Федеральное агентство железнодорожного транспорта
Уральский государственный университет путей сообщения

К.А. Паршин
Е.В. Паршина

ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ

Екатеринбург
2010

Федеральное агентство железнодорожного транспорта
Уральский государственный университет путей сообщения
Кафедра «Связь»

К.А. Паршин
Е.В. Паршина

ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ

Конспект лекций
для студентов 5 курса очного обучения
и 6 курса заочного обучения
по специальности – 071900 «Информационные системы
(на железнодорожном транспорте)»

Екатеринбург
2010

УДК 656.2
П18

Паршин К.А.
П18 Проектирование информационных систем : конспект лекций / К.А. Паршин, Е.В. Паршина. – Екатеринбург : УрГУПС, 2010. – 100 с.

Конспект лекций содержит основные сведения по методам и принципам проектирования информационных систем, а также разработке их технической документации.

Пособие предназначено для студентов 5 курса очной и 6 курса заочной формы обучения по специальности 071900 «Информационные системы (на железнодорожном транспорте)».

УДК 656.2

Утверждено на заседании кафедры «Связь» 22.01.2009 г. протокол № 16.

Авторы:
К.А. Паршин, доцент кафедры «Связь», канд. техн. наук, УрГУПС
Е.В. Паршина, ведущий инженер НИЧ НИЛ «Компьютерные системы автоматики», УрГУПС

Рецензенты:
Д.Г. Неволин, профессор кафедры «Связь»,д-р техн. наук, УрГУПС
А.В. Красновидов, доцент кафедры «Информационные вычислительные системы», канд. техн. наук, ПГУПС

Оглавление

Лекция 1

Основные понятия и классификация информационных систем

1.1 Основные понятия

Целью изучения дисциплины является освоение технологий проектирования информационных систем (ИС), базирующееся на знаниях и умениях в области вычислительной техники и методов программирования, теории информационных систем и теории надежности, управления и представления данных, методов и средств обеспечения данных.

В результате изучения дисциплины каждый студент должен:

1) получить знания об этапах проектирования ИС, их содержании;
2) изучить нормативные документы, на которых базируется процесс проектирования ИС;
3) научиться принимать решения о целесообразности создания ИС;
4) научиться формировать требования к ИС;
5) изучить принципы структурного и объектно-ориентированного подхода к проектированию ИС;
6) получить представления об инструментальных средствах, используемых при проектировании ИС.

Знание дисциплины начинается со знания основных терминологий, используемых в рамках данной предметной области. Самый основополагающий термин – это информация.

Информация – сведения о тех или иных предметах, явлениях, процессах, событиях.

В процессе своей жизнедеятельности мы постоянно участвуем в информационных процессах, непосредственно связанных с информацией.

Информационные процессы – процессы сбора, обработки, накопления, хранения, поиска и распространения информации.

В данном аспекте большое значение приобретает понятие документированной информации или документа. В большинстве случаев информация предоставляется в виде документа – зафиксированной на материальном носителе информации с реквизитами, позволяющими ее идентифицировать (исключая случаи, когда информация поступает не в виде документа, а в виде каких-либо физических величин).

Информация не только существует, но и обрабатывается.

Информационные технологии – совокупность методов и способов, позволяющих обрабатывать информацию, в том числе с использованием аппаратных и программных средств.

Как правило, с помощью информационных технологий информация документируется, т. е. превращается в информационные ресурсы, лежащие в основе ИС.

Информационный ресурс – это отдельный документ или массив документов в информационной системе.

Информационная система – организационно упорядоченная совокупность документов (их массивов) и информационных технологий, в том числе с использованием средств вычислительной техники и связи, реализующих информационные процессы.

Пример информационных систем: библиотека, архив, фонд, банк данных. Концентрация информационных ресурсов нескольких ИС на основе информационно-коммуникационного взаимодействия создает единое информационное пространство (ЕИП).

Наиболее удобным способом обработки, сбора, поиска, хранения информации является автоматизированный способ.

Автоматизированная информационная система (АИС) – информационная система, в которой представление, хранение и обработка информации осуществляется с помощью вычислительной техники.

В АИС информационные технологии всегда выполняются средствами вычислительной техники (ВТ).

Как правило, при проектировании систем говорят об автоматизации человеческой деятельности, в том числе по обработке информации. И автоматизированные системы воспринимают как системы, состоящие из персонала и комплекса средств автоматизации его деятельности, выполняющих информационную технологию.

АИС реализуют информационные технологии в виде определенной последовательности информационно связанных функций и задач, выполняемых в автоматизированном или автоматическом режиме. В частности, одной из установленных функций автоматизированной системы, реализуемых с помощью информационных технологий, может являться также документирование информации.

Далее, говоря об информационных системах, будем всегда подразумевать автоматизированные информационные системы (АИС).

1.2 Состав ИС

В состав ИС входит комплекс средств автоматизации или программно-техническая часть (КСА), организационно-методические и технические документы и специалисты.

Одним из программно-технических элементов ИС может являться автоматизированное рабочее место (АРМ) или несколько АРМов, которые автоматизируют деятельность определенного вида (АРМ бухгалтера, АРМ технолога и т.д.)

Для КСА характерно наличие следующих подсистем – частей автоматизированной системы, выделенных по функциональному или структурному признаку, отвечающему конкретным целям и задачам:

1) подсистема сбора информации;
2) подсистема представления и обработки информации;

3) функциональная подсистема выдачи информации.

Подсистема сбора информации обеспечивает отбор и накопление данных для ИС и включает совокупность источников информации и организационно-технологические цепочки отбора информации для накопления в системе (определяет эффективность функционирования системы в целом).

Подсистема представления и обработки информации – это ядро информационной системы. Представляет собой отображение предметной области, представленное разработчиками системы. Один из наиболее сложных компонентов для разработки.

Функциональная подсистема выдачи информации реализует целевой аспект назначения системы путем выполнения поставленных перед системой задач и выдает результаты.

Информационным ядром подсистемы представления и обработки информации является, как правило, база данных (БД) – носитель информации о предметной области.

База данных – совокупность данных, организованная по определенным правилам с общими принципами описания, хранения и манипулирования данными, независимая от прикладных программ. Данные систематизируются так, чтобы они могли быть обработаны с помощью ЭВМ.

Комплекс программных средств, реализующих создание БД, их поддержание в актуальном состоянии, а также обеспечивающий различным категориям пользователей возможность получать из БД необходимую информацию называется системой управления базой данных (СУБД).

1.3 Классификация ИС

ИС можно классифицировать по следующим критериям:

1. По сложности создания и сопровождения.
2. По концептуальной модели использования (целевое назначение системы).
3. По характеру представления и логической организации хранимой информации (как организована база данных).
4. По способу реализации БД (как реализована база данных).

Сложность создания и сопровождения. Специфика решаемых ИС задач, сложность их создания, сопровождения и время жизни ИС позволяют разделить их на три класса.

1. Малые.
2. Средние.
3. Крупные – корпоративные ИС (КИС).

