МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технологический университет «СТАНКИН»
(ФГБОУ ВО МГТУ «СТАНКИН»)

Кафедра "Информационные технологии и вычислительные системы"

Новоселова Ольга Вячеславовна

Современные технологии и средства разработки программного обеспечения

Курс лекций
по дисциплине «Современные технологии и средства разработки программного обеспечения»

для студентов МГТУ «СТАНКИН», обучающихся

по направлению: 09.03.01 "Информатика и вычислительная техника" ,

по профилю: "Программное обеспечение средств вычислительной техники и автоматизированных систем"

Москва 2020г.

Оглавление.
7 семестр

Раздел 2. Современные подходы и технологии анализа и моделирования бизнес-процессов

7 семестр.

Раздел 2. Современные подходы и технологии анализа и моделирования бизнес-процессов.

Лекции 5-6.

Тема 5. Программный продукт как средство автоматизации выполнения бизнес-процесса. Подходы, применяемые при разработке программных продуктов. Обзор современных технологий анализа и моделирования бизнес-процессов.
Тема 6. Основы структурного подхода к анализу и моделированию бизнес-процессов.
Тема 7. Основы объектно-ориентированного подхода к анализу и моделированию бизнес-процессов.

План лекции.

1. Основные понятия и определения.
2. Подходы, используемые при разработке программного обеспечения или автоматизированной системы.
3. Этапы эволюции автоматизации интеллектуального труда.
4. Основы структурного подхода к анализу и моделированию бизнес-процессов.
5. Основы объектно-ориентированного подхода к анализу и моделированию бизнес-процессов.

Тема 5. Программный продукт как средство автоматизации выполнения бизнес-процесса. Подходы, применяемые при разработке программных продуктов. Обзор современных технологий анализа и моделирования бизнес-процессов.
Основные понятия и определения.

Программные продукты предназначены повышать эффективность выполнения предметных задач, заключающихся в исполнении основных или вспомогательных процессов деятельности человека. Это позволяет уменьшать трудоемкость выполнения различных операций и действий, снижать временные издержки при выполнении задач, улучшать процесс принятия решений, автоматизировать и оптимизировать процессы производства, управления и прочее.

Процессы на предприятии (или в организации) можно определить следующим образом:

Бизнес-процесс — это совокупность взаимосвязанных мероприятий или задач, направленных на создание определенного продукта или услуги для потребителей. В качестве графического описания деятельности применяются блок-схемы бизнес-процессов.

Существуют три вида бизнес-процессов:

1. **Операционные** — бизнес-процессы, которые составляют основной бизнес компании и создают основной поток доходов. Примерами операционных бизнес-процессов являются Снабжение, Производство, Маркетинг и Продажи.
2. **Поддерживающие** — бизнес-процессы, которые обслуживают основной бизнес. Например, Бухгалтерский учет, Подбор персонала, Техническая поддержка, Административно-хозяйственный отдел (АХО).
3. **Управляющие** — бизнес-процессы, которые управляют функционированием системы. Примером управляющего процесса может служить Корпоративное управление и Стратегический менеджмент.

Бизнес-процесс начинается со спроса потребителя и заканчивается его удовлетворением. Процессно-ориентированные организации стараются устранять барьеры и задержки, возникающие на стыке двух различных подразделений организации при выполнении одного бизнес-процесса.

Производственный процесс — представляет собой совокупность целенаправленных действий персонала предприятия по переработке сырья и материала в готовую продукцию.

Производственный процесс — это целенаправленное, постадийное превращение исходного сырья и материалов в готовый продукт заданного свойства и пригодный к потреблению или к дальнейшей обработке. Производственный процесс начинается с его проекта и заканчивается на стыке производства и потребления, после чего происходит расходование произведенной продукции.

Техническая и организационно-экономическая характеристика производственного процесса на предприятии определяется видом продукции, объемом производства, типом и видом применяемой техники и технологии, уровнем специализации.

Производственный процесс на предприятиях подразделяется на два вида: **основной и вспомогательный.**

К основному относятся процессы, связанные непосредственно с превращением предметов труда в готовую продукцию. Например, переплавка руды в доменной печи и превращение ее в металл или превращение муки в тесто, а затем в готовый испеченный хлеб, производство деталей из заготовок и сборка турбин или станков.

Вспомогательные процессы: перемещение предметов труда, ремонт оборудования, уборка помещений и т. д. Эти виды работ лишь способствуют течению основных процессов, но сами непосредственно в них не участвуют.

Основное отличие вспомогательных процессов от основных состоит в различии места реализации и потребления. Продукция основного производства, где совершаются основные производственные процессы, реализуется потребителям на сторону, согласно заключенным договорам на поставку. Эта продукция имеет свое фирменное наименование, маркировку, на нее устанавливается рыночная цена.

Продукция вспомогательного производства, где осуществляются вспомогательные процессы и обслуживание, потребляется внутри предприятия. Затраты на выполнение обслуживания и вспомогательных работ целиком относятся на себестоимость основной продукции, которая реализуется потребителям на сторону.

Различают производственный процесс и технологический процесс. Производственный процесс включает в себя все без исключения работы, связанные с изготовлением изделий на предприятии. В производственный процесс входят обработка материала (сырья) с целью превращения его в изделия (продукцию), выпускаемые заводом; работы по доставке, хранению и распределению сырья; изготовление и ремонт инструментов: ремонт оборудования; снабжение электроэнергией, светом, теплом, паром и т. д.

Технологический процесс охватывает работы, непосредственно связанные с превращением сырья в готовую продукцию. Технологический процесс — основная часть производства (производственного процесса).

Технологический процесс состоит из целого ряда производственных операций, которые выполняются в строго определенной последовательности. Производственной операцией называется часть технологического процесса, выполняемая на определенном рабочем месте определенным инструментом или на определенном оборудовании.

"Технологический процесс" — это часть производственного процесса, содержащая целенаправленные действия по изменению и (или) определению состояния предмета труда. К предметам труда относят заготовки и изделия.

