1. Понятие системы. Общие представления о сложной системе.

Система – отграниченное, взаимно связанное множество, отражающее объективное существование конкретных отдельных взаимосвязанных совокупностей объектов и не содержащее специфических ограничений, присущих частным системам.
Основные свойства системы:
- **Ограниченность** – отделена от окружающей среды границами.
- **Целостность** – свойство целого принципиально не сводится к сумме свойств составляющих элементов.
- **Структурность** – поведение обусловлено не только особенностями отдельных элементов, сколько свойствами её структуры.
- **Взаимозависимость со средой** – формирует и проявляет свойства в процессе взаимодействия со средой.
- **Иерархичность** – соподчинённость элементов.
- **Множественность описаний** – по причине сложности познание требует множественности взглядов.

2. Основные понятия информационной системы (ИС). Классы ИС.

Информационная система — это взаимосвязанная совокупность средств, методов и персонала, используемых для хранения, обработки и выдачи информации в интересах достижения поставленной цели.
Обеспечивающие подсистемы ИС: Для полноценного функционирования ИС включает в себя несколько видов обеспечения:
информационное (системы классификации, схемы потоков данных), **техническое** (комплекс средств и документация), **математическое и программное** (алгоритмы и программы), **организационное** (взаимодействие персонала) и **правовое** (юридический статус и нормы).
Классификация ИС:
- **По типу данных:** фактографические и документальные.
- **По степени автоматизации:** ручные, автоматизированные и автоматические.
- **По сфере применения:** интегрированные системы, системы организационного управления, системы управления технологическими процессами (ТП) и САПР.
- **По характеру обработки данных:** информационно-поисковые и информационно-решающие (которые, в свою очередь, делятся на управляющие и советующие).
- **По уровню управления:** стратегические, функциональные и операционные.

3. Понятие модели. Виды моделей. Задачи моделирования. Модели ИС.

Модель — это формальное описание особенностей системы, которые являются существенными для целей её исследования.
Виды моделей по способу представления:
1. **Вербальные** (словесные описания).
2. **Натурные** (физическое воспроизведение объекта).
3. **Знаковые** (включая **математические модели**, описывающие состояние и изменения системы на языке алгоритмов и формул).

Классификация моделей ИС:
- **По строгости описания**: формальные и неформальные.
- **По степени физической реализации:** логические и физические.
- **По отображению динамики:** статические и динамические.
- **По отображаемому аспекту:** функциональные, информационные, поведенческие, компонентные и смешанные.

Процесс формирования информационной модели проходит стадии от изучения системы до её моделирования и последующего сопровождения.

4. Жизненный цикл и стадии разработки программного обеспечения.

Жизненный цикл ПО (ЖЦ ПО) — это период времени, который начинается с момента принятия решения о необходимости создания ПО и заканчивается в момент его полного изъятия из эксплуатации.
Основные этапы ЖЦ ПО:
1. **Формирование требований к системе (планирование):** определение того, что должна делать система.
2. **Проектирование:** разработка архитектуры и структуры.
3. **Реализация (кодирование):** написание программного кода.
4. **Тестирование:** проверка работоспособности и поиск ошибок.
5. **Ввод в действие:** передача системы заказчику.
6. **Эксплуатация и сопровождение:** поддержка системы в рабочем состоянии.

Для каждого этапа строго определяются состав работ, получаемые результаты, роли участников и используемые методы. ЖЦ регламентируется стандартами, такими как **ГОСТ 34.601-90**, **ISO/IEC 12207:1995**, а также методологиями **RUP**, **MSF** и **XP**.

5. Модели жизненного цикла разработки ПО.

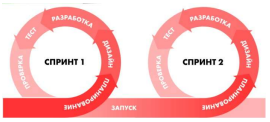
Модель ЖЦ — это структура, определяющая последовательность выполнения и взаимосвязи процессов и задач на протяжении всего цикла.
В материалах выделены три **классические модели:**
1. **Каскадная (водопадная) модель:** характеризуется строго последовательным переходом по этапам. Следующий этап начинается только после полного завершения предыдущего.
2. **Поэтапная модель с промежуточным контролем (итерационная):** в отличие от каскадной, допускает возвраты на предыдущие этапы при обнаружении несоответствий или ошибок в ходе контроля.
3. **Спиральная модель:** делает упор на анализ рисков. Разработка идет витками, на каждом из которых создается прототип системы, уточняются требования и планируется следующий виток.

Также упоминаются современные стратегии конструирования: **однотактные** и **инкрементные** (итеративные).

6. Методологии проектирования ИС.

Основные характеристики методологий:
1. Стратегия конструирования.
2. Адаптивность процесса.
3. Состав этапов модели ЖЦ и связей между ними.
4. Формулировка требований.

Стратегии конструирования: однотактные, инкрементные (итеративные), эволюционные.
Agile Model – семейство процессов разработки, а не единственный подход в разработке программного обеспечения, определяемый Agile Manifesto.



Agile не включает практик, а определяет ценности и принципы, которыми руководствуются команды.
Kanban – это метод управления задачами, основанный на визуализации работы на доске с колонками (например, «Сделать», «В работе», «Готово») и ограничении количества задач, которые одновременно находятся в работе.
Ключевая идея: не начинать новую работу, пока не завершена текущая (ограничение WIP — Work In Progress).
Принципы DSDM (Dynamic System Development Method):
- Пользователи или заказчик участвуют в процессе разработки.

- Команда самостоятельно принимает решения по проекту.
- Работа делится на спринты.
- Результат должен быть полезен бизнесу.
- Используется итеративный и инкрементный подход.
- Возможность отката действий.
- Постоянное тестирование и использование обратной связи.

