ЛЕКЦИЯ 6

Тема: метод структурного и объектно-ориентированного проектирования
Цель: изучить основные методы проектирования.

Методики, используемые при проектировании, сначала программ, а затем и систем в целом, формировались в течение длительного промежутка времени.

Необходимость таких методик проявлялась при разработке сложных программных систем в условиях дефицита времени на разработку.

Большинство методик сначала были внутренними стандартами больших корпорации, а потом перешли на международный стандарт.

В основе наиболее известных методик проектирования ИС лежат два подхода:

1. **структурный**
2. **объектно-ориентированный.**

1. **Структурные методы анализа и проектирования** используют иерархические структуры для моделирования объекта исследования.

Структурное проектирование основано на алгоритмической декомпозиции, особое внимание в которой уделяется порядку происходящих событий.

Эти методы предназначены, в основном, для построения функциональных моделей и моделей данных разного уровня.

Как уже рассказывалось, есть главная цель и есть ее под разделы. Мы разделяем все части на под части.

2.Объектно-ориентированный подход основан на выделении агентов, которые являются либо субъектами действий, либо объектами действий. При объектно-ориентированной декомпозиции каждый объект обладает своим собственным поведением и каждый из них моделирует некоторый объект реального мира.

Выделяются актеры, описываются их свойства, действия которые они выполняют и уже потом с их помощью моделируют взаимосвязь всех объектов.

вряд ли удастся спроектировать сложную систему одновременно двумя способами, но можно применить их последовательно.

СТРУКТУРНЫЙ ПОДХОД

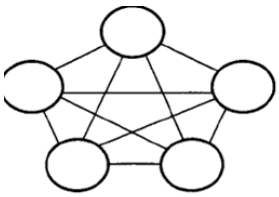
Структурный подход состоит в декомпозиции (разбиении) *системы на функциональные подсистемы, которые в свою очередь делятся на подфункции, подразделяемые на задачи*, и т.д. Процесс разбиения продолжается вплоть до конкретных процедур.

Все наиболее распространенные **структурные методы базируются на следующих принципах:**

1. - принцип разбиения сложной проблемы на множество меньших независимых задач, легких для понимания и решения;

Задачу разбиваем на не зависимые подзадачи которые можно выполнять параллельно или они не взаимосвязаны с друг другом логической цепочкой .

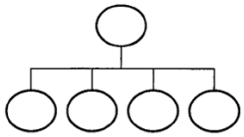
Вывести диалоговое окно приветствия, один разрабатывает структуру окна, другой рисует



окно.

2. - принцип организации составных частей в иерархические структуры.

Это когда есть разделение на модули не зависимые между собой, но они все идут от главной



задачи.

Модели используемые в структурном подходе::

1. **SADT** (Structured Analysis and Design Technique) – ***метод структурного анализа и проектирования***

модели и соответствующие функциональные диаграммы, объединенные данным названием;

2. **DFD** (Data Flow Diagrams) – ***диаграммы потоков данных***-

используются для описания структуры проектируемой системы

ERD (Entity-Relationship Diagrams) – *диаграммы* ***"сущность-связь"***

описания модели данных логического и физического уровней.

Интерпретация этих моделей зависит от стадии жизненного цикла разрабатываемого проекта.

SADT-модели используются для моделирования бизнес-процессов *на стадии формирования требований* к проектируемой ИС.

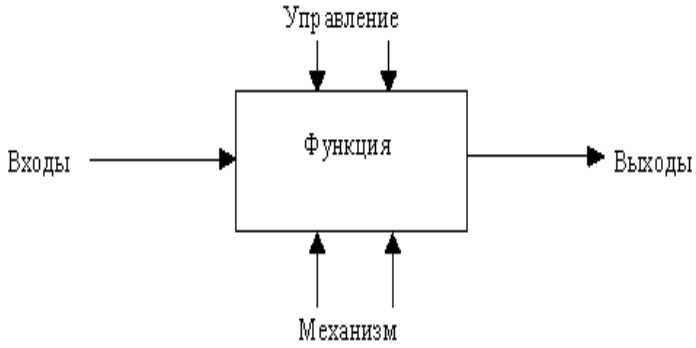
DFD-диаграммы *на стадии проектирования и анализа* ПО - отображения потоков данных, так же используются для описания структуры проектируемой системы.

ERD-диаграммы- *для описания данных на концептуальном уровне*, для описания модели данных *логического и физического уровней*.

Методология **SADT**(***Structured Analysis and Design Technique***)

концепция последовательной функциональной декомпозиции

концепция графического представления блочного моделирования.



Графика блоков и дуг SADT-диаграммы отображает функцию в виде блока, а интерфейсы входа/выхода представляются дугами, соответственно входящими в блок и выходящими из него.

.

механизм (человек или программа), который осуществляет операцию, представляется дугой, входящей в блок снизу.

ERD-ДИАГРАММЫ сущность-связь

Описание предметной области включает объекты, их свойства и отношения.

***модель сущность-связь строится с использованием трёх конструктивных элементов*:**

1.сущность,

2. атрибут

3. связь.

1. Сущность- это некоторая *абстракция (модель) реально существующего либо воображаемого объекта, процесса или явления*, имеющего существенное значение для рассматриваемой предметной области, информация о котором подлежит хранению.

Различают сущности:

1) ***Независимая сущность*** представляет *данные, которые всегда присутствуют в системе.*

Пример: Курица и петух -независимая сущность

2) ***Зависимая сущность*** представляет *данные, зависящие от других сущностей в системе.* Для существования зависимой сущности необходимо наличие сущностей, от которых она зависит.

Пример: Цыплята –зависимая сущность от курицы и петуха

3) ***Ассоциированная сущность*** представляет данные, которые связаны с отношением между сущностями.