— ГОСТ 3.1109-82

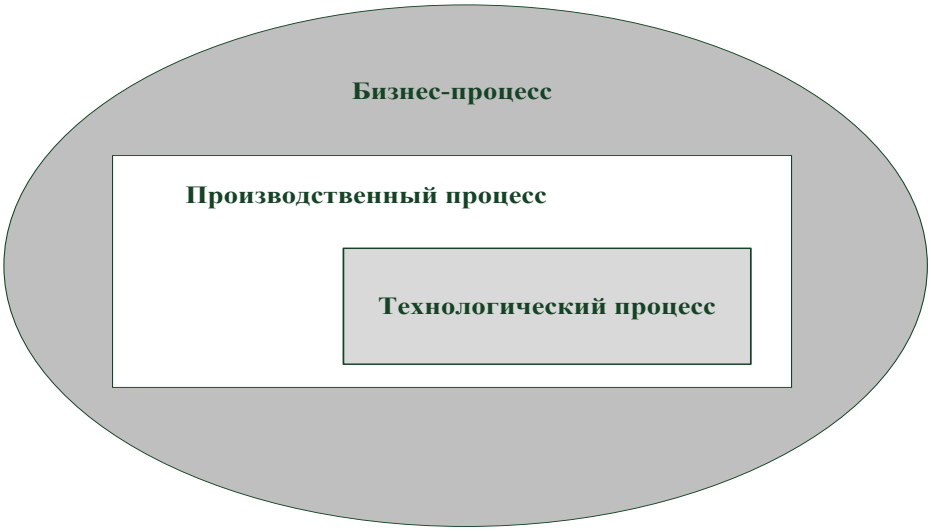

Рисунок 2.1. Схема процессов

Для различных классов предметных задач и процессов используют различные программные продукты при автоматизации. Программный продукт предназначен для автоматизированного или автоматического выполнения процесса решения задачи и является средством, заменяющим частично или полностью человека при выполнении определенных функций.

Исторически сложилось несколько подходов при автоматизации задач, наиболее применяемыми из которых являются структурный и объектно-ориентированный. В рамках каждого подхода были разработаны методы, технологии и методологии для анализа и моделирования процессов и информационной структуры предметных задач.

Методологии структурного подхода и методы получили довольно широкое применение в крупных отраслях промышленности, особенно в государственных, например, в военно-промышленном комплексе и аэрокосмических программах, так как в этих областях традиционно требуется жесткое планирование процесса разработки и подробное документирование процесса проектирования и реализации системы.

Подходы, используемые при разработке программного обеспечения или автоматизированной системы.

1.структурный подход (СП) - традиционный

- *нисходящее структурное проектирование (НСП)*
- *проектирование, структурированное по данным (ПСД)*

2.объектно-ориентированный подход (ООП)

3. когнитивный подход (КП)

1. Классический подход к разработке сложных систем базируется на методологии структурного проектирования, в основе которой лежит алгоритмическая декомпозиция системы по методу «сверху вниз».

1. 1. Этот подход заключается в декомпозиции задачи на функции или процессы, приводящей к созданию иерархии процессов и подпроцессов. Это алгоритмическая декомпозиция.

1.2. является альтернативой нисходящей методологии. При этом сначала определяются структуры данных, а затем на их основе определяется структура процессов. Таким образом, делается попытка сначала четко определить объекты, а затем сделать их структуру видимой в процессах, обеспечивающих операции над объектами. Структура системы строится как организация преобразований входных потоков в выходные.

2. Стремление избавиться от недостатков структурного подхода привело к развитию новых идей, основанных на объектной декомпозиции. Такой подход к разработке программных систем получил название объектно-ориентированного**.** В его основе лежат понятия «объект» и «класс». В объектно-ориентированном подходе учитывается важность трактовки объектов ПО как активных элементов, причем каждый объект наделен своим собственным набором допустимых операций.

В реальном мире, а точнее, в интересующей разработчика предметной области, в качестве объектов могут рассматриваться конкретные предметы, а также абстрактные или реальные сущности. Объект обладает индивидуальностью и поведением, имеет атрибуты, значения которых определяют его состояние.

Каждый объект является представителем некоторого класса однотипных объектов. Класс определяет общие свойства для всех его объектов.

Объектно-ориентированная декомпозиция заключается в представлении прикладной системы в виде совокупности классов и объектов предметной области. При этом иерархический характер сложной системы отражается в виде иерархии классов, а ее функционирование рассматривается как взаимодействие объектов. Такой подход позволяет описать системы наиболее естественным образом. В особенности он удобен для систем, интегрированных с базами данных, в частности с клиент-серверной архитектурой.

3. Когнитивный подход, основанный на знаниях (от латинского «знания») характеризуется как находящийся в стадии становления. Сущностью данного подхода является многоуровневое представление событий и явлений реального мира (каждый уровень образуется триадой функциональных центров – контекстуального, структурного и монадического – и связью уровней на основе закона цикличности).

Суть подхода заключается в таком информационном моделировании реального мира, которое позволит формировать множество взаимосвязанных моделей, при этом данное множество моделей должно описываться не как одна целостная система, а как свободно организованная сеть с открытой структурой.

Ключевой стадией в жизненном цикле процесса разработки ПО является проектирование. Ошибки и просчеты, допущенные здесь – самые дорогие. Поэтому основные усилия в области технологии программирования, технологии разработки ПО направлены на автоматизацию проектирования программных комплексов. При этом базисными понятиями являются – модель предметной области, модель системы, модель программы.

Каждый подход включает набор *методов, либо базируется на методологиях, а также поддерживается средствами*, обеспечивающими последовательность переходов от традиционного процесса решения предметной задачи к ее автоматизированному решению в вычислительной среде.

Обзор современных технологий анализа и моделирования бизнес-процессов

(таблица 2.1)

IDEF (**I**CAM **Def**inition language**, I**ntegrated **C**omputer-**A**ided **M**anufacturing) – интеграция компьютерных и промышленных технологий.

SADT (Structured Analysis and Design Technique) – технология (методология) структурного анализа и проектирования (графический язык описания функциональных систем)

CASE (Computer Aided System (Software) Engineering) – автоматизированная разработка программного обеспечения

UML (Unified Modeling Language) – унифицированный язык моделирования

Дракон- Дружелюбный Русский Алгоритмический язык, Который Обеспечивает Наглядность; дружелюбные алгоритмические конструкции общего назначения

МАИТ- методология автоматизации интеллектуального труда

ERD – entity relationship diagram – «сущность-связь» диаграмма

DFD - data flow diagram – диаграмма потоков данных

STD – state transition diagram – диаграмма переходов состояний

Таблица 2.1.