Основные признаки малых ИС: короткий жизненный цикл, ориентация на массовое использование, невысокая цена, однородность аппаратного и программного обеспечения, сопровождение разработчиком для модификаций.

Для средних ИС характерны: более длительный жизненный цикл, наличие аналитической обработки данных, администрирование штатом сотрудников, наличие средств обеспечения безопасности.

Для крупных систем характерны: длинный жизненный цикл, наследование, разнообразие аппаратного и программного обеспечения, масштабность решаемых задач, территориальная распределенность.

Концептуальная модель использования. Разработка и проектирование информационной системы начинаются с построения концептуальной модели ее использования. Концептуальная модель использования ИС определяет круг конкретных задач и функций, обеспечиваемых созданием и эксплуатацией информационной системы, а также систему сбора, накопления и выдачи информации.

Следовательно, критерием классификации ИС могут являться функции и решаемые задачи.

Основная классификация систем приводится в РД 50-680-88, но на практике чаще прибегают к следующей категоризации АС в зависимости от сферы автоматизируемой деятельности:

1. АС управления (АСУП, АСУ ТП, СППР, документооборот).
2. Системы автоматизированного проектирования (САПР).
3. Системы сбора и передачи информации (ССПИ).
4. Информационно-справочные системы (ИСС).
5. Интегрированные информационные системы (ИИС).
6. Прочие, в том числе, сочетание выше перечисленных.

Характер представления и логической организации хранимой информации. В данной категории ИС подразделяются на следующие виды:

1. Фактографические – накапливают и хранят данные в виде множества экземпляров (записей) одного или нескольких типов структурных элементов (информационных объектов). Структура каждого типа информационного объекта состоит из конечного набора реквизитов, отражающих основные сведения о предметной области. Например, запись о человеке может состоять из таких реквизитов, как фамилия, имя, отчество. Каждому человеку в базе сопоставляется запись – кортеж. В таких системах требуется обязательная структуризация входной информации.

2. Документальные – единичный элемент информации – нерасчлененный на более мелкие элементы документ (неделимый). На входе информация не структурируется. Документ может характеризоваться некоторыми формальными критериями: дата изготовления, тематика.

3. Геоинформационные – реализованы как отдельные информационные объекты с реквизитами плюс привязка к электронной карте. Характерны для территориально разнесенных информационных процессов и объектов (электронные карты, транспорт).

Способ реализации БД. Классификация ИС в зависимости от способа реализации БД может быть представлена следующим образом (рис. 1):

Рис. 1. Классификация ИС по способу реализации БД

Внутреннее состояние ИС описывается структурами (элемент – связь):

1. Функциональными (функции, задачи, процедуры – информация).

Техническими (устройства, комплексы – линии и каналы связи).

2. Организационными (коллективы людей, люди – информационные, соподчинения и взаимодействия).

3. Документальными (неделимые части АС и документы – взаимодействия, входимость, соподчинения).

4. Алгоритмическими (алгоритмы – информация).

5. Программными (программные модули – управление).

6. Информационными (формы существования информации в системе – операции ее преобразования).

Выполнение вышеперечисленных схем производится по ГОСТ 24.302-80.

Лекция 2

Жизненный цикл информационных систем

Создание любой информационной системы или ее компонента проходит определенные стадии и по определенному образцу от момента зарождения идеи создания системы с определенными требованиями до ее полной утилизации. Период зарождения, развития, существования и снятия системы с эксплуатации представляет собой жизненный цикл информационной системы (или жизненный цикл автоматизированной системы).

Жизненный цикл автоматизированной системы – это совокупность взаимосвязанных процессов создания и последовательного изменения состояния ИС, от формирования исходных требований к ней до окончания эксплуатации и утилизации комплекса средств автоматизации (ГОСТ 34.003).

Создание автоматизированных систем регламентируется «Комплексом стандартов и руководящих документов на автоматизированные системы».

Основополагающие ГОСТы:

34.601 – Этапы и стадии создания АС.

34.602 – Техническое задание на автоматизированные системы.

34.603 – Методы испытаний автоматизированных систем.

ГОСТ 34.601 определяет стадии создания АС общего вида, является как бы универсальным и поэтому не всегда в полной мере может отобразить тонкости разработки и учесть новые методологии, появившиеся в последнее время. Теперь процесс создания и сопровождения прикладного программного обеспечения, входящего в информационную систему и самой системы, раскрывается через модель жизненного цикла. В настоящее время появилось большое количество различных моделей жизненного цикла, отличающихся от регламентированной по ГОСТ 34.601.

Модель жизненного цикла – структура, состоящая из процессов, работ и задач, включающих в себя разработку, эксплуатацию, и сопровождение программного продукта, охватывающая жизнь системы от установления требований к ней до прекращения ее использования (ГОСТ Р ИСО/МЭК 12207).

Системы железнодорожного транспорта также имеют свои особенности, поэтому институтом ВНИИАС при участии других организаций железнодорожного транспорта был создан комплект отраслевых руководящих методических материалов (ОРММ), связанных с созданием информационных систем на железнодорожном транспорте:

- процессы жизненного цикла информационных систем и программных средств (ОРММ ИСЖТ 5.03-00);
- требования к составу, содержанию и оформлению документов при создании информационных систем (ОРММ ИСЖТ 2.01-00);
- порядок представления, согласования и утверждения документов, разрабатываемых при создании информационных систем (ОРММ ИСЖТ 2.02-00);
- порядок внесения изменений в программное и информационное обеспечение эксплуатируемых систем и их компонентов (ОРММ ИСЖТ 5.01-00).

Данные материалы разработаны на базе государственных стандартов, но дополнены в соответствии с новыми технологиями в области проектирования информационных систем и являются очень удобными для понимания и использования.

При этом для комплексного проектирования системы рекомендуется совместно использовать стандарты ИСО/МЭК 12207, ОРММ ИСЖТ 5.03-00 и стандарты серии ГОСТ 34.

Основными стадиями создания информационных систем для железнодорожного транспорта (согласно ОРММ ИСЖТ 5.03-00) являются следующие (таблица 1):

Таблица 1

Стадии	*Этапы создания*	*Содержание стадии*
1. Формирование требований к АС	1.1. Обследование объекта и обоснование необходимости создания АС 1.2. Формирование требований пользователя к АС 1.3. Оформление отчета о выполненной работе и заявки на разработку АС (тактико-техническое задание)	Формирование требований – сбор данных и анализ объекта, для поддержки которого предполагается создание АС, анализ существующей информационной системы (изучение информационных потоков, выявление недостатков). Обоснование целесообразности создания АС
2. Разработка концепции АС	2.1. Изучение объекта 2.2. Проведение необходимых научно-исследовательских работ 2.3. Разработка вариантов концепции АС и выбор варианта концепции АС, удовлетворяющего требованиям пользователя 2.4. Оформление отчета о выполненной работе	Создается концепция проектируемой системы, удовлетворяющая требованиям пользователя (структура, функции, программно-техническая платформа, режимы). Рассматриваются альтернативные варианты, проводится анализ, выбирается лучшая концепция
3. Техническое задание ТЗ	3.1. Разработка и утверждение технического задания на создание АС	Разрабатывается ТЗ, основа которого – требования к системе
4. Эскизный проект ЭП	4.1. Разработка предварительных проектных решений по системе и ее частям 4.2. Разработка документации на АС и ее части	Может быть объединен с техническим проектом
5. Пилот-проект (П-П)	5.1. Разработка частей проекта (общесистемной, информационной, программной, технической) для испытаний в реальных, но ограниченных условиях функционирования с целью проверки предварительно принятых решений по основ-	Разрабатывается при необходимости (этот раздел только по документу ОРММ ИСЖТ 5.03-00)