Модель SCORE: S – симптомы C – причины O – результаты R – ресурсы E – эффекты

7. Подходы к выбору методологии для разработки ПО.

Принцип 1. Большая по размерам методология нужна тогда, когда в проекте занято большое число разработчиков.

Принцип 2. Большая корректность методологии нужна тогда, когда скрытые ошибки могут повлечь значительный ущерб.

Принцип 3. Незначительное увеличение размеров или плотности методологии ведёт к существенному увеличению стоимости проекта.

Принцип 4. Наиболее эффективная форма коммуникации – непосредственное взаимодействие лицом к лицу.

8. Базовые требования ИС: календарное планирование и техническое задание.

Результаты планирования (основные документы):

- **Календарный план выполнения работ** – регламентирует состав, сроки и финансирование работ.
- **Техническое задание (ТЗ)** – отражает основные требования к системе; является основным документом, определяющим требования и порядок создания (развития или модернизации) системы.

Вариант содержания ТЗ:

- общие сведения о системе;
- назначение и цели создания ИС;
- характеристика объектов автоматизации;
- требования к ИС в целом, к функциям и обеспечению;
- состав и содержание работ по созданию системы;
- порядок контроля и приемки системы;
- требования к составу и содержанию работ по подготовке объекта автоматизации к вводу системы в действие;
- требования к документированию;
- источники разработки.

9. Организация разработки ИС. Стандартизация проектирования.

Проектирование – процесс перехода от одной модели в виде первичного описания системы (ТЗ) к ее описанию в виде набора стандартных документов, достаточных для создания ИС (проектной документации)
Стандарты, регламентирующие ЖЦ ПО:
- ГОСТ 34.601-90
- ISO/IEC 12207:1995
- Custom Development Method (CDM – методика Oracle)
- Rational Unified Process (RUP)
- Microsoft Solution Framework (MSF)
- Extreme Programming (XP)

10. Основные принципы проектирования ИС.
1. Принцип декомпозиции ("разделяй и властвуй")
2. Принцип иерархического упорядочения
3. Принцип концептуальной общности
4. Принцип абстрагирования
5. Принцип формализации
6. Принцип унификации
7. Принцип логической независимости
8. Принцип многомодельности
9. Принцип неизбыточности (согласованности)
10. Принцип информационной закрытости (инкапсуляции)
11. Принцип полиморфизма.

11. Подходы к проектированию ИС. Основные концепции каждого подхода.

1. **Структурное моделирование:**
   - Сущность: расчленение системы на части («черные ящики») и их иерархическая организация; метод исследования статических характеристик системы путем выделения подсистем и элементов различного уровня иерархии.
   - Концепция: функциональная декомпозиция (система = иерархия функций), данные хранятся отдельно от сущности, иерархия модулей = отношение композиции.
2. **Объектно-ориентированное моделирование:**
   - Сущность: декомпозиция системы на классы, которые соответствуют однотипным объектам предметной области, и построение из них иерархии с использованием отношений композиции и наследования.
   - Концепция: объектная декомпозиция (система = набор объектов), объект = атрибутивные данные + методы, иерархия отношений = композиции и наследования.

12. Структурный подход к проектированию ИС. История развития и основная сущность подхода. Основные методологии анализа проектирования.
Структурный подход основан на расчленении анализируемой системы на части и их иерархические организации.

Основные положения структурного подхода (в моделировании бизнес-процессов):

1. Разбиение исследуемого процесса на функциональные блоки — подпроцессы.
2. Возможность детализации любых процессов путем иерархической декомпозиции.
3. Использование для описания процесса графических нотаций с возможностью текстового разъясняющего дополнения.

Схема применения структурного подхода:

Методологии структурного анализа и проектирования:

- SADT (функциональная модель)
- DFD (функциональная или компонентная)
- ERD (информационная)
- Flowcharts (поведенческая)
- EPC (функциональная или поведенческая)
- BPMN (функциональная или поведенческая)

13. Методология структурного анализа и проектирования. Семейство IDEF.

IDEF (Integration Definition Methodology) — семейство совместно используемых методов для процесса моделирования.

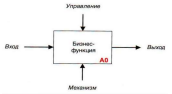
Состав семейства IDEF:

- **IDEF0 (Function Modeling)** – метод для создания функциональной модели (структурированное отображение функций системы, информации и объектов).
- **IDEF1 (Information Modeling)** – метод для построения информационной модели (структурированная информация для поддержки функций).
- **IDEF2 (Simulation Model Design)** – метод для построения динамической модели поведения функций, информации и ресурсов во времени.
- **IDEF3 (Process Description Capture)** – метод для сбора информации о состоянии системы (причинно-следственные связи и события).

14. Применение методологии IDEF0 для моделирования предметной области. Элементы графической нотации IDEF0

IDEF0 — методология функционального моделирования и графическая нотация для формализации и описания бизнес-процессов.
Базовые принципы:
1. Принцип функциональной декомпозиции (сложная функция может быть представлена в виде совокупности составляющих ее более простых функций)
2. Принцип ограничения сложности (2-6 блоков на диаграмме)
3. Принцип контекста (моделирование начинается с построения контекстной диаграммы)

Элементы нотации:
Функциональный блок: Прямоугольник, обозначающий конкретную бизнес-функцию.
Стрелки (Интерфейсные дуги):
Вход (левая сторона): Материал или информация, которые преобразуются



работой.
Управление (верхняя сторона): Правила, стратегии, стандарты. Каждый блок должен иметь хотя бы одну стрелку управления.
Выход (правая сторона): Результат выполнения функции.
Механизм (нижняя сторона): Ресурсы (персонал, станки, бюджет).
Вызов (нижняя сторона, стрелка вниз): Ссылка на другую модель для организации коллективной работы.
Типы взаимосвязей между блоками:

- По управлению
- По входу
- Обратная связь по управлению
- Обратная связь по входу
- Взаимосвязь "выход-механизм"

15. Моделирование данных. Общие сведения и назначение модели IDEF1

IDEF1 (Information Modeling) — метод применяется для построения информационной модели, которая представляет собой структурированную информацию, необходимую для поддержки функций производственной системы или среды.
Метод IDEF1 позволяет построить модель данных, эквивалентную реляционной модели в третьей нормальной форме. (короче просто как схема БД).