Название методологии	Подход	Компоненты предм. задачи	Методы решения предм. задачи	Место в ЖЦ	Область эффективного применения
IDEF (SADT)	структурный	Информац. Функцион. *Состояние*	IDEF 1X, IDEF1 IDEF 0, IDEF 3 *IDEF 2*	Анализ, Проектирование структур данных и обработки (подготовка реализации)	Документирование, моделирование и анализ процессов функционирования сложных систем. Для описания процессов производства и использования ресурсов (верхние уровни) на этапах проектирования систем (IDEF0), для сбора информации и моделирования бизнес-процессов (IDEF3) и т.д.
CASE-методология	структурный	Информац. Функцион. Состояние	ERD (метод Чена) DFD STD	Анализ, Проектирование структур данных и обработки (подготовка реализации)	Анализ, моделирование и проектирование задач для разработки системного и управляющего ПО, для стратегического планирования, управления финансами и др.
UML	ОО	Информац. Функцион.	Диаграмма классов Диаграмма прецедентов (вариантов использования), Д. Последовательности действий	Анализ, Проектирование структур объектов, проектирование обработки (подготовка реализации)	Анализ, моделирование и проектирование сложного информационно объемного программного обеспечения (АС) ОО направленности, *описание других задач*
Дракон	когнитивный	Функцион.	Дракон-схема	Анализ, Проектирование обработки (подготовка реализации)	Анализ, моделирование и проектирование АС для задач космической отрасли (создание космического корабля), *описание научно-технических, медицинских, биологических и др. задач*
МАИТ	когнитивный	Информац. Функцион.	Концепт. структура, Инфологич.структура Система предм. действий, Система предм. зависимостей, Система предм.доступов и предм.манипуляций (обработки)	Анализ, Концептуальное моделирование, Проектирование структур данных и обработки	Анализ, моделирование и проектирование АС для информационно сложноструктурированных предметных областей (производство, станкостроение, машиностроение), *описание других задач*

Этапы эволюции автоматизации интеллектуального труда

1. Электронная обработка данных
2. Информационные технологии
3. Новые информационные технологии
4. Когнитивный подход

На этапе электронной обработки данных основополагающим методом при разработке программ был так называемый метод HIPO – метод, который упорядочивал процедуру иерархической декомпозиции сложных процедур обработки данных и тем самым облегчал процесс программирования.

Переход к этапу информационных технологий характеризовался качественным скачком при разрешении количественных проблем, а именно, переход от множества переменных-данных к структурам на них. Реальный мир стал отражаться через объекты и их связи. Моделирование данных в виде структур данных получило формальные воплощения в вычислительной среде - появились модели представления данных, ориентированные на структурные аспекты представления данных предметной области или “синтаксические модели”. К наиболее распространенным можно отнести сетевую, иерархическую и реляционную, наиболее полной из которых явилась реляционная модель Кодда, разработанная в начале 60-х годов. Одним из отмечаемых недостатков подобных моделей являлась невозможность адекватного отражения семантики предметной области. Новый подход к моделированию данных был предложен Ченом в начале 70-х годов и получил название “сущность-связь-атрибут” или ER-подход. Этот подход занимает доминирующую позицию и в настоящее время. Модель “сущность-связь-атрибут” основывается на некоторой важной семантической информации о реальном мире и относится к “семантическим моделям”.

Переход к этапу информационных технологий, основанному на моделировании и структуризации данных, характеризовался бурным ростом числа методов и методологий создания автоматизированных систем и становлением разных направлений в технологии программирования. К ним можно отнести методы структурного проектирования: методологию структурного анализа Йордана/де Марко и Гейна-Сарсона, метод Константайна, методологию структурного анализа и проектирования (SADT-методология). Метод организации потоков данных полнее всего описан в работах Джексона, а также в трудах Варнье (Уорниера) и Орра, методологию Дж. Мартина.

Методы и методологии этапа информационных технологий ориентированы на жесткую процедурность и структурность функциональной составляющей автоматизированной системы. Но анализ традиционных процессов решения интеллектуальных задач, трудно поддающихся автоматизации такими методами, позволил сделать вывод о необходимости пересмотра парадигмы моделирования интеллектуальной деятельности – от данных и процедур их обработки к категориям данных, к знаниям и стратегиям решений интеллектуальных задач на этих знаниях.

Переход к этапу новых информационных технологий, основанному на моделировании и обработке знаний, характеризовался бурным ростом числа методов и методологий создания автоматизированных интеллектуальных систем. Исследования в области работы со знаниями показали, что тех знаний, которые содержатся в источниках информации и отчужденных от специалиста, недостаточно для автоматизированного решения задач. Значительную часть своего профессионального опыта специалисты не могут четко изложить. Но извлечение знаний из источников и из памяти специалистов – это только часть проблемы. Представление этих знаний в автоматизированной среде на основе моделей – не менее трудная и трудоемкая проблема.

Методы и методологии, используемые для объектно-ориентированного проектирования изложены в работах Буча, Шлеера и Меллора, Коуда и Йордана.

Основными подходами к моделированию знаний, как было уже упомянуто, являются декларативный и процедуральный подходы. Эти подходы не исключают друг друга, а взаимно дополняют.

В первом подходе различают модели семантических сетей и фреймов. В основе моделирования знаний в виде семантических сетей лежит идея описания необходимой информации в виде бинарных связей между понятиями. При этом отношения могут иметь различную природу, обусловленную предметной областью автоматизируемой задачи. В задачах ситуационного управления – временные, пространственные и казуальные (причинные) связи между объектами, в системах планирования – связи типа «цель - средство», «цель - подцель», в системах классификации – отношения «род - вид», в системах функционирования – связи «вход – выход», «аргумент - функция». Теория семантических сетей не завершена, ее развитие привело к таким расширениям как растущие семантические сети, ассоциативные сети и др.

Представление знаний в виде фреймов трансформировалось от предложенного Мински как минимального описания, содержащего существенную информацию, до гибкой конструкции «имя фрейма – имя слота – значение слота», позволяющей подставить в значение слота любые константы, выражения, ссылки и другие фреймы.

Другой подход – процедуральный – опирается на классическую логическую модель вывода, которая представляет либо логические исчисления типа исчисления предикатов и его расширений, либо системы продукций, задающие элементарные шаги преобразований или умозаключений.

Сводная характеристика методов и методологий структурной методологии проектирования, ООП и КП приведена в таблице 2.2.

Анализ перечисленных методов и методологий показывает, что практически все вышеуказанные методы ориентированы на работу с предметными задачами средней сложности или размера, поэтому техника диаграмм для представления и концептуальных структур (ER-структур), и функциональных структур таких задач наиболее эффективна, так как обеспечивает наглядность, обозримость и простоту работы с этими представлениями. Но, как отмечается практически во всех методах и методологиях, очень сложной остается проблема взаимодействия между заказчиком, аналитиком и программистом прикладных систем. Эффективность такого взаимодействия определяется не только уровнем квалификации каждого специалиста в своей области, но и другими факторами, трудно поддающимися выявлению.