	ным положениям создаваемой системы. Моделирование ИС (при необходимости) 5.2. Разработка документации ПП 5.3. Проведение испытаний на головном объекте или стенде 5.4. Анализ результатов проектирования и определение ресурсной способности системы	
6. Технический проект ТП	6.1. Разработка проектных решений по системе и ее частям 6.2. Разработка документации на АС и ее части 6.3. Разработка и оформление документации на поставку изделий для комплектования АС и (или) технических требований на их разработку 6.4. Разработка задания на проектирование в смежных частях проекта по автоматизации объекта	Определение функциональной структуры, выбор комплекса технических средств, выбор СУБД и проектирование базы данных, входных и выходных форм; разработка технологии обработки информации для выполнения требований, предъявляемых к данным, и алгоритмов обработки данных при выполнении различных функций
7. Рабочая документация	7.1 Разработка рабочей документации на систему и ее части 7.2. Разработка и адаптация программ	Адаптация приобретаемых программных средств, разработка вновь создаваемых программ, подготавливаются сведения, необходимые для ввода системы в действие и ее эксплуатации
8. Интеграция и тестирование	Загрузка БД исходными данными и тестами. Интеграция ПС с аппаратными средствами в реальной операционной системе и внешней среде	
9. Ввод в действие	8.1. Подготовка объекта автоматизации к вводу АС в действие	Необходимая доработка системы по результатам опытной эксплуатации

	8.2. Подготовка персонала 8.3. Комплектация АС поставляемыми изделиями (программными и техническими средствами, программно-техническими комплексами, информационными изделиями) 8.4. Строительно-монтажные работы 8.5. Пусконаладочные работы 8.6. Проведение предварительных испытаний 8.7. Проведение опытной эксплуатации 8.8. Проведение приемочных испытаний	
10.Тиражирование	ПО передается в ОФАП. Обучаются и консультируются пользователи. ПО поставляется на объекты внедрения	
11.Сопровождение АС	9.1. Выполнение работ в соответствии с гарантийными обязательствами 9.2. Послегарантийное обслуживание	Анализ функционирования АС, выявляются отклонения эксплуатационных характеристик и устанавливаются их причины. Вносятся изменения в документацию Консультация пользователей Передача очередных версий

По сравнению с ГОСТ 34.601 добавлены следующие стадии:

– пилот-проект;

– интеграция и тестирование;

– тиражирование.

Лекция 3

Этапы жизненного цикла автоматизированных систем и программного обеспечения

3.1 Взаимосвязь этапов ЖЦПО и ЖЦ системы

Любая АС включает в себя общесистемное программное обеспечение (ПО) и прикладное. Общесистемное ПО – покупное, поэтому при рассмотрении жизненного цикла (ЖЦ) обычно связывают ЖЦ системы с ЖЦ прикладного ПО.

Продолжительность жизненного цикла автоматизированных информационных систем, как правило, превышает продолжительность жизненных циклов общесистемных программных средств. Поэтому жизненный цикл ИС тесно связан с жизненным циклом прикладного программного обеспечения.

Основным нормативным документом, регламентирующим жизненный цикл (ЖЦ) программных средств в информационных системах, в настоящее время является ГОСТ Р ИСО/МЭК 12207. Он определяет структуру процессов жизненного цикла программного обеспечения, общий процесс создания, сопровождения и развития прикладного программного обеспечения в системе.

При проектировании железнодорожных систем рекомендуют для определения ЖЦПО применять совместно ГОСТ Р ИСО/МЭК 12207 и ОРММ ИСЖТ 5.03-00.

Основные этапы ЖЦПО были определены в ГОСТ 28195-89. При разработке системы они совмещаются с основными этапами ЖЦАС (в скобках даны номера этапов ЖЦАС по ГОСТ 34.601, с которым совмещается этап ЖЦПО).

Этапы ЖЦПО:

1) анализ требований – совмещается с формированиями требований, разработкой концепции АС и техническим заданием (1, 2, 3);
2) проектирование – совмещается с ЭП, ТП (4, 5);
3) кодирование (программирование) и тестирование – совмещается с РД (6);
4) внедрение – совмещается с вводом в действие (7);
5) сопровождение – совмещается с сопровождением (8).

Порядок следования этапов ЖЦПО также подчиняется определенным моделям ЖЦ. Используемые реально модели жизненного цикла программных средств в последнее время постоянно меняются, что связано с внедрением объектных технологий. Поэтому стандарт ГОСТ 19.102 серии ЕСПД для создания ПО систем железнодорожного транспорта считается неактуальным.

Сейчас наибольшее внимание уделяют не непосредственному созданию программных средств, а системному анализу и проектированию, т. е. началь-

ным этапам разработки, и все больше возрастает роль поддержки графической визуализации этих этапов проектирования.

3.2 Назначение этапов ЖЦ

В отличие от ЖЦПО в ЖЦАС принято формирование требований и разработку концепции называть этапом стратегии.

Стратегия – это детальное обследование условий проектирования и задач системы. Основная цель – оценка реального объема проектирования, целей и задач проекта. На этом этапе определяется, нужно ли продолжать проект дальше. Информация о системе может быть получена путем бесед, семинаров с руководством, пользователями. Список требований к системе должен включать следующие: совокупность условий, в которых предполагается эксплуатировать будущую систему (аппаратные и программные ресурсы, предоставляемые системе, внешние условия ее функционирования, состав персонала, требования к входной информации); описание функций, выполняемых системой; ограничения в процессе разработки (сроки завершения этапов, имеющиеся ресурсы, мероприятия по защите информации). Под функцией понимается совокупность действий ИС, направленная на достижение определенной цели.

Этап «Анализ требований» – это преобразование требований к системе в более точные определения. На данном этапе дается ответ на вопрос: «Что должна делать будущая система?».

Вся информация, собранная на этапе стратегии. На этом этапе требования заказчика уточняются и документируются.

Проводится подробное исследование функций системы, определенных на этапе выбора стратегии, и информации, необходимой для их выполнения. Осуществляется выбор средств разработки системы.

Уточняются предварительно принятые решения.

Результат этапа – модели будущей системы, описывающие ее с различных сторон. В их числе:

- функциональная модель или иерархия функций, которая разбивает процесс разработки на составные части (что выполняется, из чего состоит);
- информационная модель или модель данных (инфологическая).