16. Применение методологии IDEF3 для моделирования предметной области. Элементы графической нотации IDEF3

IDEF3 — методология описания процессов, происходящих в системе (описывает логику выполнения действий). Используется самостоятельно или вместе с IDEF0. Единица описания – диаграмма (workflow diagram)

Методы IDEF3:
1. **PFD (Process Flow Description)** — описание процессов с описанием того, как организована работа между элементами.
2. **OSTD (Object State Transition Description)** — описание переходов состояний объектов.

Каждая работа в IDEF3 описывает какой-либо сценарий бизнес-процесса и может являться составляющей другой работы.

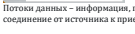
Компоненты диаграммы:
- **Работы (boxes, activities):** прямоугольник, имя (глагол или отглагольное существительное), уникальный номер.
- **Связи – показывают взаимоотношение работ**

- **Перекрёстки (Junctions):**
  - **Fan-in (разворачивающие):** разбиение потока (завершение одного действия вызывает начало нескольких).
- **Fan-out (сворачивающие):** объединение потоков (завершение одного или нескольких действий вызывает начало другого).
- **Типы соединений:** «и», «эксклюзивное "или"», «или» (каждый может быть разворачивающимся или сворачивающимся).

17. Методология функционального моделирования. Применение методологии DFD для описания потоков данных предметной области

DFD (Data Flow Diagrams) — диаграммы потоков данных; иерархия функциональных процессов, связанных потоками данных. Один из основных инструментов структурного анализа и проектирования.

Основные компоненты DFD:
- Внешние сущности (источник или приемник информации) – изображаются в виде прямоугольника с тенью, располагаются по краям диаграммы
- Системы и подсистемы
- Процессы (или работы) – преобразование входных данных в выходные по алгоритму
- Накопители данных (хранилища) — абстрактное устройство для хранения информации



- Потоки данных – информация, передаваемая через некоторое соединение от источника к приемнику (изображаются стрелками)

- Поскольку в DFD каждая сторона работы не имеет четкого назначения, как в IDEF0, стрелки могут подходить и выходить из любой грани прямоугольника работы.
- В DFD также применяются двунаправленные стрелки для описания диалогов типа «команда-ответ» между работами, между работой и *внешней сущностью* и между *внешними сущностями*.

Назначение: построение иерархии DFD для того, чтобы сделать описание системы ясным и понятным на каждом уровне детализации, разбить его на части с точно определенными отношениями между ними.
Основные правила:
1. Все потоки данных перемещаются между хранилищами через процессы.
2. DFD не содержит логических операторов.
3. Потоки данных должны быть названы и описаны в словаре данных.
4. Не важно, с какой стороны стрелка входит в блок.
5. Проектирование начинается с создания контекстной диаграммы (уровень «черного ящика»).

18. Понятие CASE-технологии и её составляющие. CASE-инструменты.

CASE-технология — это методология проектирования ИС, а также набор методов, нотаций и инструментальных средств, которые позволяют в наглядной форме моделировать предметную область, анализировать модель ИС на всех этапах её ЖЦ и разрабатывать приложения.

Основные цели:
- Максимальная автоматизация стадий анализа и проектирования ИС для построения формальных и непротиворечивых моделей.
- Вынесение части деятельности из стадии кодирования в стадию проектирования.

Инструменты для создания DFD-диаграмм (SoftDesign_1_08, Слайд 20):
- Ramus
- MS Visio или Draw.io
- StarUML или BPwin
- Lucidchart (облачное решение для командной работы)

19. Объектно-ориентированное моделирование. Основные понятия в ООП. Базовые составляющие ООП.

Сущность ООП: декомпозиция системы на классы, которые соответствуют однотипным объектам предметной области, и построение из них иерархии в виде ориентированного графа с использованием отношений композиции и наследования.
Базовые понятия и составляющие:
- **Класс:** шаблон для создания объектов, описывающий их свойства и поведение. На диаграммах изображается прямоугольником, разделенным на три части: имя, атрибуты и операции.
- **Объект:** конкретный экземпляр класса.
- **Структура системы:** представляется в виде ориентированного графа, где иерархия строится на отношениях композиции и наследования.
- **Отношения между элементами:**
  - **Зависимость (dependency):** изменение одного элемента может повлиять на другой.
  - **Ассоциация:** связь между классами с указанием кратности (например, 1, 0..1, *).
  - **Обобщение (generalization):** отношение наследования от предка к потомку.
- **Ключевые принципы (из общих принципов проектирования):** инкапсуляция (закрытость данных), наследование и полиморфизм.