ЭТАПЫ РАЗВИТИЯ АВТОМАТИЗАЦИИ ИНТЕЛЛЕКТУАЛЬНОГО ТРУДА:

1. ЭЛЕКТРОННАЯ ОБРАБОТКА ДАННЫХ (интуитивное программирование)
2. ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ (традиционное, структурное программирование - на основе моделирования данных)
3. НОВЫЕ ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ (на основе фреймов и других видов программирования: логического, функционального, объектно-ориентированного и др.)
4. КОГНИТИВНЫЙ ПОДХОД

Таблица 2.2

№	**Реальный мир**	**Концептуальный мир**	**Информационный мир**					
			Организация информации и знаний		**Организация доступа**	**Организация обработки**	**Организация интерфейса**	
			данные	**категории**			**тип**	**реализация**
1	Свойства объектов	Атрибуты	Переменные/ данные	-	ЯП	ЯП	Символьный	ЯП
2	Объекты и их состояния, связи объектов и их свойств	ERA-подход	Структуры на переменных/ структуры данных (ЯОД)	-	ЯМД доступ к данным	ЯП/(ЯМД) управляющие конструкции	Символьный, графический	ЯП (позиционирование курсора, меню)
3	Объекты, связи объектов, свойства объектов, события	ERA-подход	Структуры на переменных/ структуры данных (фрейм/объект)	Структуры на переменных/ структуры данных (фрейм/объект)	ЯЛП или ЯФП, ЯЯП, ООЯП	ЯЛП или ЯФП, ЯЯП, ООЯП	Символьный, графический, звуковой	ЯП, ООЯП (позиционирование курсора, блока, выпадающие меню, окна)
4	Многоуровневое представление ситуаций или явлений: i-й уровень	Функциональные центры: К– контекстуальный, С – структурный, М – монадический. Закон цикличности: M(i)=K(i+1)	?	?	?	?	Символьный, графический, звуковой	?

Тема 6. Основы структурного подхода к анализу и моделированию бизнес-процессов.

Структурный подход основывается на алгоритмической декомпозиции, разложении более сложных задач на простые. Исторически сложились две методологии, работающие по принципу структурного подхода – CASE-методология и IDEF-методология.

IDEF-методологии

Наиболее распространенные методы IDEF-методологии задают единый подход к моделированию сложных систем и их составляющих, но они не предлагают увязку разнородных моделей в единое целое. Указанные методы, как правило, применяются на начальных этапах моделирования предметной области, а именно, при самостоятельном моделировании функциональных, процессных, информационных и других структур.

Семейство методов IDEF представлено набором следующих стандартов:

IDEF0 – FunctionModeling – метод функционального моделирования. IDEF0 является графическим языком, с помощью которого разработчики и аналитики описывают систему в виде набора взаимосвязанных функций. Данный метод часто применяют на ранних стадиях разработки проекта, для моделирования процесса «как есть». Результаты IDEF0 анализа в дальнейшем могут быть детализированы с помощью метода моделирования IDEF3.

IDEF1 – Information Modeling – метод моделирования информационных потоков внутри системы, позволяющий отображать и анализировать их структуру и взаимосвязи.

IDEF1X (IDEF1 Extended) – Data Modeling – метод построения реляционных структур. IDEF1X использует методологию Чена, позволяющую формировать отношения “сущность-связь” и, как правило, используется для моделирования реляционных баз данных, имеющих отношение к рассматриваемой системе.

IDEF2 – SimulatioModeling – метод динамического (имитационного) моделирования систем. Развитие данного метода было приостановлено и от него практически отказались в связи с большими сложностями анализа динамических систем. Однако в настоящее время существуют алгоритмы и их компьютерные реализации, которые позволяют превращать набор статических диаграмм IDEF0 в динамические модели, построенные на базе “раскрашенных сетей Петри” (CPN – ColorPetriNets).

IDEF3 – Process Description Capture – метод документирования процессов, происходящих в системе, который используется, например, при исследовании технологических процессов на предприятиях. С помощью IDEF3 описываются сценарий и последовательность операций для каждого процесса. IDEF3 имеет прямую взаимосвязь с методологией IDEF0 – каждая функция (функциональный блок) может быть представлена более детально в виде отдельного процесса средствами IDEF3.

IDEF4 – Object-oriented Design – метод построения объектно-ориентированных систем. Средства IDEF4 позволяют отображать структуру объектов и принципы их взаимодействия, что позволяет анализировать и оптимизировать сложные объектно-ориентированные системы.

IDEF5 – Ontology Description Capture – метод онтологического исследования сложных систем. С помощью методологии IDEF5 онтология системы описывается при помощи определенного словаря терминов и правил. На основании правил формируются достоверные утверждения о состоянии рассматриваемой системы в определенный момент времени, далее на основе этих утверждений формируются выводы о развитии системы и производится её оптимизация.

IDEF6 – Design Rationale Capture - использование рационального опыта проектирования.

IDEF7 – Information System Audit Method – метод анализа информационной системы.

IDEF8 – User Interface Modeling – моделирование интерфейса пользователя автоматизированной системы. Данный метод проектирует интерфейс пользователя создаваемой автоматизированной системы.

IDEF9 – Scenario-drivenInfoSysDesignSpec - метод исследования и описания предметных ограничений.

IDEF10 – Implementation Architecture Modeling – моделирование архитектуры выполнения.

IDEF11 – Information Artifact Modeling – информационное моделирование артефактов.

IDEF12 – Organization Modeling – организационное моделирование.

IDEF13 – Three Schema Mapping Design – трехсхемный дизайн карт.

IDEF14 – Network Design – метод проектирования компьютерных сетей, основанный на анализе требований, специфичных сетевых компонентов, существующих конфигураций сетей. Также он обеспечивает поддержку решений, связанных с рациональным управлением материальными ресурсами, что позволяет достичь существенной экономии.

Наиболее известными из них являются IDEF0, IDEF1, IDEF1X, IDEF3.

Тема 7. Основы объектно-ориентированного подхода к анализу и моделированию бизнес-процессов.

История создания UML. Методы объектно-ориентированной разработки

Первым общепризнанным объектно-ориентированным языком стал язык Simula-67, появившийся в 1967 году. Этот язык так и не получил широкого распространения, однако оказал сильное влияние на несколько других объектно-ориентированных языков, которые появились вскоре после него. В начале 80-х массовому интересу к объектно-ориентированным языкам способствовало распространение языка Smalltalk, за которым последовали и другие: Objective C, C++,, Eiffel и CLOSS. Приблизительно через пять лет после того, как обрел популярность язык Smalltalk, были опубликованы первые описание объектно-ориентированных методов. Вслед за первыми публикациями последовала книга Грэди Буча *(Спаду* Booch) "Object-oriented Analysis and Design with Applications" и еще несколько книг таких авторов, как Джеймс Рамбо (James Rumbaugh), Уильям Лоренсен (William Lorensen), Ребекка Вирфс-Брок (Rebecca Wirfs-Broek), Фредерик Эдди (Frederick Eddy) и другие. Вместе с более ранними изданиями, посвященными объектно-ориентированным языкам, эти работы начали процесс формирования методологии объектно-ориентированного программирования. Первая фаза этого процесса завершилась к концу 1990 года. Немногим позже появилась книга Айвара Якобсона (Ivar Jacobson) о процессе, названном Objectory (можно перевести как "создатель объектов"). В нем используется подход, в котором основное внимание уделяется вариантам использования и процессу разработки.