Пример: Пасхальное яйцо

2. **Атрибут**– это поименованная характеристика сущности, являющаяся средством для описания её свойств, значимых для рассматриваемой предметной области Атрибут предназначен для квалификации, идентификации, классификации, количественной характеристики или выражения состояния сущности

Это то с помощью чего мы описываем свойства объекта или сущности

Пример: свойство цыплят : желтые, размером не более 100г

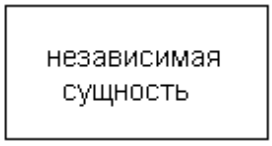
Связь между сущностями, моделирующая отношения, которые всегда существуют, пока существуют соединяемые ими объекты, называется ***неограниченной*** (обязательной) связью. ***Ограниченная*** (необязательная) связь моделирует условные отношения между объектами.

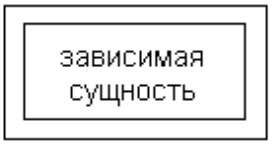
Виды связей:

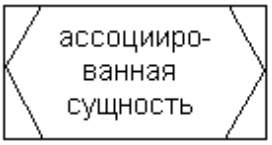
1. ***неограниченной*** (обязательной) 2. ***Ограниченная*** (необязательная)

связи классифицируют по типу отношения между экземплярами сущностей следующим образом:

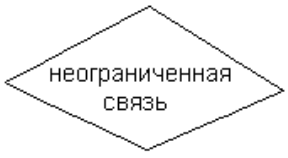
2. **Связь один-к-одному** (1:1): каждому экземпляру сущности А соответствует один экземпляр сущности В и наоборот (пример: студент - зачётка);
3. **Связь один-ко-многим** (1:M): одному экземпляру сущности А соответствует несколько экземпляров сущности В, и каждому экземпляру сущности В соответствует один экземпляр сущности А (пример: группа - студент);
4. **Связь многие-к-одному** (M:1): обратная по отношению к связи один-ко-многим (пример: студент - группа);
5. **Связь многие-ко-многим** (M:N): каждому экземпляру сущности А соответствует несколько экземпляров сущности В и наоборот (пример: студент - учебный предмет).





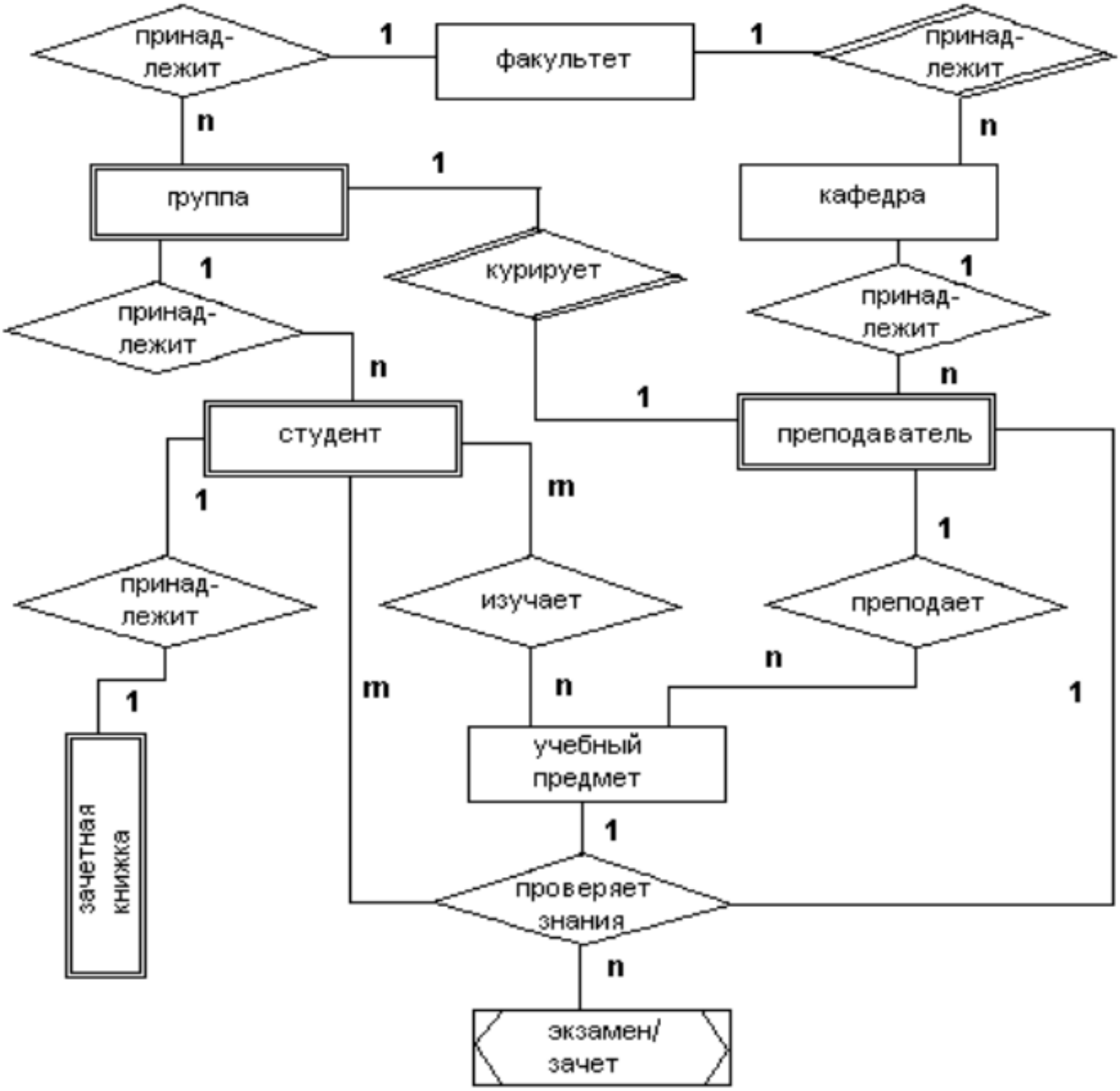


Для изображения ER-диаграммы в нотации Чена используются следующие графические примитивы:



Примитивы, изображающие сущности и связи, соединяются линиями, над которыми указывается тип связи.