20. Методологии, поддерживающие ООП. Унифицированный процесс

Методологии, поддерживающие ООП:

- Унифицированный процесс (Unified Process, UP)
- Экстремальное программирование (eXtreme Programming, XP)
- Гибкое моделирование (Agile Modeling, AM)

Унифицированный процесс (Unified Process) — это процесс разработки ПО, который обеспечивает упорядоченный подход к распределению задач и обязанностей в организации-разработчике.
Унифицированный процесс (Unified Process):
- Подразумевает построение нескольких видов диаграмм для описания разных аспектов системы.
- Для сложных систем строится множество диаграмм одного и того же вида для разной степени детализации

21. UML. История и назначение. Режимы использования

UML (Unified Modeling Language) — стандартная нотация визуального моделирования программных систем, принятая консорциумом OMG осенью 1997 года.

Три режима использования UML разработчиками:

1. **Режим эскиза:** создание набросков для быстрого обсуждения идей.
2. **Режим проектирования:** детальное моделирование, включающее **прямую разработку** (forward-engineering — генерация кода по модели) и **обратную разработку** (reverse-engineering — восстановление модели из кода).
3. **Режим языка программирования:** использование исполняемого UML (**Executable UML**).

22. Структура UML. Семантика и синтаксис. Понятие нотации. Понятие UML-диаграммы. Взаимосвязи между диаграммами

Общая структура UML

UML (Unified Modeling Language) — стандартная нотация визуального моделирования программных систем, принятая консорциумом OMG осенью 1997 года.
Семантика определяет **смысл (значение)** моделей UML.
Семантика описывает, что именно означает та или иная конструкция языка.
Для UML семантика — это отображение элементов модели на некоторое множество возможных реализаций (реальных систем)

Пример конкретных связей между диаграммами [10]:

Диаграмма	Связана с	Тип связи
Диаграмма последовательности	Диаграмма классов	Объекты соответствуют классам, сообщения — операциям
Диаграмма состояний	Диаграмма классов	Диаграмма описывает поведение одного класса
Диаграмма компонентов	Диаграмма развёртывания	Компоненты размещаются на физических узлах
Диаграмма деятельности	Диаграмма вариантов использования	Деятельность реализует сценарий прецедента
Композиция модели	Модель собирается из нескольких диаграмм разных типов, которые вместе дают полное описание	

НОТАЦИЯ – установленные способы отображения элементов системы (графы, таблицы, блок-схемы, формальные и естественные языки).
ПАРАДИГМА – исходная концептуальная схема (модель) постановки проблемы и ее решения.

23. Классификация UML-диаграмм. Виды моделей ИС и их связь с UML-диаграммами

Структурные: классов, пакетов, развертывания и др. Диаграммы поведения: вариантов использования, последовательности, деятельности, состояний и др. Всего UML 2 содержит 13 официальных типов диаграмм:

1. Деятельности – процедурное и параллельное поведение
2. Классов – класс, свойства и отношения
3. Взаимодействия - Взаимодействие между объектами; акцент на связях
4. Компонентов - Структура и взаимосвязи между компонентами
5. Составных структур - Декомпозиция класса во время выполнения
6. Развертывания - Развертывание артефактов в узлы
7. Обзора взаимодействия - Комбинация диаграммы последовательности и диаграммы деятельности
8. Объектов - Вариант конфигурации экземпляров
9. Пакетов - Иерархическая структура времени компиляции
10. Последовательности - Взаимодействие между объектами; акцент на последовательности
11. Конечных автоматов - Как события изменяют объект в течение его жизни
12. Временная - Взаимодействие между объектами; акцент на синхронизации
13. Прецедентов - Как пользователи взаимодействуют с системой

ДИАГРАММА		НАЗНАЧЕНИЕ	МОДЕЛИ ИС		
			По степени физической реализации	По отображению динамики	По отображению аспекту
Вариантов использования (use case)		Отображает функции ИС, взаимодействия между актерами и функциями	Логическая	Статическая	Функциональная
Классов (class)		Отображает набор классов, интерфейсов и отношений между ними	Логическая или физическая	Статическая	Функционально-информационная
Пакетов (package)		Отображает набор пакетов и отношений между ними и процесс их изменения цикл	Логическая или физическая	Статическая	Компонентная
Поведения (behavior)	Автоматов (state machine)	Отображает состояния сущности и переходы между ними в процессе ее изменения цикл	Логическая	Динамическая	Поведенческая
	Деятельности (activity)	Отображает бизнес-процессы в системе (описание алгоритмов поведения)			
	взаимодействия: Последовательности (sequence)	Отображает последовательность передачи сообщений между объектами и актерами			
	Коммуникации (communication)	Аналогична диаграмме последовательности, но основной акцент делается на структуру взаимодействия между объектами			
Реализации (implementation)	Компонентов (component)	Отображает компоненты ИС (программы, библиотеки, таблицы и т.п.) и связи между ними	Физическая	Статическая	Компонентная

При разработке отдельной модели системы в Унифицированном процессе строят несколько видов диаграмм. При разработке модели сложной системы, как правило, строят несколько диаграмм одного и того же вида. Можно не создавать отдельные виды диаграмм, если в этом нет необходимости. Часть диаграмм после их построения требует развития и уточнения в рамках разработки следующей модели (технологического процесса).

24. Диаграммы поведения. Общие сведения, назначение и взаимодействие поведенческих диаграмм

Диаграммы поведения (Behavior Diagrams) описывают то, как система функционирует и взаимодействует с пользователями или внешней средой.

- **Основные типы:**
  1. **Диаграмма прецедентов (Use Case):** определяет функциональное назначение (что делает система).
  2. **Диаграмма деятельности (Activity):** описывает логику процедур, потоки работ и параллельное поведение.
  3. **Диаграмма состояний (State Machine):** отображает состояния сущности и переходы между ними под влиянием событий (триггеров).
  4. **Диаграммы взаимодействия:** фокусируются на обмене сообщениями между объектами. Например, **диаграмма последовательности** (Sequence diagram) подчеркивает временную упорядоченность сообщений.

