению к клиентам**: ориентированные на клиента, внутренние.
* **По уровню подробности**: верхнеуровневые, детализированные.
* **По уровню сложности**: простые, сложные (включают множество этапов).

### 85. Технология разработки программного обеспечения. Определение. Краткая характеристика

**Технология разработки ПО** – совокупность методов, процессов и инструментов, используемых для создания программного обеспечения.

**Характеристика**:

* Упрощает создание сложных систем.
* Снижает затраты на разработку.
* Включает управление требованиями, тестирование, эксплуатацию.

### 86. Основные требования, предъявляемые к технологии разработки ПО

1. **Эффективность** – минимизация времени и ресурсов.
2. **Масштабируемость** – возможность адаптации к росту системы.
3. **Гибкость** – возможность внесения изменений.
4. **Безопасность** – защита данных.
5. **Качество** – соответствие стандартам и требованиям.

### 87. Понятие качества программного обеспечения. Механизмы обеспечения при создании программных систем

**Качество ПО** – соответствие системы функциональным и нефункциональным требованиям.

**Механизмы обеспечения**:

* Тестирование на разных уровнях.
* Управление требованиями.
* Использование стандартов (ISO/IEC 25010).
* Непрерывная интеграция и развертывание (CI/CD).

### 88. Основные параметры качества в соответствии со стандартом ISO/IEC 25010:2011

1. **Функциональность** – соответствие требованиям.
2. **Надежность** – способность работать без сбоев.
3. **Юзабилити** – удобство использования.
4. **Эффективность** – оптимальное использование ресурсов.
5. **Удобство сопровождения** – простота внесения изменений.

### 89. Необходимость архитектуры системы. Описание архитектуры системы с использованием пакетов

**Необходимость архитектуры**:

* Определяет структуру и поведение системы.
* Обеспечивает масштабируемость и надежность.

**Описание с использованием пакетов**:

* Использование UML-диаграмм пакетов для группировки компонентов системы.
* Упрощение анализа и проектирования.

### 90. Определения бизнес-процесса. Свойства бизнес-процесса

**Свойства**:

1. **Целевая направленность** – достижение конкретного результата.
2. **Измеримость** – наличие метрик.
3. **Логическая структура** – последовательность действий.
4. **Ориентация на клиента** – обеспечение ценности для клиента.

### 91. Методологии структурного анализа и проектирования. Структурный анализ. Нотация IDEF0

**Структурный анализ** – методология, направленная на разделение системы на составные части для упрощения проектирования и управления.

**Нотация IDEF0**:

* Используется для моделирования функций и процессов.
* Диаграммы IDEF0 отображают:
  + **Входы** – данные или материалы.
  + **Выходы** – результаты выполнения функции.
  + **Управление** – ограничения и правила.
  + **Механизмы** – ресурсы и инструменты.

### 92. Инструментальные средства моделирования бизнес-процессов и проектирования архитектуры ПО

**Средства моделирования**:

1. **ARIS** – для проектирования бизнес-процессов.
2. **Enterprise Architect** – создание UML-диаграмм и проектирование ПО.
3. **Bizagi** – моделирование процессов в нотации BPMN.
4. **Visual Paradigm** – моделирование архитектуры и процессов.

### 93. Технология разработки программного обеспечения. Определение. Краткая характеристика

**Технология разработки ПО** – совокупность процессов, методов и инструментов для создания программного продукта.

**Характеристика**:

* Инкрементальный и итеративный подходы.
* Включает сбор требований, проектирование, тестирование, внедрение и сопровождение.

### 94. Концептуальное проектирование информационных систем. Проблематика концептуальных схем

**Концептуальное проектирование ИС** – разработка модели системы на уровне абстракции, независимой от технологий реализации.

**Проблематика концептуальных схем**:

* Сложность согласования с заказчиком.
* Проблемы совместимости с технической реализацией.
* Высокая трудоемкость в больших проектах.

### 95. Жизненный цикл разработки ПО. Экстремальное программирование. Шаблон проектирования MVC

1. **Жизненный цикл разработки ПО**: последовательность этапов создания, внедрения и сопровождения ПО.
2. **Экстремальное программирование (XP)**: методология Agile, акцент на взаимодействии команды, постоянной обратной связи и простоте.
3. **MVC (Model-View-Controller)**:
   * **Model**: управление данными.
   * **View**: отображение данных.
   * **Controller**: обработка взаимодействия пользователя.

### 96. Основные характеристики объектно-ориентированного моделирования. Принципы объектно-ориентированного анализа

**Характеристики ОО-моделирования**:

* Использование классов и объектов.
* Инкапсуляция, наследование, полиморфизм.

**Принципы ОО-анализа**:

1. **Идентификация объектов и классов**.
2. **Определение атрибутов и методов**.
3. **Выявление связей и взаимодействий между объектами**.

### 97. Архитектура предприятия. Понятие. Подходы к описанию архитектуры предприятия

**Архитектура предприятия** – совокупность структур, процессов и технологий, обеспечивающих работу организации.

**Подходы**:

1. **TOGAF (The Open Group Architecture Framework)** – управление архитектурой.
2. **Zachman Framework** – описание архитектуры с разных точек зрения.
3. **Архитектура 4 доменов**: бизнес, данные, приложения, технологии.

### 98. Архитектура предприятия. Понятие. Моделирование процессов в организации

**Моделирование процессов**:

* Использование нотаций (BPMN, EPC) для описания рабочих процессов.
* Инструменты: Bizagi, ARIS, Camunda.

### 99. Системный подход к проектированию информационных систем. Визуальное моделирование

**Системный подход**: комплексный анализ всех компонентов системы, их связей и взаимодействий.

**Визуальное моделирование**:

* Использование графических языков (UML, SysML).
* Создание диаграмм: классов, последовательности, компонентов.

### 100. Основные понятия графической нотации и семантики языка UML

**UML (Unified Modeling Language)** – стандарт для визуального представления систем.

**Основные понятия**:

* **Графическая нотация**: классы, отношения, пакеты, компоненты.
* **Семантика**: определяет значение графических элементов.
* **Диаграммы UML**: классов, случаев использования, компонентов, последовательности.