Пример ER-диаграммы



Что такое программирование? Н. Вирт красноречиво определяет это понятие в заголовке своей книги: **Алгоритмы + Данные = Программы** [Вирт85, Вирт89]. Перефразируем это по-другому: **программная система - это набор механизмов для выполнения некоторых действий над некоторыми данными**.

Это означает, что имеется два ортогональных (но взаимодополняющих) способа посмотреть на организацию программы: в начальный момент мы можем сосредоточиться на функциях или на данных. **Основное**

различие между традиционными структурными методологиями проектирования и более новыми объектно-ориентированными методологиями находится в их первичном фокусировании: *Структурные методы* проектирования фокусируются на функциях системы: ***"Что она делает"***. *Объектно-ориентированные методы* фокусируются на данных (объектах) системы: ***"Что делается с..."***. Как мы увидим, этот, кажущийся очевидным, сдвиг фокуса радикально изменяет процессы проектирования, анализа и конструирования программного обеспечения. Объектная ориентация действительно может предоставить инструментальные средства, обеспечивающие более высокое качество программного обеспечения.

Наиболее важной частью объектно-ориентированного анализа является распределение обязанностей между классами (в виде операций классов). Оно выполняется с помощью диаграмм взаимодействия. При построении диаграмм взаимодействия возникают проблемы правильного распределения обязанностей между классами. Для их решения существует ряд образцов [14].

Атрибуты классов анализа определяются, исходя из знаний о предметной области и требований к системе. Связи между классами (ассоциации) определяются на основе анализа кооперативных диаграмм, затем анализируются и уточняются.

Целью объектно-ориентированного проектирования является адаптация предварительного системного проекта (набора классов «анализа»), составляющего стабильную основу архитектуры системы, к среде реализации с учетом всех нефункциональных требований.

Объектно-ориентированное проектирование включает два вида деятельности:

проектирование архитектуры системы;

проектирование элементов системы.

Проектирование архитектуры системы выполняется архитектором системы и включает в себя:

идентификацию архитектурных решений и механизмов, необходимых для проектирования системы;

анализ взаимодействий между классами анализа, выявление подсистем и интерфейсов;

формирование архитектурных уровней;

проектирование конфигурации системы.

Проектирование элементов системы включает в себя:

проектирование классов (детализация классов, уточнение операций и атрибутов, моделирование состояний, уточнение связей между классами);

проектирование баз данных (в зависимости от типа используемой для хранения данных СУБД - объектной или реляционной).

3 Структурное и объектно-ориентированное программирование в проектировании программного обеспечения распределенных информационных систем

3.1 Проектирование программного обеспечения распределенных информационных систем

Проектирование программного обеспечения информационных систем требует большой и трудоемкой работы. Современные крупные проекты информационных систем характеризуются, следующими особенностями:

сложность описания (достаточно большое количество функций, процессов, элементов данных и сложные взаимосвязи между ними), требующая тщательного моделирования и анализа данных и процессов;

наличие совокупности тесно взаимодействующих компонентов (подсистем), имеющих свои локальные задачи и цели функционирования (например, традиционных приложений, связанных с обработкой транзакций и решением регламентных задач, и приложений аналитической обработки (поддержки принятия решений), использующих нерегламентированные запросы к данным большого объема);

отсутствие прямых аналогов, ограничивающее возможность использования каких-либо типовых проектных решений и прикладных систем;

необходимость интеграции существующих и вновь разрабатываемых приложений;

функционирование в неоднородной среде на нескольких аппаратных платформах;

разобщенность и разнородность отдельных групп разработчиков по уровню квалификации и сложившимся традициям использования тех или иных инструментальных средств;

существенная временная протяженность проекта, обусловленная, с одной стороны, ограниченными возможностями коллектива разработчиков, и, с другой стороны, масштабами организации-заказчика и различной степенью готовности отдельных ее подразделений к внедрению ИС.

Для успешной реализации проекта объект проектирования должен быть прежде всего адекватно описан, должны быть построены полные и непротиворечивые функциональные и информационные модели информационной системы. Однако до недавнего времени проектирование информационных систем выполнялось в основном на интуитивном уровне с применением неформализованных методов, основанных на искусстве, практическом опыте, экспертных оценках и дорогостоящих экспериментальных проверках качества функционирования информационных систем. Кроме того, в процессе создания и функционирования информационных систем информационные потребности пользователей могут изменяться или уточняться, что еще более усложняет разработку и сопровождение таких систем.

Перечисленные факторы способствовали развитию исследований в области методологии программирования. Программирование обрело черты системного подхода с разработкой и внедрением языков высокого уровня, методов структурного и модульного программирования, языков проектирования и средств их поддержки, формальных и неформальных языков описаний системных требований и спецификаций и т.д.

Для целенаправленного выполнения проекта должен быть выполнен ряд работ, различных как по своему назначению, так и по квалификационным требованиям, предъявляемым к разработчикам. Иными словами, в ходе развития проекта командой разработчиков выполняются те или иные функции.

Функции, выполняемые разработчиками, — понятие неформализованное. В разных проектах оно может обретать свое содержание. Тем не менее типовые функции, которые предполагают практически все программные проекты, можно перечислить. Так, в любом программном проекте есть функции кодирования, т.е. записи программы на алгоритмическом языке по имеющимся спецификациям, анализа требований, т.е. выявления истинной потребности в создаваемой программе, тестирования и отладки. В рамках деятельности менеджера любого проекта необходимо организовать распределение функций проекта между исполнителями. Вполне естественно считать эти действия одной из функций менеджера. В результате ее выполнения члены команды, выполняющей проект, начинают играть соответствующие роли.

Состав и значимость ролей разработчиков и тех, кто с ними связан, различаются в зависимости от выполняемого проекта, от коллектива исполнителей, принятой технологии, от других факторов. Неоднозначность выбора ролей приводит к тому, что их список часто становится большим (иллюстрацией тому может служить первая редакция документации по Rational Unified Processing (RUP), в которой определено свыше 20 ролей; в последующих версиях документа это число сокращено до обозримого уровня [30]). Чрезмерное увеличение числа есть следствие отождествления роли и функции и, соответственно, игнорирования понятия родственности функций. В то же время, если ролей выбрано недостаточно, есть опасность, что не все нужные в проекте функции будут охвачены планированием и управлением.