Взаимодействие: Эти диаграммы дополняют друг друга: прецеденты задают общие требования, деятельность детализирует алгоритм их выполнения, а диаграммы состояний описывают внутреннюю логику отдельных объектов.

25. Назначение и состав диаграммы вариантов использования. Примеры

Диаграмма вариантов использования описывает функциональное назначение системы (как система будет использоваться в процессе своего функционирования). Основные элементы: актёр, прецедент, подсистема.

Назначение: спецификация внешних требований к системе и определение того, как она будет использоваться.

Состав нотации:

Актеры (Actors): роли, которые пользователи или другие системы играют по отношению к ИС.

Прецеденты (Use Cases): Цель варианта использования заключается в том, чтобы определить законченный аспект или фрагмент поведения некоторой сущности без раскрытия внутренней структуры этой сущности. С системно-аналитической точки зрения варианты использования могут применяться как для спецификации внешних требований к проектируемой системе, так и для спецификации функционального поведения уже существующей системы.

Рамки системы: границы моделируемой подсистемы.

Связи: ассоциация (uses) – вид: (сплошная линия) – элементы связи: актер-прецедент. Обобщение – вид: (---------|>) - элементы связи: актер-актер

Включить (include), расширить (extend) – вид: (--------->) - элементы связи: прецедент-прецедент.

Пример построения: выделяем необходимые роли в системе, выделяем «внешние» функции системы (прецеденты).

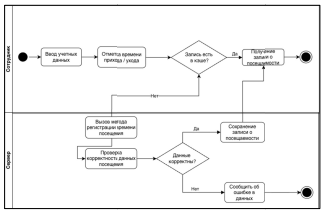
26. Назначение и состав диаграммы деятельности. Примеры нотаций

Диаграмма деятельности (Activity diagram) — это технология, предназначенная для описания логики процедур, бизнес-процессов и потоков работ. Она позволяет визуализировать параллельное и процедурное поведение системы.

Состав нотации:

- **Начальный узел (initial node):** точка начала выполнения процесса.
- **Операции (action):** основные шаги или действия в рамках деятельности.
- **Ветвление (fork) и Объединение (join):** используются для описания параллельно выполняемых потоков работ.
- **Решение (decisions) и Слияние (merges):** используются для описания условного ветвления процесса (логика «если… то»).
- **Конечный узел (final node):** завершение конкретного потока или всей деятельности.
- **Ребра (потоки):** стрелки, соединяющие узлы и показывающие последовательность действий.

Пример: В лекциях приведена диаграмма процесса доставки заказа («Deliver Order»), где после принятия заказа поток разделяется на две ветви: срочная доставка («Rush Order») и обычная доставка («Regular Delivery»), которые затем сливаются в общий узел завершения.



27. Диаграмма состояний. Назначение и состав диаграммы. Примеры нотаций

Отображает состояния объекта и переходы между ними в течение жизненного цикла. Элементы: начальное состояние (initial pseudostate), состояния (states), переходы (transition), конечное состояние (final state). **Состав нотации:**

- **Начальное псевдосостояние (initial pseudostate):** точка входа в жизненный цикл.
- **Состояния (states):** условия или ситуации в жизни объекта, во время которых он выполняет определенную деятельность.
- **Переход (transition):** перемещение из одного состояния в другое под влиянием триггера.
- **Конечное состояние (final state):** завершение жизненного цикла объекта.

Метки переходов и активности:

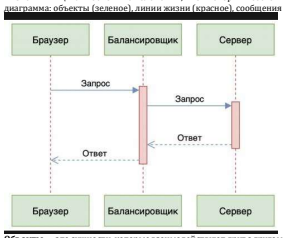
- Каждый переход может иметь метку: **Триггер [Защита] / Активность** («событие [сторожевое условие] / действие»).
- Внутри состояния могут быть прописаны внутренние активности: **entry**/ (входная), **do**/ (выполняемая в состоянии), **exit**/ (выходная).

РАЗЛИЧАЮТ ДВА ВИДА ПЕРЕХОДОВ: НЕТРИГГЕРНЫЙ И ТРИГГЕРНЫЙ.

- Переход первого вида, называемый также переходом по завершении, срабатывает неявно, когда все основные операции (с метками entry, do и exit) в исходном состоянии успешно завершают свою работу. Данный вид перехода обозначается стрелкой без надписи.
- Для наступления триггерного перехода необходимо наступление некоторого события, которое записывается над стрелкой.
- Переход также может быть рефлексивным, т.е. направлен в то же состояние, из которого он выходит.

28. Диаграмма взаимодействия. Назначение и состав диаграммы последовательности. Примеры нотаций

Диаграмма последовательности используется для визуализации взаимодействия между объектами в системе. Она показывает порядок сообщений, которыми объекты обмениваются друг с другом во времени и их жизненный цикл. Основные элементы, из которых состоит sequence-диаграмма: объекты (зеленое), линии жизни (красное), сообщения (синее).



Объекты — это сущности, которые взаимодействуют друг с другом. Например, пользователь, топик или очередь, микросервис. Помимо названия, объекты могут еще отличаться своим типом:

Линии жизни (lifelines) отображают течение времени, а также активность или исполнение определенных функций объекта. Представляют из себя вертикальную линию с блоком активности.

Сообщения (messages) показывают обмен информацией между объектами. Например, клик пользователя на кнопку, но чаще это запрос или ответ, вызов хранимой процедуры или отправка сообщения. Сообщения отображаются в виде стрелок, направленных от одной линии жизни к другой, и бывают нескольких типов.