Следующий этап «Проектирование» дает ответ на вопрос: «Каким образом система будет удовлетворять предъявленным к ней требованиям?». Проектирование в данном аспекте определяется как «Процесс получения логической модели системы вместе со строго сформулированными целями, поставленными перед нею, а также написания спецификаций физической системы, удовлетворяющей этим требованиям».

Конечный продукт этого этапа:

- схема базы данных (строится на базе инфологической модели);
- набор спецификаций программных модулей системы (строится на базе функциональной модели).

Этап «Кодирование» заключается в написании программных кодов по спецификациям программных модулей.

Этап «Тестирование» заключается в проверке и отладке функционирования системы.

Для систем архитектуры клиент/сервер, при разработке которых используются средства конфигурационного управления (СКУ) и соответствующие средства автоматизированной разработки, могут осуществляться следующие виды тестирования:

- *функциональное тестирование* – базовое тестирование, состоит в проверке (контроле) функциональности операций на единственном экземпляре приложения архитектуры клиент/сервера;
- *конфигурационное тестирование* – тестирование приложения, которое охватывает клиентскую и серверную части приложения во всех комбинациях между платформами клиента и сервера;
- *конкурентное тестирование* – тестирование двух и более клиентов, использующих один сервер. Это тестирование – разновидность функционального, когда проверяется способность сервера одновременно обслуживать двух и более клиентов.
- *стрессовое тестирование* – тестирование на большом числе транзакций и позволяет обнаружить ошибки в исходной реализации в динамике, выполняя одну или множество команд продолжительное время;
- *тестирование загрузки* – тестирование для верифицирования при работе большого числа конкурирующих клиентов и предназначено для обнаружения взаимных блокировок и проблем с очередями;
- *тестирование эффективности спроектированной системы* – тестирование для получения количественных характеристик системы с учетом реального окружения системы (например, 500 пользователей на один сервер базы данных);
- *регрессионное тестирование* – тестирование степени независимости и сохранения работоспособности той части системы, которая не подверглась модификации.

Этап «Внедрение» заключается в том, что система вводится поэтапно в три фазы: первоначальная загрузка информации; накопление информации; выход на проектную мощность.

Этап «Эксплуатация и сопровождение» выполняется организациями-разработчиками и содержит следующий перечень работ:

1. Корректировку программного обеспечения и информационного обеспечения в соответствии с требованиями заказчика (например, в связи с изменениями в тарифной политике, реорганизации в управлении перевозочным процессом, внедрением новых информационных технологий и пр.), требующую не более 10% изменений программного обеспечения.

2. Проверку и тестирование программного обеспечения после внесенных изменений.

3. Установку на объектах внедрения новых версий действующего программного обеспечения.

4. Адаптацию действующего программного обеспечения к работе на новых технических средствах и в условиях новых информационных технологий при незначительных изменениях в программах и в технологии работы.

5. Актуализацию нормативно-справочной информации.

6. Устранение ошибок или сбоев в действующем программном обеспечении.

7. Анализ функционирования разработанного программного обеспечения с целью улучшения эксплуатационных характеристик.

8. Обучение пользователей работе с новыми версиями программного обеспечения.

9. Внесение изменений в документацию в соответствии с требованиями ГОСТ.

10. Проведение консультаций пользователей по всем вопросам функционирования системы.

11. Контроль за своевременным подключением на объектах внедрения передаваемых изменений в программном и информационном обеспечении эксплуатируемых систем (комплексов задач, АРМов), осуществляемый посредством общесистемных или прикладных инструментальных средств.

3.3 Модели ЖЦ

ЖЦ образуется в соответствии с принципом нисходящего проектирования и, как правило, носит итерационный характер: реализованные этапы могут повторяться в соответствии с изменениями требований и внешних условий. На каждом этапе порождается определенный набор документов и технических решений. Существующие модели ЖЦ определяют порядок исполнения перечисленных этапов, а также критерии перехода к последующему этапу.

Наиболее распространены следующие модели ЖЦ:

1. Каскадная (70–80 гг.).

2. Поэтапная (80–85 гг.).

3. Спиральная (86–90 гг.).

Каскадная модель (рис. 2) предполагает переход к следующему этапу после полного окончания работ на предыдущем («водопад»).

Положительные стороны применения каскадного подхода заключаются в следующем: на каждом этапе формируется законченный набор проектной документации, отвечающий критериям полноты и согласованности; выполняемые в логичной последовательности этапы работ позволяют планировать сроки завершения всех работ и соответствующие затраты.

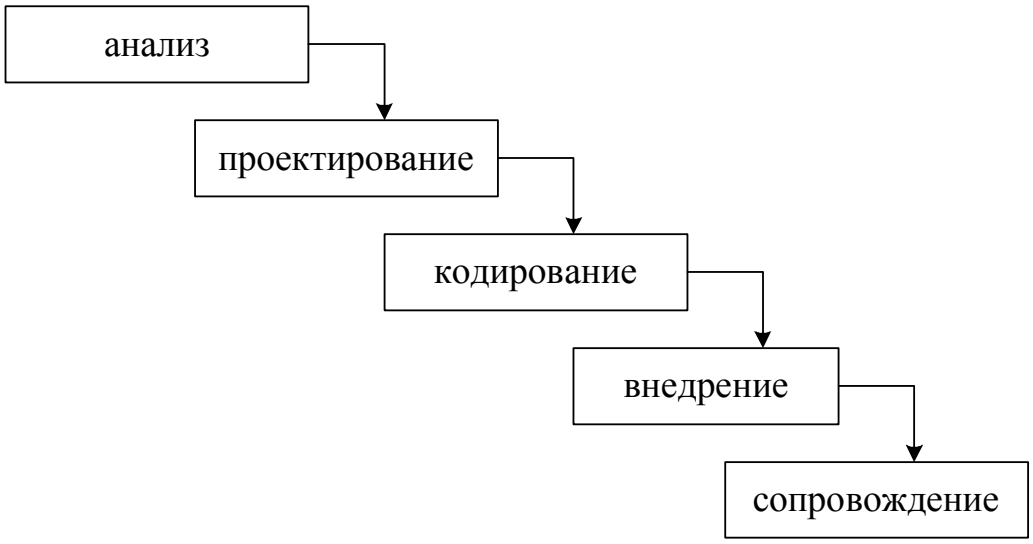


Рис. 2. Каскадная модель разработки ПО

Поэтапная модель (рис. 3) – это итерационная модель разработки с циклами обратной связи между этапами. Межэтапные корректировки менее трудоемки, но этапы более растянуты, что является основным недостатком.