Разработка современного программного комплекса представляет собой сложный и длительный процесс, который состоит из следующих этапов:

Предпроектные исследования, или анализ;

Проектирование и подготовка спецификаций;

Программирование, или кодирование;

Отладка;

Тестирование.

Во время предпроектного исследования собирается информация о предметной области, подбираются исходные данные и определяются функциональные требования к системе. Изучаются необходимые потребительские характеристики разрабатываемого программного обеспечения.

На этапе проектирования широко используется литература, стандарты и нормативные документы. Рассматриваются различные варианты реализации тех или иных функций и выбираются оптимальные. Здесь же разрабатывается архитектура, программный комплекс разбивается на подсистемы, определяются способы их взаимодействия. Результатом этого этапа является документация, с которой можно приступать к программированию, а именно: блок-схемы, SDL-диаграммы, функциональные схемы, описания алгоритмов.

На стадии программирования, или кодирования, происходит воплощение всех наработок, сделанных на предыдущем этапе в работающую систему. Все действия фиксируются как на уровне

комментариев в исходных кодах программ, так и в проектной документации. Необходимо организовать процесс таким образом, чтобы в любой момент времени был возможен откат на несколько шагов назад.

Отладка тесно связана с программированием, эти этапы разделяются весьма условно. Отладка может производиться либо непосредственно на оборудовании, для которого предназначено разрабатываемое программное обеспечение, либо на различных эмуляторах. После программирования и отладки должна быть получена законченная рабочая система.

Тестирование представляет собой проверку всех функциональных возможностей разработанного программного обеспечения на том оборудовании, для которого оно разрабатывалось и в условиях, максимально приближенных к реальным. Важным моментом, обеспечивающим объективность, является необходимость проведения тестирования людьми, не участвовавшими в программировании.

На каждом этапе применяются специализированные средства разработки: во время разработки алгоритмов и архитектуры в основном используются различные редакторы; при программировании применяются компиляторы и линковщики, при отладке – отладчики, при тестировании может использоваться специальное тестовое оборудование. В самом простом случае, без какой-либо автоматизации, все перечисленные средства являются автономными и никак не связанны между собой.

Кроме чисто технических задач процесс разработки программного обеспечения, включает также функции планирования и управления, а в условиях распределённого проектирования эти функции становятся особенно актуальными. В рамках планирования производится определение временных параметров проектирования, распределение подсистем между коллективами разработчиков. Кроме того, в течение всего процесса разработки необходимо отслеживать текущее состояние проекта для осуществления контроля и оперативного управления, а также для обеспечения синхронизации проектных процедур.

Исходя из вышесказанного, схема процесса создания программного комплекса будет выглядеть следующим образом (Приложение Г).

3.2 Структурный подход к проектированию информационных систем

Сущность структурного подхода к разработке информационных систем заключается в ее декомпозиции (разбиении) на автоматизируемые функции: система разбивается на функциональные подсистемы, которые в свою очередь делятся на подфункции, подразделяемые на задачи и так далее. Процесс разбиения продолжается вплоть до конкретных процедур. При этом автоматизируемая система сохраняет целостное представление, в котором все составляющие компоненты взаимосвязаны. При разработке системы «снизу-вверх» от отдельных задач ко всей системе целостность теряется, возникают проблемы при информационной стыковке отдельных компонентов.

Структурный подход к программированию представляет собой методологию создания программ. Его внедрение обеспечивает:

повышение производительности труда программистов при написании и контроле программ;

получение программ, которые более пригодны для сопровождения, так как состоят из отдельных модулей;

создание программ коллективом разработчиков;

окончание создания программ в заданный срок.

В структурированных программах обычно легко прослеживается основной алгоритм, они удобнее в отладке и менее чувствительны к ошибкам программирования. Эти свойства являются следствием важной особенности подпрограмм, каждая из которых представляет собой во многом самостоятельный фрагмент программы, связанный с основной программой лишь с помощью нескольких параметров. Такая самостоятельность подпрограмм позволяет локализовать в них все детали программной реализации того или иного алгоритмического действия, и поэтому изменение этих деталей, например в процессе отладки, обычно не приводит к изменениям основной программы.

Все наиболее распространенные методологии структурного подхода [9,11,12,13] базируются на ряде общих принципов [3]. В качестве двух базовых используются следующие принципы:

1. принцип «разделяй и властвуй» - принцип решения сложных проблем путем их разбиения на множество меньших независимых задач, легких для понимания и решения;

2. принцип иерархического упорядочивания - принцип организации составных частей проблемы в иерархические древовидные структуры с добавлением новых деталей на каждом уровне.

Выделение двух базовых принципов не означает, что остальные принципы являются второстепенными, поскольку игнорирование любого из них может привести к непредсказуемым последствиям (в том числе и к провалу всего проекта). Основными из них являются следующие принципы:

принцип абстрагирования - заключается в выделении существенных аспектов системы и отвлечения от несущественных аспектов;

принцип формализации - заключается в необходимости строгого методического подхода к решению проблемы;

принцип непротиворечивости - заключается в обоснованности и согласованности элементов;

принцип структурирования данных - заключается в том, что данные должны быть структурированы и иерархически организованы.

В структурном анализе используются в основном две группы средств, иллюстрирующих функции, выполняемые системой и отношения между данными. Каждой группе средств соответствуют определенные виды моделей (диаграмм), наиболее распространенными среди которых, являются следующие:

SADT (Structured Analysis and Design Technique) модели и соответствующие функциональные диаграммы;

DFD (Data Flow Diagrams) диаграммы потоков данных;

ERD (Entity-Relationship Diagrams) диаграммы «сущность-связь».