В следующие пять лет вышло множество книг по методологиям объектно-ориентированного программирования, причем в каждой использовались собственные понятия, определения, нотация и терминология. Авторы новых книг отталкивались от существовавших методов и просто вносили в них небольшие изменения и дополнения. В конце концов образовалось множество общих для всех базовых концепций и огромное количество концепций, встречающихся у одного - двух авторов и не получивших широкого распространения. Все это очень затрудняло процесс сравнения методов, особенно для обычного читателя.

UML был создан в результате работы по объединению и упрощению множества существующих методов объектно-ориентированной разработки.

Первая удачная попытка выработать обобщенный комбинированный метод была сделана, когда в 1994 году Джеймс Рамбо пришел в корпорацию Rational Software, где уже работал Грэди Буч. Они стали работать над объединением принципов OMT и метода Буча. Первые результаты появились уже в 1995 году. В это же время в компанию Rational Software перешел Якобсон и присоединился к Рамбо и Бучу. Результат их усилий и получил название Унифицированного языка моделирования (Unified Modeling Language - UML). Редкий случай совместной работы авторов трех ведущих методов над выработкой единого подхода сразу же изменил положение дел в объектно-ориентированной области программирования.

В 1996 году команда Object Management Group (OMG) объявила конкурс на создание стандартного метода объектно-ориентированного моделирования. Авторы языка UML начали сотрудничать с разработчиками и методологами из других компаний с целью выработать единый стандартный метод. Наконец, все возможные варианты слились в одну конечную версию UML, которая и была представлена OMG в сентябре.

В ноябре 1997 года OMG объявила UML стандартным языком объектно-ориентированного моделирования и приняла на себя обязанности по его дальнейшему развитию. Этот язык объединяет в себе опыт многих специалистов и, будучи официально признанным стандартом, устраняет неудобства использования разных несхожих инструментов.

UML предназначен для моделирования в различных предметных областях, включая моделирование больших и сложных систем, систем реального времени, распределенных систем и систем с интенсивными вычислениями или обработкой данных. Конечно, есть области, где лучше использовать специальные языки моделирования, но в общем UML разработан с таким расчетом, чтобы решать задачи не хуже любого языка моделирования общего назначения.

Введение в UML

Унифицированный язык моделирования (UML) - это язык визуального моделирования для решения задач общего характера, который используется при определении, визуализации, конструировании и документировании артефактов программной системы.

Он включает в себя семантические концепции, нотацию и руководящие указания. UML состоит из четырех частей, описывающих различные аспекты системы: статические, динамические, организационные и относящиеся к окружению.

Система моделируется как группа дискретных объектов, которые взаимодействуют друг с другом таким образом, чтобы удовлетворить требования пользователя. В статической структуре задаются типы объектов, значимые для системы и ее реализации, а также отношения между этими объектами. Динамическое поведение определяет историю объектов и их взаимодействие для достижения конечной цели. Наиболее полного и разностороннего понимания системы можно достичь при моделировании с различных, но взаимосвязанных точек зрения.

UML - дискретный язык моделирования. Он не предназначен для разработки непрерывных систем, встречающихся в физике и механике. UML создавался как язык моделирования общего назначения для применения в таких дискретных системах, как программное обеспечение, аппаратные средства или цифровая логика.

Проектирование на основе прецедентов состоит из следующих этапов:

- моделирование предметной области;
- моделирование прецедентов (вариантов использования);
- построение диаграмм последовательности.

Процесс ICONIX основан на прецедентах, с помощью этого процесса создаются вполне конкретные и простые для понимания прецеденты, которые легко использовать для разработки системы. В практической работе никогда не хватает времени для

моделирования, анализа и проектирования. Всегда требуется быстрый переход к кодированию, поскольку степень завершенности программных проектов традиционно оценивается по количеству написанных строк. Описываемый процесс придерживается минималистского подхода, обращая внимание на область между прецедентами и кодом. Он подзывает, что должно происходить в той точке жизненного цикла проекта, когда уже есть несколько прецедентов и необходимо заняться анализом и проектированием.

В объектно-ориентированных системах структура кода определяется классами - следовательно, нужно выяснить, какие классы понадобятся. С этой целью рисуют одну или несколько диаграмм классов. Для каждого класса необходимо описать все атрибуты, то есть данные, и операции, которые представляют собой функции программы. Также нужно показать, как функции и данные инкапсулируются в классах. Для иллюстрации способов организации и взаимодействия классов используются имеющиеся в UML диаграммы классов. В конечном счете будут получены очень детальные диаграммы классов уровня проектирования (имеется в виду такой уровень детализации, при котором диаграмма классов может служить шаблоном для создания кода - она должна точно отражать организацию будущей программы).

Один из самых сложных вопросов при разработке объектно-ориентированных систем заключается в распределении поведения, что подразумевает определение функций, из которых будет состоять программа. Требуется распределить диаграммы последовательности. Они разрабатываются отдельно для каждого сценария и показывают, какой объект отвечает за ту или иную функцию. На диаграмме последовательности видно, что экземпляры объектов взаимодействуют путем обмена сообщениями. Каждое сообщение приводит к вызову той или иной функции объекта-получателя.

Открытым остается вопрос о выявлении объектов, не учтенных при первой попытке спроектировать систему. Поэтому начинать следует с модели предметной области. Это своего рода словарь основных абстракций, то есть самых важных существительных в пространстве задачи - доменных объектов. В самом начале анализа и проектирования будет создана модель предметной области, в которой все доменные объекты будут изображены на одной большой диаграмме классов.

Таким образом, следует отталкиваться от модели предметной области, которая представляет собой диаграмму классов аналитического уровня, - первое приближение к статической структуре системы. Затем - ее обогащение, стремясь к получению детального проекта. Диаграмма классов -это статическое описание организации кода, тогда как прецеденты описывают динамическое поведение системы во время выполнения. Первые представления о системе фиксируются в виде статической модели, затем начинается исследование различных прецедентов. При проработке каждого из них происходит уточнение диаграммы классов. После рассмотрения всех сценариев, которые должна поддерживать система, добавления необходимых деталей и повторного рецензирования результата должен получиться проект, удовлетворяющий всем требованиям. После этого можно приступать к написанию кода.

Укрупненная схема процесса ICONIX представлена на рис. 2.2.