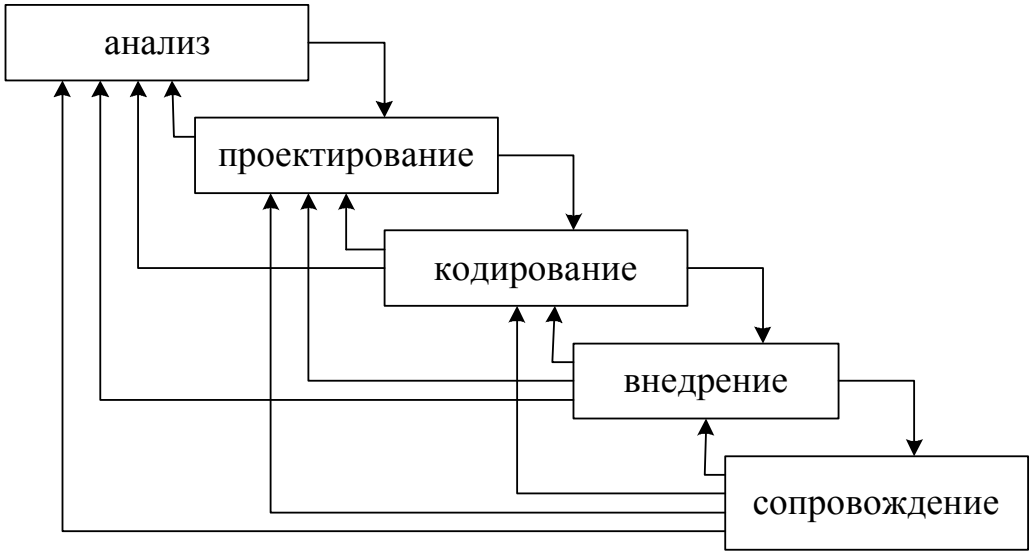


Рис. 3. Каскадная модель разработки ПО

Спиральная модель (рис. 4) делает упор на начальные этапы ЖЦ: анализ требований и проектирование. На этих этапах проверяется и обосновывается реализуемость технических решений путем создания прототипов. Каждый виток спирали соответствует поэтапной модели создания фрагмента или версии системы или программного продукта. На каждом витке определяются качество проекта и уточняются его цели, а также планируется дальнейшая работа. Неполное завершение работ на каждом этапе позволяет переходить на следующий этап, не дожидаясь полного завершения работы на текущем. При итеративном способе разработки недостающую работу можно

будет выполнить на следующей итерации. Главная задача – как можно быстрее показать пользователям системы работоспособный продукт, тем самым, активизируя процесс уточнения и дополнения требований. Лучший вариант доводится до реализации.

Преимущества спиральной модели:

- накопление версий и повторное использование ПС, моделей, прототипов;
- ориентация на развитие и модификацию ПО или системы в процессе их проектирования;
- анализ риска и издержек в процессе проектирования.

Основная проблема спирального цикла – определение момента перехода на следующий этап. Для ее решения необходимо ввести временные ограничения на каждый из этапов жизненного цикла. Переход осуществляется в соответствии с планом, даже если не вся запланированная работа закончена. План составляется на основе статистических данных, полученных в предыдущих проектах, и личного опыта разработчиков.

Рис. 4. Спиральная модель ЖЦ

В случае, когда создание ПО является частью информационной системы, необходимо рассматривать это как часть процесса создания информационной системы. Поэтому начальные этапы проектирования и заключительные этапы испытаний и сдачи заказчику должны объединять как анализ, так и интеграцию программных, информационных и технических средств всей системы, полностью решающей требуемые функциональные задачи.

ГОСТ Р ИСО/МЭК 12207 не предписывает конкретную модель ЖЦ и методы разработки программного обеспечения. Он является общим для любых моделей и методологий разработки ПО.

Разработчик проекта сам выбирает, какую модель он будет использовать при проектировании ПО. Процесс адаптации является процессом исключения процессов, видов деятельности и задач, не применимых в конкретном проекте.

Лекция 4

Основные принципы разработки документации на АС

4.1 Комплектование документации

Документация может разрабатываться как на систему в целом, так и на отдельные ее части. Рекомендуют пользоваться следующими системами ГОСТ для разработки компонентов систем (таблица 2):

Таблица 2

Наименование компонента	Система ГОСТ	Пример компонента
Программные средства	ЕСПД	Расчетная программа
Информационные средства	КС АС, ЕСКД	
Программно-технические комплексы	КС АС, ЕСПД	Система удаленной диагностики и мониторинга
АРМ в АС	КС АС, ЕСПД	АРМ оператора техконторы
АРМ автономные	ЕСПД	АРМ экономиста
Аппаратно-программные комплексы	ЕСКД, ЕСПД, КС АС	Интеллектуальные модемы, узлы коммутации, мультиплексоры, контроллеры, драйверы специализированного ввода информации
Базы и банки данных	КС АС, ЕСПД	Электронные картотеки
Технические средства в АС	ЕСКД, КС АС, ЕСПД	Модуль ввода дискретной информации

Допускается выпуск документации по международным стандартам при использовании специальных средств автоматизированного проектирования.

Документация на систему состоит:

- из документации разработки;
- из эксплуатационной документации.

Документация разработки в свою очередь включает в себя:

- общесистемные решения (ОР);
- решения по организационному обеспечению (ОО);
- решения по техническому обеспечению (ТО);
- решения по математическому обеспечению (МО);

– решения по программному обеспечению (ПО);
– решения по информационному обеспечению (ИО).

4.2 Техническое задание

Разработка любой системы начинается с разработки технического задания (ТЗ) согласно этапам жизненного цикла.

ТЗ является центральным моментом при создании АС и регламентирует все требования (технические, функциональные) и технические, организационные и программные решения при ее проектировании.

Техническое задание – основной документ, определяющий требования и порядок создания (развития или модернизации) автоматизированной системы, в соответствии с которым проводится разработка АС и ее приемка при вводе в действие.

Как правило, ТЗ разрабатывают на систему в целом, хотя допускается разрабатывать ТЗ на отдельные части (модули) системы, если она уже существует, и планируется внедрение ее развития и модернизации. ТЗ разрабатывается на основании исходных данных об объекте автоматизации по результатам проведенного, например, акта обследовательских работ.

ТЗ разрабатывают на:
– подсистемы АС;
– комплексы задач АС;
– базу данных;
– на комплектующие средства технического обеспечения и программно-технические комплексы по ЕСКД;
– на программные средства по стандартам ЕСПД.

ТЗ согласно ГОСТ 34.602 на АС содержит следующие основные разделы:

1. Общие сведения.
2. Назначение и цели создания системы.
3. Характеристика объектов автоматизации.
4. Требования к системе.
5. Состав и содержание работ по созданию системы.
6. Порядок контроля и приемки системы.
7. Требования к составу и содержанию работ по подготовке объекта автоматизации и вводу системы в действие.
8. Требования к документированию.
9. Источники разработки.

Как правило, по разделу «Общие сведения» требуется предоставить следующие сведения:
– полное наименование системы и ее условное обозначение;
– наименование предприятий заказчика и разработчика;
– перечень документов, на основании которых создается система;
– плановые сроки начала и окончания выполнения работ.

Раздел «Назначение и цели создания системы» – один из основополагающих. Должен описываться всегда, иначе отпадает весь смысл в создании ИС. Как правило, в разделе «Назначение» приводится название процесса, который автоматизируется, для автоматизации которого предназначается данная система и области ее применения, или объекты, где предполагается ее использование. Она независима или создается как часть другой системы.