На стадии проектирования информационной системы модели расширяются, уточняются и дополняются диаграммами, отражающими структуру программного обеспечения: архитектуру программного обеспечения, структурные схемы программ и диаграммы экранных форм.

Перечисленные модели в совокупности дают полное описание информационных систем независимо от того, является ли она существующей или вновь разрабатываемой. Состав диаграмм в каждом конкретном случае зависит от необходимой полноты описания системы.

Структурное проектирование позволяет одновременно сосредотачиваться на меньшем количестве деталей.

Нисходящее проектирование хорошо работает , когда проблема имеет ясно выраженный иерархический характер.

3.3 Проектирование информационных систем на основе объектно-ориентированного подхода

Структурное проектирование работает хорошо, потому что оно позволяет нам одновременно сосредотачиваться на меньшем количестве деталей. Это логичная методика, которая поощряет организованную доводку системы и уменьшает уровень сложности (степени интеграции) на каждой из последующих стадий проекта. По очевидным причинам, нисходящее (структурное) проектирование подходит лучше всего тогда, когда применяется к проблемам, которые имеют ясно выраженный иерархический характер. К сожалению, многие из реальных проблем не иерархические. Проект, основанный на построении сверху вниз, имеет также и следующие ограничения, которые станут очевидными при разработке и сопровождении больших программных систем:

функциональную точку зрения трудно развивать;

реальные системы трудно охарактеризовать функционально;

фокусирование на функциональности теряет из виду данные;

функциональная ориентация производит код, менее пригодный для многократного использования.

Методы структурного проектирования помогают упростить процесс разработки сложных систем за счет использования алгоритмов как готовых строительных блоков. Аналогично, методы объектно-ориентированного проектирования созданы, чтобы помочь разработчикам применять мощные выразительные средства объектного и объектно-ориентированного программирования, использующего в качестве блоков классы и объекты.

Структурное проектирование характеризуется перемещением от общей формулировки того, что программа делает к все более детализированным формулировкам этого относительно каждой специфической задачи.

Структурное проектирование не подходит для разработки больших программных систем, потому что оно выторговывает краткосрочное удобство в обмен на отсутствие гибкости при длительном сопровождении. Существует незаконная привилегия одной функции над другими, теряются из виду

данные, оставаясь на заднем плане задачи. Затрудняется возможность многократного использования.

Первое отличие структурного подхода от объектно-ориентированного подхода заключается в принципах декомпозиции и структурной организации элементов (компонентов, модулей) системы. Согласно этим принципам система представляет собой структуру, состоящую из четко выраженных модулей, связанных между собой определенными отношениями.

При использовании структурного подхода (первый вид декомпозиции) выполняется функциональная (процедурная, алгоритмическая) декомпозиция системы, т. е. она представляется в виде иерархии (дерева) взаимосвязанных функций. На высшем уровне система представляется единым целым с наивысшей степенью абстракции и по мере детализации (добавления уровней) разбивается на функциональные компоненты с более конкретным содержанием.

Второй вид декомпозиции – объектно-ориентированный. В рамках этого подхода система разбивается на набор объектов, соответствующих объектам реального мира, взаимодействующих между собой путем посылки сообщений.

Вторым отличием является объединение в объекте как атрибутивных данных (характеристики, свойства), так и поведения (функции, методы). В функционально-ориентированных системах функции и данные хранятся (существуют) отдельно.

Третье отличие двух подходов заключается в структурной организации внутри модулей системы. В структурном подходе модуль состоит из функций, иерархически связанных между собой отношением композиции, т. е. функция состоит из подфункций, подфункция из подподфункций и т.д. В объектно-ориентированном подходе иерархия выстраивается с использованием двух отношений: композиции и наследования. При этом в объектно-ориентированном подходе «объект-часть» может включаться сразу в несколько «объектов-целое». Таким образом, модуль в структурном подходе представляется в виде дерева, а в объектно-ориентированном подходе – в виде ориентированного графа, т. е. с помощью более общей структуры.

Объектно-ориентированное проектирование – это конструирование программных систем как структурных коллекций, реализующих абстрактные типы данных.

Объектно-ориентированное проектирование и объектно-ориентированное программирование улучшают возможности нисходящего проектирования, концентрируя больше внимание на данных системы, а не на том, что система делает. Это подход позволяет создавать системы, которые легче сопровождать, они более гибкие, более устойчивые и более приспособлены к многократному использованию, чем создаваемые при нисходящем структурном подходе.

Преимущества объектно-ориентированного метода:

- работают на более высоком уровне абстракции;

- нет «прыжков» между фазами;

- поддерживают данные, которые имеют тенденцию, к большей стабильности, чем функции;

- поощряют и поддерживают классические достоинства хорошего программирования и проектирования;

- сопровождаются инструментами, обеспечивающими поддержку повторного использование кода.

Объектно-ориентированный подход имеет два аспекта:

объектно-ориентированная разработка программного обеспечения;

объектно-ориентированная реализация программного обеспечения.

Объектно-ориентированная разработка программного обеспечения связана с применением объектно-ориентированных моделей при разработке программных систем и их компонентов. К объектно-ориентированной разработке относятся:

- объектно-ориентированные технологии разработки программных систем;

- инструментальные средства, поддерживающие эти технологии.

Объектно-ориентированная разработка может начаться на самом первом этапе жизненного цикла; она не связана с языком программирования, на котором предполагается реализовать разрабатываемую программную систему: этот язык может и не быть объектно-ориентированным.

Объектно-ориентированная разработка программного обеспечения связана с применением объектно-ориентированных технологий. Обычно эти объектно-ориентированные методологии поддерживаются инструментальными программными средствами, но и без таких средств они полезны, так как позволяют хорошо понять различные аспекты и свойства разрабатываемой программной системы, что в последующем существенно облегчает ее реализацию, тестирование, сопровождение, разработку новых версий и более существенную модификацию.