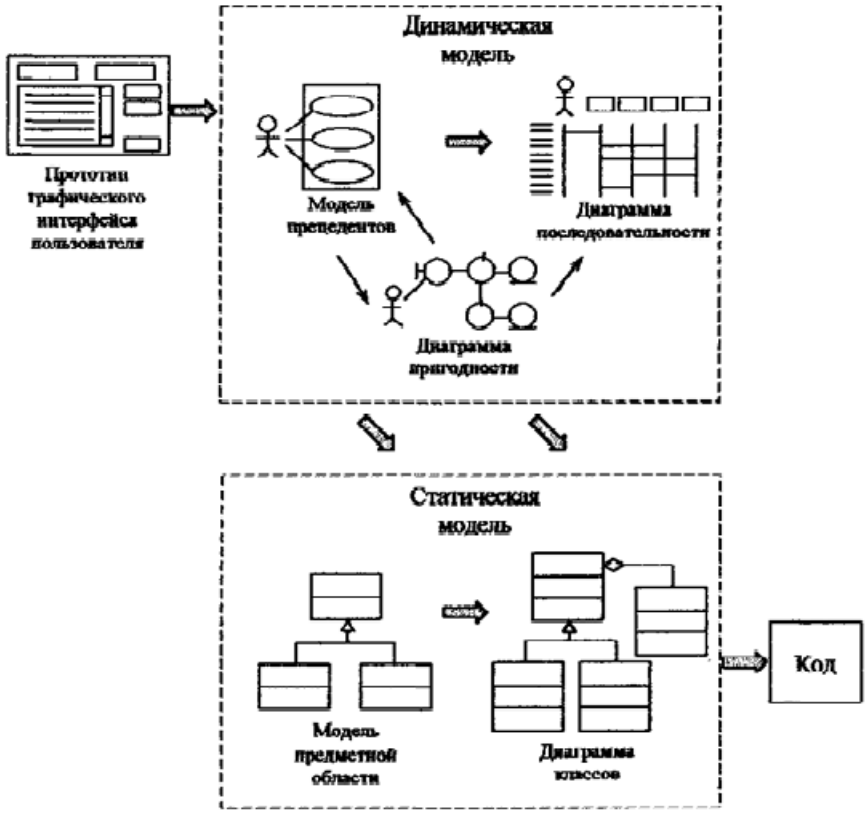

Рисунок 2.2. Процесс ICONIX – простой процесс моделирования с помощью UML.

Назначение UML

Разработка языка UML преследовала несколько различных целей. Прежде всего, UML создавался как язык моделирования общего характера, которым могут пользоваться все, занимающиеся моделированием. UML не является частной собственностью, он основан на соглашении большинства специалистов в компьютерной области. В него были включены основные понятия из наиболее известных методологий, поэтому он может использоваться совместно с ними. UML разрабатывался для поддержки таких ценных наработок, как инкапсуляция, разделение сущностей и выявление сути конструкции модели. UML отвечает требованиям современной разработки программного обеспечения, в том числе таким, как крупномасштабность, параллелизм, распределенность, возможность использования образцов и удобство при командной работе.

По мере выполнения работ, необходимо придерживаться набора принципов, которые обеспечивают достижение поставленных целей. В качестве таких принципов обычно выделяют:

- абстрагирование (абстракция);
- иерархия;
- сокрытие информации (инкапсуляция);
- модульность;
- *типизация (единообразие);*
- *полнота,*
- *подтверждаемость*.

1. ***Абстрагирование*** (абстракция) - выделение существенных особенностей объекта, отличающих его от всех других объектов.

Примечание: Этот принцип присутствовал еще в ранних языках программирования. Первая абстракция – процедура, реализуемая в виде подпрограммы, прямо вытекала из

традиционного взгляда на программные средства. Программы имели относительно простую структуру, состоящую из глобальных данных и подпрограмм. В процессе разработки программ была возможность логического разделения различных данных, но механизмы языков его не поддерживали. Ошибка имела серьезные последствия, так как язык не гарантировал целостность данных в процессе внесения значений, что нарушало надежность системы. Применение процедур решило ряд противоречий; усилило управление алгоритмическими абстракциями, но не решило задачи использования многообразия данных.

Абстракция – одно из основных средств для управления сложными системами ПО. Суть абстракции состоит в выделении существенных свойств (характеристик) некоторого объекта, отличающих его от всех других видов объектов и, таким образом, четко определяет его концептуальные границы с точки зрения наблюдателя. Абстрагирование применяется как к данным, так и к алгоритмам. Абстрагирование позволяет отделить существенные особенности от несущественных. Абстракция фокусируется на существенных характеристиках объекта (с точки зрения наблюдателя), которые в зависимости от наблюдателя могут быть разными. Так, метод, описывающий транспортный поток деталей на гибкой производственной линии в некоторых случаях может не заботиться о материале, из которого они изготовлены, тогда как метод, управляющий металлообработкой, будет учитывать это свойство.

2. ***Иерархия*** – упорядоченная система абстракций. Структурная иерархия отражает взаимосвязи типа «часть - целое» и определяет отношение агрегирования между объектами. *Порядок подчиненности низших звеньев высшим*. (Агрегация – процедура построения нового объекта из исходных. При этом исходные объекты независимы или разнородны, отражают разные качества нового объекта.)

3. ***Сокрытие или упрятывание информации*** имеет целью сделать недоступными детали, которые могут повлиять на остальные, более существенные части системы. Упрятывание информации обычно скрывает реализацию объекта или операции и, таким образом, позволяет фиксировать внимание на более высоком уровне абстракции. *Скрытие описания реализации объекта (например, модуля программы, типа данных) от использующих его модулей. Фактически, это защита информации, реализуемая посредством управления доступом к информации через методы объекта. Внутренняя структура объекта скрыта от пользователя, а объекты считаются самостоятельными сущностями.*

Инкапсуляция дополняет понятие абстракции: абстрагирование направлено на наблюдаемое поведение объекта, а инкапсуляция занимается его внутренним устройством. Обычно скрываются и внутренняя структура объекта, и реализация его методов. Инкапсуляция необходима для определения четких границ между абстракциями. При проектировании базы данных, например, принято писать программы так, чтобы они не зависели от физического представления данных; вместо этого сосредоточиваются на схеме, отражающей логическое строение самих данных. Таким образом, объекты, находящиеся выше в иерархической цепочке, защищаются от деталей реализации объектов более низкого уровня.

Абстракция и сокрытие информации способствуют модифицируемости и понимаемости систем ПО. На каждом уровне абстракции допускается применение лишь определенного набора операций и делается невозможным применение любых операций, нарушающих логическую концепцию данного уровня, за счет чего повышается надежность программной системы.