29. Структурные диаграммы. Общие сведения, назначение и взаимодействие структурных диаграмм

Предназначены для описания статической структуры системы. Основные виды: диаграммы классов, пакетов, компонентов и развертывания.

Структурные диаграммы - описывают структуру сложных объектов и систем, показывают статическую структуру системы и ее частей на разных уровнях абстракции и реализации, а также их взаимосвязь

К структурным диаграммам относят следующие 7 типов диаграмм:

1. Диаграмма составной структуры
2. Диаграмма развертывания
3. Диаграмма пакетов
4. Диаграмма профилей
5. Диаграмма классов
6. Диаграмма объектов
7. Диаграмма компонентов

30. Диаграммы классов и объектов. Назначение и состав диаграмм. Примеры

Диаграмма описывает типы объектов системы и различного рода статические отношения, которые существуют между ними. Нотации: Прямоугольники на диаграмме представляют классы и разделены на три части: • имя класса, • его атрибуты • и его операции.

СВЯЗИ НА ДИАГРАММЕ КЛАССОВ: Считается, что между двумя элементами существует ЗАВИСИМОСТЬ (dependency), если изменения в определении одного элемента (сервера) могут вызвать изменения в другом элементе (клиенте).

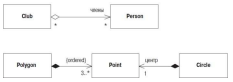
Ассоциация: связь между классами с указанием кратности

Агрегация – один элемент может существовать без другого, композиция – наоборот.

Зависимость (dependency): изменение одного элемента может повлиять на другой

Обобщение (generalization): наследование свойств родительского класса наследником

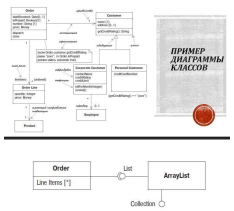
АГРЕГАЦИЯ И КОМПОЗИЦИЯ



Кратность свойства обозначает количество объектов, которые могут заполнять данное свойство. n, n..m (n..*), * Чаще всего встречаются следующие кратности: 1, 0..1 , *

ВИДЫ СВЯЗЕЙ НА ДИАГРАММЕ КЛАССОВ:

- Зависимость
- Обобщение
- **Ассоциация**
  - Агрегация
  - Композиция



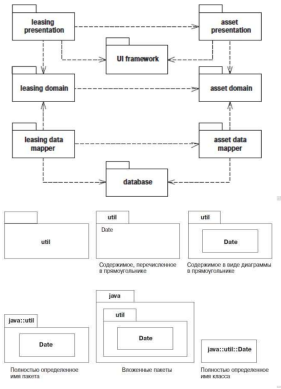
НОТАЦИЯ ИНТЕРФЕЙСОВ (БЕЗ ИХ РЕАЛИЗАЦИИ)

ПРИМЕЧАНИЯ – ЭТО КОММЕНТАРИИ НА ДИАГРАММАХ.

31. Диаграмма пакетов. Назначение и состав диаграмм. Примеры нотаций

Отображает набор пакетов и отношения между ними. Используется для представления иерархической структуры системы на этапе компиляции.

Состав:

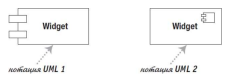


32. Назначение и состав диаграммы компонентов. Примеры нотаций

Показывает разбиение программной системы на компоненты и зависимости между ними. Компонент является физической программной сущностью системы.

Показывает разбиение программной системы на структурные компоненты и связи (зависимости) между компонентами.

ДИАГРАММА КОМПОНЕНТОВ ПОЗВОЛЯЕТ ОПРЕДЕЛИТЬ СОСТАВ ПРОГРАММНЫХ КОМПОНЕНТОВ, В РОЛИ КОТОРЫХ МОЖЕТ ВЫСТУПАТЬ ***ИСХОДНЫЙ, БИНАРНЫЙ И ИСПОЛНЯЕМЫЙ КОД.***



33. Модель физической реализации. Назначение и состав диаграммы развертывания. Примеры нотаций

Используется для отображения физической структуры системы. Основные элементы: артефакт и узел (node). Узлы бывают двух типов: устройство (device) и среда выполнения (execution environment). Диаграмма позволяет показать размещение компонентов на узлах и физические каналы связи.

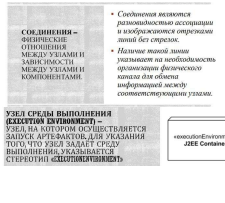
ОСНОВНЫЕ ЦЕЛИ РАЗРАБОТКИ ДИАГРАММЫ РАЗВЕРТЫВАНИЯ

Состав:

АРТЕФАКТ – НЕКАЯ ФИЗИЧЕСКАЯ СУЩНОСТЬ, ПРОГРАММНЫЙ КОМПОНЕНТ, КОТОРЫЙ ИСПОЛЬЗУЕТСЯ ИЛИ СОЗДАЁТСЯ ВО ВРЕМЯ РАБОТЫ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ.

УЗЕЛ (NODE) – НЕКОТОРЫЙ ФИЗИЧЕСКИ СУЩЕСТВУЮЩИЙ ЭЛЕМЕНТ СИСТЕМЫ, ОБЛАДАЮЩИЙ ОПРЕДЕЛЕННЫМ ВЫЧИСЛИТЕЛЬНЫМ РЕСУРСОМ.

- Узлы бывают двух типов:
- Устройство (device) – это физическое оборудование: компьютер или устройство, связанное с системой.
- Среда выполнения (execution environment) – это программное обеспечение, которое само может включать другое программное обеспечение, например операционную систему или процесс-контейнер.