Можно выделить следующие объектно-ориентированные методологии разработки программного обеспечения:

- RUP (Rational Unified Process);

- OMT (Object Modeling Technique);

- SA/SD (Structured Analysis/Structured Design);

- JSD (Jackson Structured Development);

- OSA (Object-Oriented System Analysis)

Объектно-ориентированный подход в проектировании, как и функционально-ориентированный, предполагает декомпозицию информационных систем. Если в функционально-ориентированном подходе декомпозиции подлежат процессы обработки, то в объектно-ориентированном подходе декомпозиции подлежат объекты, которые характеризуются определенной структурой данных. Здесь декомпозиция идет от данных. В объектно-ориентированном подходе выделяют классы объектов. Каждый класс содержит однородные объекты. Объектам одного класса присуще одинаковое множество методов реагирования на внешние сообщения. Иерархическая декомпозиция системы представляется в виде иерархии классов объектов, а функционирование системы – в виде взаимодействия объектов, обменивающихся сообщениями.

Среди свойств объектов в объектно-ориентированном подходе можно отметить:

- инкапсуляция, что означает скрытие информации. Смысл этого свойства в том, что состав и структура атрибутов объекта не зависит от сообщений, поступающих извне;

- наследование – это свойство, связанное с выделением иерархических классов объектов, то есть существуют родительские и дочерние классы. Оно проявляется в том, что, методы реагирования объекта, предусмотренные родительским классом, автоматически присваивают объектам дочерних классов, то есть родительские классы имеют общие методы, а дочерние – как общие, так и частные;

- полиморфизм – возможность выбора объектом в ответ на получаемые им сообщения какого-либо метода из множества методов в зависимости от того, какое пришло сообщение.

Наличие этих свойств у объекта позволяет в объектно-ориентированном подходе добиться параллельности и автономности разработки отдельных компонент системы, т.е. возможно создание прототипов с дальнейшей интеграцией отдельных прототипов в единую систему и использование итерационного подхода к разработке информационных систем.

Другое достоинство объектно-ориентированного подхода состоит в упрощении накопления типовых проектных решений с тем, чтобы в дальнейших разработках новых информационных систем осуществить сбор новой системы из готовых компонент. Эта особенность связана с тем, что классы объектов повторяются в определенной мере при переходе от одной информационной системы к другой, а для повторяющихся классов уже запрограммированы методы, разработаны и описаны структуры объектов данных.

Модель проектирования информационных систем на основе объектно-ориентированного подхода представлена в приложении (Приложение Д)

Отличительной чертой модели объектно-ориентированного проектирования является отсутствие строгой последовательности в выполнении стадий как в прямом, так и в обратном направлениях процесса проектирования по отдельным компонентам.

Основное преимущество объектно-ориентированного подхода состоит в упрощении проектирования информационных систем, при наличии типовых проектных решений по отдельным компонентам, а также легкости модификации, поскольку модификация касается лишь отдельных компонент.

3.4. Сопоставление и взаимосвязь структурного и объектно-ориентированного подходов

Гради Буч сформулировал главное достоинство объектно-ориентированного подхода следующим образом: объектно-ориентированные системы более открыты и легче поддаются внесению изменений, поскольку их конструкция базируется на устойчивых формах. Это дает возможность системе развиваться постепенно и не приводит к полной ее переработке даже в случае существенных изменений исходных требований.

Буч отметил ряд следующих преимуществ объектно-ориентированного подхода:

объектная декомпозиция дает возможность создавать программные системы меньшего размера путем использования общих механизмов, обеспечивающих необходимую экономию выразительных средств. Использование объектно-ориентированного программирования существенно повышает уровень унификации разработки и пригодность для повторного использования не только программного обеспечения, но и проектов, что в конце концов ведет к сборочному созданию программного обеспечения. Системы зачастую получаются более компактными, чем их не

объектно-ориентированные эквиваленты, что означает не только уменьшение объема программного кода, но и удешевление проекта за счет использования предыдущих разработок;

объектная декомпозиция уменьшает риск создания сложных систем программного обеспечения, так как она предполагает эволюционный путь развития системы на базе относительно небольших подсистем. Процесс интеграции системы растягивается на все время разработки, а не превращается в единовременное событие;

объектная модель вполне естественна, поскольку в первую очередь ориентирована на человеческое восприятие мира, а не на компьютерную реализацию;

объектная модель позволяет в полной мере использовать выразительные возможности объектных и объектно-ориентированных языков программирования.

К недостаткам объектно-ориентированного программирования относятся некоторое снижение производительности функционирования программного обеспечения (которое, однако, по мере роста производительности компьютеров становится все менее заметным) и высокие начальные затраты. Объектная декомпозиция существенно отличается от функциональной, поэтому, переход на новую технологию связан как с преодолением психологических трудностей, так и дополнительными финансовыми затратами.

Объектно-ориентированный подход не дает немедленной отдачи. Эффект от его применения начинает сказываться после разработки двух-трех проектов и накопления повторно используемых компонентов, отражающих типовые проектные решения в данной области. Переход организации на объектно-ориентированную технологию - это смена мировоззрения, а не просто изучение новых CASE-средств и языков программирования.

Таким образом, структурный подход по-прежнему сохраняет свою значимость и достаточно широко используется на практике. На примере языка UML хорошо видно, что его авторы заимствовали то рациональное, что можно было взять из структурного подхода: элементы функциональной декомпозиции в диаграммах вариантов использования, диаграммы состояний, диаграммы деятельности и др. Очевидно, что в конкретном проекте сложной системы невозможно обойтись только одним способом декомпозиции. Можно начать декомпозицию каким-либо одним способом, а затем, используя полученные результаты, попытаться рассмотреть систему с другой точки зрения.

Основой взаимосвязи между структурным и объектно-ориентированным подходами является общность ряда категорий и понятий обоих подходов (процесс и вариант использования, сущность и класс и др.). Эта взаимосвязь может проявляться в различных формах. Так, одним из возможных вариантов является использование структурного анализа как основы для объектно-ориентированного проектирования. При этом структурный анализ следует прекращать, как только структурные модели начнут отражать не только деятельность организации (бизнес-процессы), а и систему программного обеспечения. После выполнения структурного анализа можно различными способами приступить к определению классов и объектов. Так, если взять какую-либо отдельную диаграмму потоков данных, то кандидатами в классы могут быть элементы структур данных.