Цель создания системы – это всегда достижение каких-либо положительных эффектов (экономического, повышение эффективности выполнения процессов, повышение безопасности). В этом же разделе можно привести перечень основных задач, которые нужно выполнить для успешного создания системы.

Пример:

«Цель разработки: повышение эффективности обслуживания устройств АиТ. Для достижения поставленной цели необходимо решить следующие задачи:

- автоматизация выполнения работ по техническому обслуживанию, связанных с аналоговыми измерениями (напряжения и фазы на путевых реле, изоляция, замедления сигнальных реле и т. п.);
- ускорение поиска отказов за счет непрерывной записи информации о технологической ситуации на станции (дискретный контроль состояния основных реле исполнительной и наборной группы);
- выявление предотказного состояния на основе экспертных оценок функциональных зависимостей между измеряемыми величинами и вероятностью отказа».

В разделе «Характеристика объекта автоматизации» приводят описание объекта автоматизации или автоматизируемого процесса. Все описывается так, как есть на текущий момент, если есть какие-то инструкции, в соответствии с которыми выполняется данный процесс или работает объект автоматизации. Приводятся сведения об условиях эксплуатации объекта. В этот же раздел можно включить схему информационных потоков, связанных с данной системой, организационную структуру.

Все основные технические, организационные и программные требования отражаются в разделе «Требования к системе». При установке требований к системе рекомендуется принимать во внимание указания ГОСТ 24.104-85.

Раздел «Требования к системе» состоит из 3 подразделов:

1. Требования к системе в целом.
2. Требования к функциям (задачам), выполняемым системой.
3. Требования к видам обеспечения.

Состав требований к системе (1 подраздел) может меняться в зависимости от вида конкретной системы.

Требования к системе в целом: требования к режимам функционирования, к надежности, к безопасности, к персоналу, к физической структуре (в т. ч. принципы построения: одномодульный, многомодульный). Если в состав

системы входят несколько подсистем, обеспечивающих реализацию ряда определенных функций, необходимо перечислить их с указанием перечня решаемых подсистемами задач. Эти подсистемы могут быть реализованы в виде АРМ или самостоятельных программных модулей.

Пример:

Административная подсистема, подсистема обмена информацией, подсистема ведения баз данных, технологическая подсистема и т. д.

Требования к функциям (задачам): по каждой подсистеме перечень функций задач или их комплексов, временной регламент выполнения каждой функции, требования к качеству реализации каждой функции, к форме представления выходной и входной информации, достоверность результатов.

Требования по видам обеспечения: требования по математическому, лингвистическому, программному, техническому, метрологическому, организационному и т. д. видам обеспечения.

В требованиях к информационному обеспечению содержатся следующие разделы:

- состав, структура и способы организации данных в системе (информационно-логическая схема);
- информационный обмен между компонентами системы;
- информационная совместимость со смежными системами;
- унификация и стандартизация системы;
- к структуре процесса сбора, обработки и передачи данных в системе и представлению данных;
- к защите данных в системе;
- к контролю, хранению и восстановлению данных.

В требованиях к программному обеспечению руководствуются ГОСТ 24.104-85.

Разработка программного обеспечения регламентирована комплексом стандартов по программной документации «Единая система программной документации (ЕСПД)». ПО должно быть выполнено в объеме, достаточном для надежного функционирования всех подсистем и устойчивой связи между ними.

В требованиях к аппаратному обеспечению указывают характеристики аппаратных средств и вычислительной техники, которые будут использоваться в системе (покупные) или создаваться в рамках проекта.

Метрологическое обеспечение целесообразно указывать только для информационно-измерительных систем.

Лингвистическое, согласно определению, предъявляет требования к языкам программирования, применяемым для кодирования ПО.

4.3 Виды испытаний АС по ГОСТ 34.603

Испытания АС проводят на стадии «Ввода в действие» ГОСТ 34.601 с целью проверки соответствия создаваемой АС требованиям технического задания (ТЗ).

Для АС, согласно ГОСТ 34.603, устанавливают следующие основные виды испытаний:

- предварительные (автономные или комплексные);
- опытная эксплуатация;
- приемочные испытания.

Примечания:

1. Допускается дополнительно проведение других видов испытаний АС и их частей.

2. Виды испытаний и статус приемочной комиссии устанавливают в договоре и (или) ТЗ в разделе «Порядок контроля и приемки системы».

Предварительные испытания могут быть автономные или комплексные. Автономные испытания охватывают части АС. Их проводят по мере готовности частей АС к сдаче в опытную эксплуатацию. Комплексные испытания проводят для групп, взаимосвязанных частей АС или для АС в целом. Для планирования проведения всех видов испытаний разрабатывают документ «Программа и методика испытаний».

Предварительные испытания АС проводят для определения ее работоспособности и решения вопроса о возможности приемки АС в опытную эксплуатацию.

Опытную эксплуатацию АС проводят с целью определения фактических значений количественных и качественных характеристик АС и готовности персонала к работе в условиях функционирования АС, определения фактической эффективности АС, корректировке (при необходимости) документации.

Приемочные испытания АС проводят для определения соответствия АС техническому заданию и решения вопроса о возможности приемки АС в постоянную эксплуатацию.

Все испытания закрываются актами.

Лекция 5

Этап «Анализ»

5.1 Основные принципы и методы структурного анализа

Фаза анализа – основополагающая. Ошибки этой фазы имеют самые тяжелые последствия.

Рассмотрим подробнее фазу анализа как наиболее важную во всем процессе создания ИС. Анализ состоит в исследовании системных требований и проблемы, а не в поисках путей их решения. Результат анализа выражается в

модели предметной области (Domain model), которая иллюстрируется в виде набора диаграмм с изображенными на них понятиями или объектами предметной области.

Модель предметной области – это представление понятий, выраженных в терминах предметной области задачи.

Анализ это очень широкое понятие, которое в свою очередь можно поделить на две разновидности:

1) структурный системный анализ;
2) объектно-ориентированный анализ.

Структурный анализ – это исследование системы, которое начинается с ее общего обзора и затем детализируется, приобретая иерархическую структуру со все большим числом уровней. Структурный подход не обеспечивает возможность создания предельно сложных систем (неэффективен в объектно-ориентированных языках программирования (ООЯП))

Объектно-ориентированный анализ основан на объектах и их взаимодействии между собой. Достоинство – объектно-ориентированные системы более гибкие и лучше развиваются.

Одновременно двумя способами проектировать систему нельзя.

Для проведения структурного анализа принято использовать методологии, базирующиеся на наборе основополагающих методов, так называемых структурных.

Методология – это совокупность методов, применяемых в ЖЦ системы. Методология включает руководящие указания для оценки и выбора проекта, определяет шаги работы и последовательность применения методов для достижения поставленной цели.

Метод – это последовательный процесс создания моделей, которые описывают вполне определенными средствами различные стороны разрабатываемой программной системы. Появились в 60–70 годы.

Методы, в свою очередь, созданы на основе принципов.