4.***Модульность*** – это свойство системы, которая была разложена на связанные внутренне, но мало связанные между собой и часто скрытые друг от друга группы, в которых описывается поведение объектов и процессов. В общем плане модули могут быть функциональными (процедурно-ориентированными) или декларативными (объектно-ориентированными). В идеале должны разрабатываться слабо связанные модули. При разработке необходимо учитывать, что модуль – неделимый блок, который можно использовать

в программе повторно. Прочность (свойство модульности, помимо связности модулей) определяет насколько сильно связаны элементы внутри модуля.

5. *Единообразие* (типизация) – способ защититься от применения одних типов объектов вместо других (типовые конструкции, типовые абстракции и т.д.), а также способ управлять таким использованием. Это ограничение на выполнение операций, препятствующих взаимозамене различных типов объектов. Это идея согласования типов объектов.

6. Полнота – заключается в контроле на присутствие лишних элементов. Фактически должно быть определено количество информации, необходимой для принятия решения.

7. Подтверждаемость (сохраняемость) – способность объекта существовать во времени и пространстве до и после породившего его процесса.

По-другому можно сказать, что использование языка UML основывается на общих принципах моделирования:

- ***абстрагирование*** - в модель следует включать только те элементы проектируемой системы, которые имеют непосредственное отношение к выполнению ей своих функций или своего целевого предназначения. Другие элементы опускаются, чтобы не усложнять процесс анализа и исследования модели;
- ***многомодельность*** - никакая единственная модель не может с достаточной степенью точности описать различные аспекты системы. Допускается описывать систему некоторым числом взаимосвязанных представлений, каждое из которых отражает определенный аспект её поведения или структуры;
- ***иерархическое построение*** – при описании системы используются различные уровни абстрагирования и детализации в рамках фиксированных представлений. При этом первое представление системы описывает её в наиболее общих чертах и является представлением концептуального уровня, а последующие уровни раскрывают различные аспекты системы с возрастающей степенью детализации вплоть до физического уровня. Модель физического уровня в языке UML отражает компонентный состав проектируемой системы с точки зрения ее реализации на аппаратурной и программной платформах конкретных производителей.

Сущности в UML

В UML определены четыре типа сущностей: ***структурные***, ***поведенческие***, ***группирующие*** и ***аннотационные***. Сущности являются основными объектно-ориентированными элементами языка, с помощью которых создаются модели.

Структурные сущности - это имена существительные в моделях на языке UML. Как правило, они представляют статические части модели, соответствующие концептуальным или физическим элементам системы. Примерами структурных сущностей являются «класс», «интерфейс», «кооперация», «прецедент», «компонент», «узел», «актер».

Поведенческие сущности являются динамическими составляющими модели UML. Это глаголы, которые описывают поведение модели во времени и в пространстве. Существует два основных типа поведенческих сущностей:

- *взаимодействие* - это поведение, суть которого заключается в обмене сообщениями между объектами в рамках конкретного контекста для достижения определенной цели;
- *автомат* - алгоритм поведения, определяющий последовательность состояний, через которые объект или взаимодействие проходят в ответ на различные события.

Группирующие сущности являются организующими частями модели UML. Это блоки, на которые можно разложить модель. Такая первичная сущность имеется в единственном экземпляре - это пакет.

Пакеты представляют собой универсальный механизм организации элементов в группы. В пакет можно поместить структурные, поведенческие и другие группирующие сущности. В отличие от компонентов, которые реально существуют во время работы программы, пакеты носят чисто концептуальный характер, то есть существуют только в процессе разработки.

Аннотационные сущности – это пояснительные части модели UML: комментарии для дополнительного описания, разъяснения или замечания к любому элементу модели. Имеется только один базовый тип аннотационных элементов - примечание. Примечание используют, чтобы снабдить диаграммы комментариями или ограничениями, выраженными в виде неформального или формального текста.

Отношения в UML

В языке UML определены следующие типы отношений: ***зависимость***, ***ассоциация***, ***обобщение*** и ***реализация***. Эти отношения являются основными связующими конструкциями UML и также как сущности применяются для построения моделей.

Зависимость (dependency) - это семантическое отношение между двумя сущностями, при котором изменение одной из них, независимой, может повлиять на семантику другой, зависимой.

Ассоциация (association) - структурное отношение, описывающее совокупность смысловых или логических связей между объектами.

Обобщение (generalization) - это отношение, при котором объект специализированного элемента (потомок) может быть подставлен вместо объекта обобщенного элемента (предка). При этом, в соответствии с принципами объектно-ориентированного программирования, потомок наследует структуру и поведение своего предка.

Реализация (realization) является семантическим отношением между классификаторами, при котором один классификатор определяет обязательство, а другой гарантирует его выполнение. Отношение реализации встречаются в двух случаях:

- между интерфейсами и реализующими их классами или компонентами;
- между прецедентами и реализующими их кооперациями.

Общие механизмы UML

Для точного описания системы в UML используются, так называемые, ***общие механизмы***:

- ***спецификации*** (specifications);
- ***дополнения*** (adornments);
- ***деления*** (common divisions);
- ***расширения*** (extensibility mechanisms).

UML является не только графическим языком. За каждым графическим элементом его нотации стоит ***спецификация***, содержащая текстовое представление соответствующей конструкции языка. Например, пиктограмме класса соответствует спецификация, которая описывает его атрибуты, операции и поведение, хотя визуально, на диаграмме, пиктограмма часто отражает только малую часть этой информации. Более того, в модели может присутствовать другое представление этого класса, отражающее совершенно иные его аспекты, но, тем не менее, соответствующее спецификации. Таким образом, графическая нотация UML используются для визуализации системы, а с помощью спецификаций описывают ее детали.

Практически каждый элемент UML имеет уникальное графическое изображение, которое дает визуальное представление самых важных его характеристик. Нотация сущности «класс» содержит его имя, атрибуты и операции. Спецификация класса может содержать и другие детали, например, видимость атрибутов и операций, комментарии или указание на то, что класс является абстрактным. Многие из этих деталей можно визуализировать в виде графических или текстовых ***дополнений*** к стандартному прямоугольнику, который изображает класс.

При моделировании объектно-ориентированных систем существует определенное ***деление*** представляемых сущностей.

Во-первых, существует деление на классы и объекты. Класс - это абстракция, а объект - конкретное воплощение этой абстракции. В связи с этим, практически все конструкции языка характеризуются двойственностью «класс/объект». Так, имеются прецеденты и экземпляры прецедентов, компоненты и экземпляры компонентов, узлы и экземпляры узлов. В графическом представлении для объекта принято использовать тот же символ, что и для класса, а название подчеркивать.

Во-вторых, существует деление на интерфейс и его реализацию. Интерфейс декларирует обязательства, а реализация представляет конкретное воплощение этих обязательств и обеспечивает точное следование объявленной семантике. В связи с этим, почти все конструкции UML характеризуются двойственностью «интерфейс/реализация». Например, прецеденты реализуются кооперациями, а операции - методами.