Другой формой проявления взаимосвязи можно считать интеграцию объектной и реляционной технологий. Реляционные СУБД являются на сегодняшний день основным средством реализации крупномасштабных баз данных и хранилищ данных. Причины этого достаточно очевидны: реляционная технология используется достаточно долго, освоена огромным количеством пользователей и разработчиков, стала промышленным стандартом, в нее вложены значительные средства и создано множество корпоративных базах данных в самых различных отраслях,

реляционная модель проста и имеет строгое математическое основание; существует большое разнообразие промышленных средств проектирования, реализации и эксплуатации реляционных баз данных. Вследствие этого реляционные базы данных в основном используются для хранения и поиска объектов в так называемых объектно-реляционных системах.

Взаимосвязь между структурным и объектно-ориентированным подходами достаточно четко просматривается в различных технологиях создания программного обеспечения.

Главный недостаток структурного подхода заключается в следующем: процессы и данные существуют отдельно друг от друга (как в модели деятельности организации, так и в модели программной системы), причем проектирование ведется от процессов к данным. Таким образом, помимо функциональной декомпозиции, существует также структура данных, находящаяся на втором плане.

В объектно-ориентированном подходе основная категория объектной модели - класс - объединяет в себе на элементарном уровне как данные, так и операции, которые над ними выполняются (методы). Именно с этой точки зрения изменения, связанные с переходом от структурного к объектно-ориентированному подходу, являются наиболее заметными. Разделение процессов и данных преодолено, однако остается проблема преодоления сложности системы, которая решается путем использования механизма компонентов.

Данные по сравнению с процессами являются более стабильной и относительно редко изменяющейся частью системы. Отсюда следует главное достоинство объектно-ориентированного подхода: объектно-ориентированные системы более открыты и легче поддаются внесению изменений, поскольку их конструкция базируется на устойчивых формах.

Безусловно, объектно-ориентированная модель наиболее адекватно отражает реальный мир, представляющий собой совокупность взаимодействующих (посредством обмена сообщениями) объектов. Но на практике в настоящий момент продолжается формирование стандарта языка объектно-ориентированного моделирования UML, и количество CASE-средств, поддерживающих объектно-ориентированный подход, невелико по сравнению с поддерживающими структурный подход. Кроме того, диаграммы, отражающие специфику объектного подхода (диаграммы классов и т.п.), гораздо менее наглядны и плохо понимаемы непрофессионалами. Поэтому одна из главных целей внедрения CASE-технологии, а именно снабжение всех участников проекта (в том числе и заказчика) общим языком «для передачи понимания», обеспечивается на сегодняшний день только структурными методами.

При переходе от структурного подхода к объектному, как при всякой смене технологии, необходимо вкладывать деньги в приобретение новых инструментальных средств. Здесь следует учесть и расходы на обучение (овладение методом, инструментальными средствами и языком программирования). Для некоторых организаций эти обстоятельства могут стать серьезными препятствиями.

Несмотря на отдельные критические замечания в адрес ООП, в настоящее время именно эта парадигма используется в подавляющем большинстве промышленных проектов. Однако, нельзя считать, что ООП является наилучшей из методик программирования во всех случаях.

Процедурное программирование лучше подходит для случаев, когда важны быстродействие и используемые программой ресурсы, но требует большего времени для разработки.

Объектное программирование подходит когда важна управляемость проекта и его модифицируемость, а также скорость разработки.

Исследование Thomas E. Potok, Mladen Vouk и Andy Rindos показало отсутствие значимой разницы в продуктивности разработки программного обеспечения между ООП и процедурным подходом.

Кристофер Дэйт указывает на невозможность сравнения ООП и других технологий во многом из-за отсутствия строгого и общепризнанного определения ООП.

Александр Степанов, в одном из своих интервью, указывал на то, что ООП «методологически неправильно» и что «… ООП практически такая же мистификация как и искусственный интеллект…».

Фредерик Брукс (Frederick P. Brooks, Jr.) в своей статье «No Silver Bullet. Essence and Accidents of Software Engineering» (Computer Magazine; April 1987) указывает на то, что наиболее сложной частью создания программного обеспечения является « … спецификация, дизайн и тестирование концептуальных конструкций, а отнюдь не работа по выражению этих концептуальных конструкций…». ООП (наряду с такими технологиями как искусственный интеллект, верификация программ, автоматическое программирование, графическое программирование, экспертные системы и др.), по его мнению, не является «серебряной пулей», которая могла бы на порядок величины (то есть примерно в 10 раз, как говорится в статье) снизить сложность разработки программных систем. Согласно Бруксу, «…ООП позволяет сократить только привнесённую сложность в выражение дизайна. Дизайн остаётся сложным по своей природе…».

Эдсгер Дейкстра указывал: «… то о чём общество в большинстве случаев просит - это змеиное масло. Естественно, «змеиное масло» имеет очень впечатляющие имена, иначе будет очень трудно что-то продать: «Структурный анализ и Дизайн», «Программная инженерия», «Модели зрелости», «Управляющие информационные системы» (Management Information Systems), «Интегрированные среды поддержки проектов», «Объектная ориентированность», «Реинжиниринг бизнес-процессов»…».

Никлаус Вирт считает, что ООП - не более чем тривиальная надстройка над структурным программированием, и преувеличение её значимости, выражающееся, в том числе, во включении в языки программирования всё новых модных «объектно-ориентированных» средств, вредит качеству разрабатываемого программного обеспечения.

Патрик Киллелиа в своей книге «Тюнинг веб-сервера» писал: «… объектно-ориентированное программирование предоставляет вам множество способов замедлить работу ваших программ …»

Известная обзорная статья проблем современного объектно-ориентированного программирования перечисляет некоторые типичные проблемы объектно-ориентированного программирования.