Структурные методы являются строгой дисциплиной системного анализа и проектирования. Методы структурного анализа и проектирования направлены на упрощение сложных систем. Общий подход к проведению анализа представлен на рис. 5

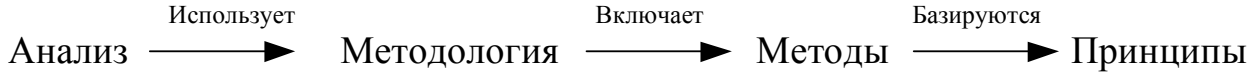


Рис. 5. Структурный анализ

В основе методов структурного анализа лежат следующие основные 3 принципа:

1. Разбиение системы на черные ящики (принцип «разделяй и властвуй»).

Разбиение должно удовлетворять следующим критериям:

– каждый черный ящик должен реализовывать одну единственную функцию системы;

– функция каждого черного ящика должна быть легко понимаема независимо от его сложности;

– связь между черными ящиками должна вводиться только при наличии связи между существующими реально функциями системы;

– связи между черными ящиками должны быть по возможности простыми, для обеспечения независимости между ними.

2. Идея иерархии (принцип иерархического упорядочивания).

Помимо разбиения системы на части их необходимо определенным образом упорядочить. В виде иерархических структур.

3. Структурные методы широко используют графические нотации, служащие для облегчения понимания сложных систем.

Нотация – это средство описания структуры системы, данных и этапов обработки в виде диаграмм, графов, блок-схем, таблиц, языков.

Помимо двух основных принципов в структурном анализе используется еще и ряд второстепенных, но не менее важных. К ним относятся:

1) принцип абстрагирования – выделение существенных с некоторых позиций аспектов системы и отвлечение от несущественных для простоты представления системы;
2) принцип формализации – необходимость строго методического подхода к решению проблемы;
3) принцип упрятывания – упрятывание несущественной на конкретном этапе информации;
4) принцип полноты – контроль на присутствие лишних элементов;
5) принцип непротиворечивости – обоснованность и согласованность элементов;
6) принцип логической независимости – сосредоточение внимания на логическом проектировании без привязки его к физическому;
7) принцип независимости данных – модели данных должны анализироваться и проектироваться без привязки к их логической обработке и физическому распределению;
8) принцип структурирования данных – структуризация данных.

На основе перечисленных принципов создано три метода, позволяющих построить:

– функциональную структуру (функции, которые система должна выполнять);

– информационную структуру (отношения между данными);

– алгоритм функционирования системы (зависящее от времени поведение системы).

Методы используют нотации. Методы (или техники построения структурной диаграммы функций, данных или событий) используют следующие средства моделирования системы (нотации):

– IDEF0 функциональные диаграммы;
– DFD (Data Flow Diagrams) диаграммы потоков данных совместно со словарями данных и спецификациями процессов или миниспецификациями;
– ERD (Entity-Relationship Diagrams) диаграммы «сущность-связь»;
– STD (State Transition Diagrams) – диаграммы перехода состояний.

Модели, построенные с использованием перечисленных диаграмм в их различной комбинации, в совокупности дают полное описание ИС независимо от того, является ли она существующей или вновь разрабатываемой. Состав диаграмм в каждом конкретном случае зависит от необходимой полноты описания системы.

Перечисленные методы в различной комбинации используются методологиями.

5.2 Методологии структурного анализа и проектирования

Методология структурного анализа и проектирования определяет руководящие указания для оценки и выбора проекта разрабатываемого ПО, шаги работы, которые должны быть выполнены, их последовательность, правила распределения и назначения операций и методов.

Современные структурные методологии анализа и проектирования классифицируются по следующим признакам:

1. По отношению к школам – Software Engineering (SE) и Information Engineering (IE).

2. По порядку построения модели – процедурно-ориентированные, ориентированные на данные, информационно-ориентированные.

3. По типу целевых систем – для систем реального времени (СРВ) и для информационных систем (ИС).

Поясним вышесказанное. SE является универсальной дисциплиной разработки программного обеспечения ПО для информационных систем._Полученное в результате ПО будет представлять собой иерархическую, структурированную, модульную программу. SE применяется и для информационных систем и для систем РВ.

IE является дисциплиной построения систем вообще, а не только ПО информационной системы, и включает этапы более высокого уровня (например, стратегическое планирование). На этапе проектирования систем ПО эти дисциплины аналогичны.

Разработка любого ПО системы основана на модели ВХОД–ОБРАБОТКА–ВЫХОД (данные входят в систему, обрабатываются или преобразуются и выходят из системы). Такая модель используется во всех структурных методологиях. Процедурно-ориентированный подход регламентирует первичность проектирования функциональных компонент по отношению к проектированию структур данных (ОБРАБОТКА первична). При подходе, ориентированном на данные, ВХОД–ВЫХОД являются наиболее важными элементами модели. Структуры данных определяются первыми,

а процедурные компоненты являются производными от данных. Информационно-ориентированный подход, как часть IE-дисциплины, отличается от подхода, ориентированного на данные, тем, что позволяет работать с неиерархическими структурами данных.

Системы реального времени контролируют и контролируются внешними событиями. Реагирование на эти события во времени – основная функция таких систем.

Основная функция информационных систем – обработка данных.

В настоящее время успешно используются практически все известные методологии структурного анализа и проектирования, однако наибольшее распространение получили следующие:

1. SADT (Structured Analysis and Design Technique) (процедурно-ориентированная, ИС, IE).
2. Структурного системного анализа Гейна-Сарсона (Gane-Sarson) (процедурно-ориентированная, ИС и СРВ, SE).
3. Структурного анализа и проектирования Йодана/Де Марко (Yourdon/De Marko) (процедурно-ориентированная, ИС и СРВ, SE).
4. Информационного моделирования Мартина (Martin) (информационно-ориентированная, ИС, IE).

Лекция 6

Функциональные модели

6.1 SADT – методология структурного анализа и проектирования

Данная методология впервые была введена в 1973 г., разработана Дугласом Россом.

Методология SADT – это совокупность методов, правил и процедур, предназначенных для функционального моделирования предметной области.

На базе технологии SADT был разработан стандарт IDEF0, являющийся частью семейства IDEF, которое поддерживается министерством обороны США. В рамках данной методологии были разработаны несколько официальных стандартов анализа и проектирования. Методология SADT может использоваться для широкого круга систем. Для уже существующих систем возможно применение SADT для анализа функций, выполняемых системой, и механизмов, которые управляют этими функциями.

Полная методология заключается в построении активностной модели и модели данных (рис.6), но наиболее распространена активностная модель.

Активностная модель – это совокупность диаграмм, текста и глоссария. Диаграммы – главные компоненты модели. В моделях используется графический язык. Разработчик диаграмм – аналитик. Диаграммы могут детализироваться введением новых уровней, организуются в иерархические древовидные структуры. Каждая диаграмма нижнего уровня детализирует «внутреннее состояние» блока на родительской диаграмме.