UML является открытым языком, то есть допускает контролируемые ***расширения***, чтобы отразить особенности моделей предметных областей. ***Механизмы расширения*** UML включают:

- ***стереотипы*** (stereotype) - расширяют словарь UML, позволяя на основе существующих элементов языка создавать новые, ориентированные для решения конкретной проблемы;
- ***помеченные значения*** (tagged value) - расширяют свойства основных конструкций UML, позволяя включать дополнительную информацию в спецификацию элемента;
- ***ограничения*** (constraints) - расширяют семантику конструкций UML, позволяя создавать новые и отменять существующие правила.

Совместно эти три механизма расширения языка позволяют модифицировать его в соответствии с потребностями проекта или особенностями технологии разработки.

Виды диаграмм UML

Графические изображения моделей системы в UML называются ***диаграммами***. В терминах языка UML определены следующие их виды:

- ***диаграмма вариантов использования или прецедентов*** (use case diagram)
- ***диаграмма классов*** (class diagram)
- ***диаграммы поведения*** (behavior diagrams)
  - ***диаграмма состояний*** (statechart diagram)
  - ***диаграмма деятельности*** (activity diagram)
- ***диаграммы взаимодействия*** (interaction diagrams)
  - ***диаграмма последовательности*** (sequence diagram)
  - ***диаграмма кооперации*** (collaboration diagram)
- ***диаграммы реализации*** (implementation diagrams)
  - ***диаграмма компонентов*** (component diagram)
  - ***диаграмма развертывания*** (deployment diagram)

Каждая из этих диаграмм конкретизирует различные представления о модели системы. При этом, диаграмма вариантов использования представляет концептуальную модель системы, которая является исходной для построения всех остальных диаграмм. Диаграмма классов является логической моделью, отражающей статические аспекты структурного построения системы, а диаграммы поведения, также являющиеся разновидностями логической модели, отражают динамические аспекты её функционирования. Диаграммы реализации служат для представления компонентов системы и относятся к ее физической модели.

Из перечисленных выше диаграмм некоторые служат для обозначения двух и более подвидов. В качестве же самостоятельных представлений используются следующие диаграммы: ***вариантов использования***, ***классов***, ***состояний***, ***деятельности***, ***последовательности***, ***кооперации***, ***компонентов*** и ***развертывания***.

Для диаграмм языка UML существуют три типа визуальных обозначений, которые важны с точки зрения заключенной в них информации:

- ***связи***, которые представляются различными линиями на плоскости;
- ***текст***, содержащийся внутри границ отдельных геометрических фигур;
- ***графические символы***, изображаемые вблизи визуальных элементов диаграмм.

При графическом изображении диаграмм рекомендуется придерживаться следующих правил:

- каждая диаграмма должна быть законченным представлением некоторого фрагмента моделируемой предметной области;
- представленные на диаграмме сущности модели должны быть одного концептуального уровня;
- вся информация о сущностях должна быть явно представлена на диаграмме;
- диаграммы не должны содержать противоречивой информации;
- диаграммы не следует перегружать текстовой информацией;
- каждая диаграмма должна быть самодостаточной для правильной интерпретации всех ее элементов;
- количество типов диаграмм, необходимых для описания конкретной системы, не является строго фиксированным и определяется разработчиком;
- модели системы должны содержать только те элементы, которые определены в нотации языка UML.

Контрольные вопросы:

1. Почему нельзя оценивать программный продукт вне контекста задач, для решения которых этот продукт предназначен?
2. Как связаны между собой понятия «бизнес-процесс», «производственный процесс» и «технологический процесс»?
3. Подходы, применяемые при разработке программных продуктов.
4. Особенности структурного подхода.
5. Каковы область эффективного применения, сильные и слабые стороны технологии анализа и моделирования бизнес-процессов IDEF и почему?
6. Как качество моделирования бизнес-процессов влияет на эффективность функционирования программного продукта в информационной среде предприятия?
7. В чем состоит основное отличие объектно-ориентированного подхода к анализу и моделированию предметных задач от других подходов аналогичного назначения?
8. Особенности объектно-ориентированного подхода.
9. Что называется «решением задачи» с точки зрения объектно-ориентированного подхода? Что такое «стратегия решения задачи», и чем она отличается от алгоритма?

Список литературы

1.Зубкова, Т.М. Технология разработки программного обеспечения: учебное пособие / Т.М. Зубкова; Министерство образования и науки Российской Федерации, Федеральное государственное бюджетное образовательное учреждение высшего образования «Оренбургский государственный университет», Кафедра программного обеспечения вычислительной техники и автоматизированных систем. - Оренбург: ОГУ, 2017. - 469 с.: ил. - Библиогр.: с. 454-459. - ISBN 978-5-7410-1785-2; То же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=485553.

2.Соловьев, Н. Системы автоматизации разработки программного обеспечения: учебное пособие / Н. Соловьев, Е. Чернопрудова ; Министерство образования и науки Российской Федерации, Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Оренбургский государственный университет». - Оренбург:

ОГУ, 2012. - 191 с. : ил., схем., табл. - Библиогр.: с. 182-183. То же [Электронный ресурс]. – URL:http://biblioclub.ru/index.php?page=book&id=270302.

3.Антамошкин, О.А. Программная инженерия. Теория и практика : учебник / О.А. Антамошкин ; Министерство образования и науки Российской Федерации, Сибирский Федеральный университет. - Красноярск : Сибирский федеральный университет, 2012. - 247 с. : ил., табл., схем. - Библиогр.: с. 240. - ISBN 978-5-7638-2511-4 ; То же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=363975.

4.Программная инженерия: учебное пособие / сост. Т.В. Киселева ; Министерство образования и науки РФ, Федеральное государственное автономное образовательное учреждение высшего образования «Северо-Кавказский федеральный университет». - Ставрополь : СКФУ, 2017. - Ч. 1. - 137 с. : ил. - Библиогр. в кн. ; То же [Электронный ресурс]. - URL:http://biblioclub.ru/index.php?page=book&id=467203.

Дополнительно

5. Леоненков А.В. «Самоучитель UML 2» - СПб.: БХВ-Петербург, 2007. – 576 с.: ил.

6. Орлов С. А. Программная инженерия: Технологии разработки программного обеспечения. Учебник для вузов. 5-е издание обновлённое и дополненное [Текст] – СПб.: Питер, 2016. – 640 с.: ил. – ISBN 978-5-496-01917-0 Режим доступа URL: https://www.studmed.ru/orlov-sa-tehnologiya-razrabotki-programmnogo-obespecheniya_fc460ac2b04.html (дата обращения 04.07.2019)