34. Механизмы расширения UML диаграмм. Основные подходы и методы.

Механизмы расширения UML — это встроенные средства, позволяющие адаптировать стандартный язык моделирования под требования конкретной предметной области, специфику технологии или методологию разработки без изменения базового мета-языка.

UML поддерживает три основных механизма расширения:

1. Стереотипы (Stereotypes)

Позволяют создавать новые типы строительных блоков на основе существующих.

Назначение: Расширяют словарь языка. Стереотип меняет семантику (смысл) элемента, хотя его графическое представление может оставаться прежним, но с добавлением специального значка.

Обозначение: Имя стереотипа заключается в «кавычки-елочки» (например, «entity», «control» или «boundary» на диаграмме классов).

2. Помеченные значения (Tagged Values)

Служат для добавления новой метаинформации (атрибутов) к существующим элементам модели.

Назначение: Позволяют привязать к элементу пары «ключ — значение» для хранения административной или технической информации, которая не влияет на логику модели, но важна для генерации кода или управления конфигурацией (например, версия, автор, язык программирования).

Обозначение: Записываются в фигурных скобках под именем элемента, например {version = 2.4, author = Иванов}.

3. Ограничения (Constraints)

Используются для определения специфических правил и условий, которым должны следовать элементы модели.

Назначение: Уточняют семантику, задавая условия истинности (предусловия, постусловия) или инварианты, которые не могут быть нарушены.

Обозначение: Правило записывается в фигурных скобках (...). Для формального и строгого описания ограничений обычно используется язык OCL (Object Constraint Language).

Профиль UML (UML Profile)

Для удобного управления созданным набором стереотипов, помеченных значений и ограничений используется понятие профиля. Профиль группирует расширения для определенной платформы (например, профиль для моделирования баз данных или для Enterprise Java Beans) и реализуется через диаграмму профилей.

35. Архитектура ИС. Классификация ИС по их архитектуре.

Понятие архитектуры ИС — это фундаментальная организация системы, выраженная в её компонентах, их взаимосвязи друг с другом и с внешней средой, а также принципы, определяющие её проектирование и развитие. Простыми словами, это «скелет» системы, определяющий, как она будет работать и взаимодействовать с пользователями и другими системами.

Классификация по архитектуре. Лекции не содержат явной классификации, но, опираясь на общие принципы проектирования (например, «принцип декомпозиции» и «иерархического упорядочения»), можно выделить следующие основные типы архитектур:

Одноуровневая (монолитная) архитектура: Все компоненты системы (интерфейс, бизнес-логика, работа с данными) находятся на одном компьютере.

Двухуровневая архитектура («клиент-сервер»): Разделение на две части: клиентскую (отвечает за интерфейс и часть логики) и серверную (хранит данные и выполняет основные операции).

Многоуровневая (трёхуровневая и более) архитектура: Разделение системы на три и более независимых уровня:

Уровень представления (клиент): Пользовательский интерфейс.

Уровень бизнес-логики (сервер приложений): Обрабатывает правила и логику работы.

Уровень доступа к данным (сервер БД): Хранит данные.

Этот подход вытекает из «принципа декомпозиции» , разделяющего сложную систему на части.

36. Базовые архитектурные шаблоны ИС (на примере веб-ориентированных систем). Архитектура тонкого клиента, толстого клиента и веб-доставки.

Лекции описывают архитектуру на более абстрактном уровне. Однако, опираясь на понятия «внешней модели» и «технического обеспечения» , можно дать следующую характеристику:

Архитектура «Толстого клиента» (Fat Client): Основная нагрузка по обработке данных и логике ложится на компьютер пользователя (клиентскую часть). Сервер в основном используется для хранения данных.

«Техническое обеспечение» клиентской части должно быть мощным.

«Внешняя модель» (пользовательский интерфейс) тесно связана с логикой.

Архитектура «Тонкого клиента» (Thin Client): На компьютере пользователя запускается только минимальный набор функций для отображения информации (например, веб-браузер). Вся обработка данных и логика выполняется на сервере.

Соответствует «принципу абстрагирования», где пользователь взаимодействует с упрощённым интерфейсом, не вникая в сложность серверных процессов.

Архитектура веб-доставки: Является вариацией «тонкого клиента» для веб-среды. Система предоставляется пользователю как набор веб-страниц, генерируемых сервером. Эта архитектура реализует «принцип многомодельности» , так как позволяет работать с системой с различных устройств через единый интерфейс — браузер.

37. Модульная архитектура. Виды декомпозиций и их возможности.

Этот вопрос напрямую опирается на лекции.

Модульная архитектура — это подход, при котором система строится из отдельных, логически завершённых и слабо связанных блоков — модулей. Это прямая реализация «принципа декомпозиции» («разделяй и властвуй») .

Виды декомпозиций и их возможности:

Функциональная декомпозиция: Разбиение системы по выполняемым функциям.

Возможности: Позволяет создавать функциональные модели (например, в нотации IDEF0) , где сложная бизнес-функция (контекстная диаграмма A-0) последовательно раскладывается на более простые (диаграммы A0, A1, A2 и т.д.). Это помогает структурировать процесс «как есть» (AS-IS) и «как должно быть» (TO-BE) .

Информационная декомпозиция: Разбиение по информационным объектам и их связям.

Возможности: Позволяет строить информационные модели (например, в нотации IDEF1) , что необходимо для проектирования базы данных.

Поведенческая декомпозиция: Разбиение по событиям и состояниям системы.

Возможности: Описывает логику выполнения действий. Методология IDEF3 как раз предназначена для этого, описывая технологические процессы (PFD) и переходы состояний объектов (OSTD).