При классификации критических высказываний в адрес объектно-ориентированного программирования, можно выделить несколько аспектов критики данного подхода к программированию:

1. Критика рекламы объектно-ориентированного программирования.

Критикуется явно высказываемое или подразумеваемое в работах некоторых пропагандистов объектно-ориентированного программирования, а также в рекламных материалах «объектно-ориентированных» средств разработки представление об объектном программировании как о некоем всемогущем подходе, который магическим образом устраняет сложность программирования. Как замечали многие, в том числе Брукс и Дейкстра, «серебряной пули не существует» - независимо от того, какой парадигмы программирования придерживается разработчик, создание нетривиальной сложной программной системы всегда сопряжено со

значительными затратами интеллектуальных ресурсов и времени. Из наиболее квалифицированных специалистов в области объектно-ориентированного программирования никто, как правило, не отрицает справедливость критики этого типа.

2. Оспаривание эффективности разработки методами объектно-ориентированного программирования.

Критики оспаривают тезис о том, что разработка объектно-ориентированных программ требует меньше ресурсов или приводит к созданию более качественного программного обеспечения. Проводится сравнение затрат на разработку разными методами, на основании которого делается вывод об отсутствии у объектно-ориентированного программирования преимуществ в данном направлении. Учитывая крайнюю сложность объективного сравнения различных разработок, подобные сопоставления, как минимум, спорны.

3. Производительность объектно-ориентированных программ.

Указывается на то, что целый ряд «врождённых особенностей» объектно-ориентированные технологии делают построенные на её основе программы технически менее эффективными, по сравнению с аналогичными необъектными программами. Не отрицая действительно имеющихся дополнительных накладных расходов на организацию работы объектно-ориентированных программ, нужно, однако, отметить, что значение снижения производительности часто преувеличивается критиками. В современных условиях, когда технические возможности компьютеров чрезвычайно велики и постоянно растут, для большинства прикладных программ техническая эффективность оказывается, менее существенна, чем функциональность, скорость разработки и сопровождаемость. Лишь для некоторого, очень ограниченного класса программ (программное обеспечение встроенных систем, драйверы устройств, низкоуровневая часть системного программного обеспечения, научное программное обеспечение) производительность остаётся критическим фактором.

4. Критика отдельных технологических решений в объектно-ориентированных-языках и библиотеках.

Эта критика многочисленна, но затрагивает она не объектно-ориентированное программирование как таковое, а приемлемость и применимость в конкретных случаях тех или иных реализаций её механизмов. Одним из излюбленных объектов критики является язык C++, входящий в число наиболее распространённых промышленных объектно-ориентированных-языков.

Гради Буч указывает на следующие причины, приводящие к снижению производительности программ из-за использования объектно-ориентированных средств:

1.Динамическое связывание методов.

Обеспечение полиморфного поведения объектов приводит к необходимости связывать методы, вызываемые программой (то есть определять, какой конкретно метод будет вызываться) не на этапе компиляции, а в процессе исполнения программы, на что тратится дополнительное время. При этом реально динамическое связывание требуется не более чем для 20 % вызовов, но некоторые объектно-ориентированные-языки используют его постоянно.

2.Значительная глубина абстракции.

Объектно-ориентированная-разработка часто приводит к созданию «многослойных» приложений, где выполнение объектом требуемого действия сводится к множеству обращений к объектам более низкого уровня. В таком приложении происходит очень много вызовов методов и возвратов из методов, что, естественно, сказывается на производительности.

3.Наследование «размывает» код.

Код, относящийся к «оконечным» классам иерархии наследования (которые обычно и используются программой непосредственно) - находится не только в самих этих классах, но и в их классах-предках. Относящиеся к одному классу методы фактически описываются в разных классах. Это приводит к двум неприятным моментам:

- снижается скорость трансляции, так как компоновщику приходится подгружать описания всех классов иерархии;

- снижается производительность программы в системе со страничной памятью поскольку методы одного класса физически находятся в разных местах кода, далеко друг от друга, при работе фрагментов программы, активно обращающихся к унаследованным методам, система вынуждена производить частые переключения страниц.

4. Инкапсуляция снижает скорость доступа к данным.

Запрет на прямой доступ к полям класса извне приводит к необходимости создания и использования методов доступа. И написание, и компиляция, и исполнение методов доступа сопряжено с дополнительными расходами.

5.Динамическое создание и уничтожение объектов.

Динамически создаваемые объекты, как правило, размещаются в куче, что менее эффективно, чем размещение их на стеке и, тем более, статическое выделение памяти под них на этапе компиляции.

Несмотря на отмеченные недостатки, выгоды от использования объектно-ориентированного программирования более весомы. Кроме того, повышение производительности за счёт лучшей организации объектно-ориентированного-кода, в некоторых случаях компенсирует дополнительные накладные расходы на организацию функционирования программы. Можно также заметить, что многие эффекты снижения производительности могут сглаживаться или даже полностью устраняться за счёт качественной оптимизации кода компилятором. Например, упомянутое выше снижение скорости доступа к полям класса из-за использования методов доступа устраняется, если компилятор вместо вызова метода доступа использует онлайн-подстановку (современные компиляторы делают это вполне уверенно).

Контрольные вопросы:

1. Какие две методики проектирования вы знаете?
2. Что такое сущность ?
3. Что такое атрибут?
4. Какие сущности виды вы знаете?
5. Какие модели проектирования в структурном подходе вы знаете? Опишите их
6. Что представляет собой ER-диаграмма?
7. Какие типы связей вы знаете?
8. Какие графические примитивы вы знаете ?

Список использованных источников:

1. Технологии разработки программного обеспечения С.А. Орлов
2. Технологии разработки программного обеспечения В.В. Бахтизин, Л.А. Глухова
3. Project Management For Dummies / Управление проектами для "чайников"
4. Л. Н. Боронина З. В. Сенук основы управления проектами
5. https://studopedia.su/3_11546_metodi-strukturnogo-proektirovaniya.html
6. https://studfiles.net/preview/3828360/page:7/
7. https://studfiles.net/preview/3828360/page:6/