38. Понятие «типовое решение». Типовое проектное решение. Архитектурные шаблоны.

Лекции не дают прямого определения, но концепция логически вытекает из них.

Типовое (проектное) решение — это готовое, многократно используемое архитектурное или дизайнерское решение для определённого класса задач. Это практическая реализация «принципа унификации» , призванная сократить время и стоимость разработки за счёт использования проверенных схем.

Архитектурный шаблон — это общее, повторно используемое решение часто встречающейся проблемы в архитектуре программного обеспечения в заданном контексте. Это более абстрактное понятие, чем типовое решение. Примеры: шаблоны «Клиент-сервер», «Многоуровневая архитектура», «Модель-Представление-Контроллер» (MVC).

39. Паттерны проектирования. Виды паттернов проектирования. Основные цели и задачи их применения.

Лекции не затрагивают эту тему, но она является краеугольным камнем современной разработки.
Паттерны проектирования (Design Patterns) — это типовые, проверенные способы решения часто встречающихся проблем при проектировании классов и объектов на уровне исходного кода. Это более «мелкие» решения, чем архитектурные шаблоны.
Основные цели применения:
Ускорить разработку, предлагая готовые, отлаженные решения.
Повысить читаемость и поддерживаемость кода, так как паттерны являются стандартным «языком» разработчиков.
Снизить количество ошибок, которые могут возникнуть при «изобретении велосипеда».
Виды паттернов (по книге «Банды четырёх» — GoF):

Порождающие (Creational): Решают проблемы создания объектов (например, Singleton, Factory, Builder).
Структурные (Structural): Описывают, как классы и объекты объединяются в более крупные структуры (например, Adapter, Decorator, Facade).
Поведенческие (Behavioral): Определяют эффективные механизмы взаимодействия между объектами и распределения ответственности (например, Observer, Strategy, Command).
40. Архитектурные шаблоны MV*.
Лекции не содержат этой информации. Архитектурные шаблоны семейства MV+ строятся вокруг разделения данных (Model), их представления (View) и логики их обработки (последний компонент).
MVC (Model-View-Controller): Классический шаблон. Controller получает запрос от пользователя, работает с Model (бизнес-логика и данные) и выбирает View для отображения результата.
MVP (Model-View-Presenter): Эволюция MVC. Presenter выступает посредником между View и Model, полностью обрабатывая логику представления. View становится пассивным («тонким клиентом»).
MVVM (Model-View-ViewModel): Активно используется в технологиях с привязкой данных (data binding). ViewModel преобразует данные из Model в команды и свойства, с которыми автоматически синхронизируется View.
Эти шаблоны являются практической реализацией «принципа многомодельности» и «принципа логической независимости» , так как они разделяют модель данных (логику) и её отображение, позволяя изменять их независимо.

41. Архитектура корпоративных программных приложений. Современные подходы к организации ИС.
Лекции описывают классический подход (модели AS-IS, TO-BE, затем проектирование ИС). Современные же подходы эволюционировали в сторону большей гибкости:
Отход от больших монолитных релизов.
Сервис-ориентированная архитектура (SOA): Система строится как набор взаимодействующих через сеть сервисов.
Микросервисная архитектура: Эволюция SOA.
Событийно-ориентированная архитектура (EDA): Компоненты системы реагируют на события, что повышает асинхронность и отказоустойчивость.
Бессерверные вычисления (Serverless).
42. Архитектура микросервисов (Microservices Architecture).
Это один из главных современных подходов.

Определение: Подход, при котором одно приложение строится как набор небольших, независимо развёртываемых сервисов. Каждый сервис реализует определённую бизнес-функцию.
Характеристики:
Независимость: Сервисы могут разрабатываться, деплоиться и масштабироваться разными командами с использованием разных технологий.
Ограниченный контекст: Каждый сервис владеет своей бизнес-логикой и базой данных.
Устойчивость: Отказ одного сервиса не приводит к падению всего приложения.
Связь с лекциями: Это высшая степень реализации «принципа декомпозиции» на уровне целого приложения.
43. Бессерверная архитектура (Serverless Architecture).
Определение: Модель развёртывания и исполнения кода, при которой разработчик не управляет серверами. Облачный провайдер (AWS Lambda, Google Cloud Functions) полностью берёт на себя запуск, масштабирование и обслуживание инфраструктуры.
Принцип работы: Разработчик загружает в облако функцию (кусок кода). Платформа запускает её только при наступлении определённого события (например, HTTP-запроса) и автоматически выделяет под неё столько ресурсов, сколько нужно.
Связь с лекциями: Это экстремальная степень «принципа абстрагирования» , где управление инфраструктурой полностью скрыто от разработчика, и он фокусируется исключительно на коде.

44. Понятие «рефакторинга». Методы рефакторинга.
Рефакторинг (Refactoring) — это процесс изменения внутренней структуры программного кода без изменения его внешнего поведения. Цель — улучшить читаемость, понятность и снизить сложность кода, облегчить его поддержку и последующее развитие.
Методы рефакторинга (основаны на книге Мартина Фаулера):
Извлечение метода (Extract Method): Превращение фрагмента кода в отдельный метод.
Переименование (Rename): Изменение названия переменной, метода, класса на более понятное.
Удаление дублирования (Remove Duplication): Объединение одинаковых фрагментов кода в одном месте.
Инкапсуляция поля (Encapsulate Field): Создание геттеров и сеттеров для прямого доступа к данным.

Замена условного оператора полиморфизмом (Replace Conditional with Polymorphism): Использование наследования вместо сложных if/else или switch.