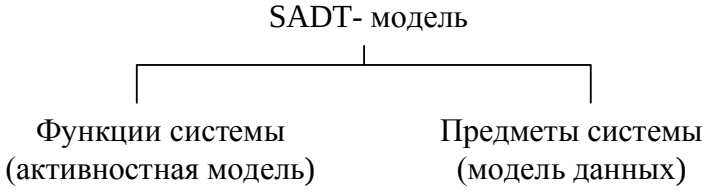


Рис. 6. Полная методология SADT

В состав диаграммы входят блоки (активности) и дуги (взаимоотношения) между блоками.

Блок – это функция моделируемой системы. Название блока – глагол или глагольный оборот. Структура блока показана на рис. 7.

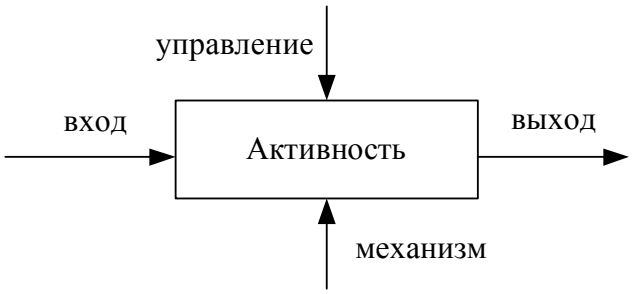


Рис. 7. Активность

Входы преобразуются в выходы, управление – ограничивает или предписывает условия выполнения (информация, управляющая активностью), исполнители – описывают, за счет чего выполняется преобразование (реализация: люди, техника).

SADT не допускает более 6 блоков на одной диаграмме (от 3 до 6). Это обеспечивает наглядность диаграмм. Размещение блоков организуется по принципу доминирования – степени важности. Нумерация блоков – последовательная, также в порядке доминирования. Номер блока – это идентификатор системной функции, позволяющий организовать ее в иерархическую модель.

Дуга изображает объекты предметной области, через которые связаны функции (рис. 8).

Рис. 8. Дуга

Дуга – это множество объектов, где объект абстрактное понятие, которое может включать, например, план производства, данные, машины, людей, документы. SADT диаграммы – это совокупность блоков, связанных дугами, которые определяют, как блоки влияют друг на друга. Поэтому такие диа-

граммы относят к предписывающим, указывающим правила преобразования вход-выход и интерфейсы между функциями системы. В SADT – пять типов взаимозависимости между блоками для описания их отношений (рис. 9).

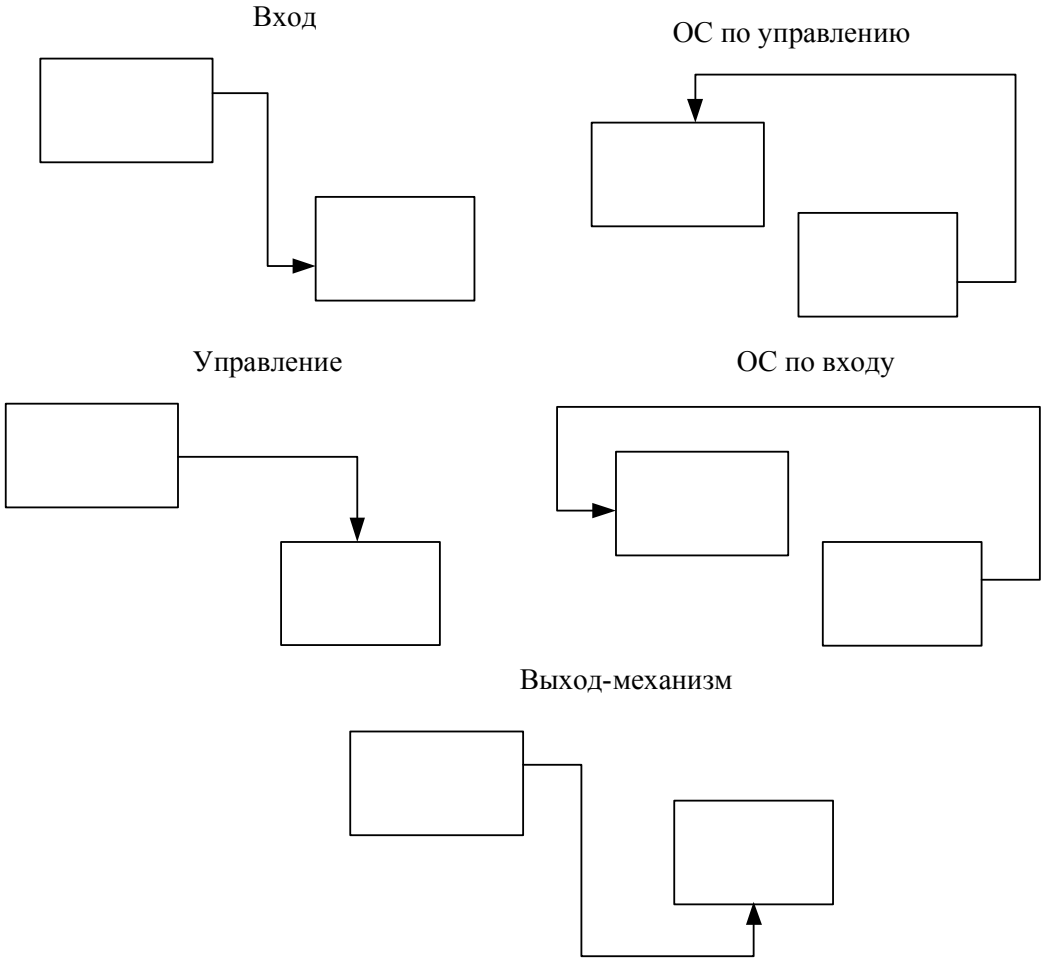


Рис. 9. Взаимозависимости между блоками

«Выход-механизм» характерны для задач планирования и распределения ресурсов.

«Управление» характерны для разработки руководящих указаний, применяемых для выполнения последующих процессов.

Поскольку дуги представляют собой наборы объектов, то у них может быть множество конечных точек, что приводит к расщеплению дуг или, наоборот, к их объединению. Поэтому были приняты соглашения:

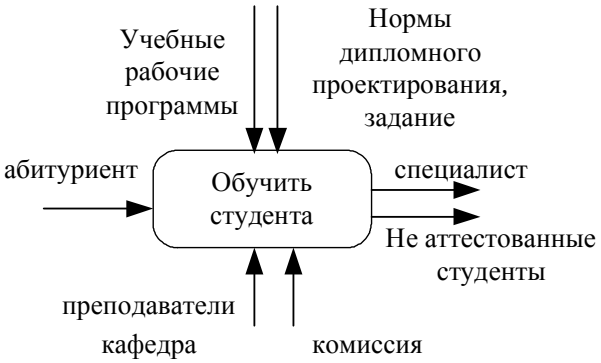
1) дуги помечаются до разветвления;
2) если ветвь не помечена – она содержит все объекты, указанные перед расщеплением;
3) помеченные ветви показывают, какую часть из родительской они содержат.

Те же правила распространяются и на слияние. Достоинство данной методологии – отражение характеристик управления, обратных связей, исполнителей. Основная область применения – для реорганизации хорошо специ-

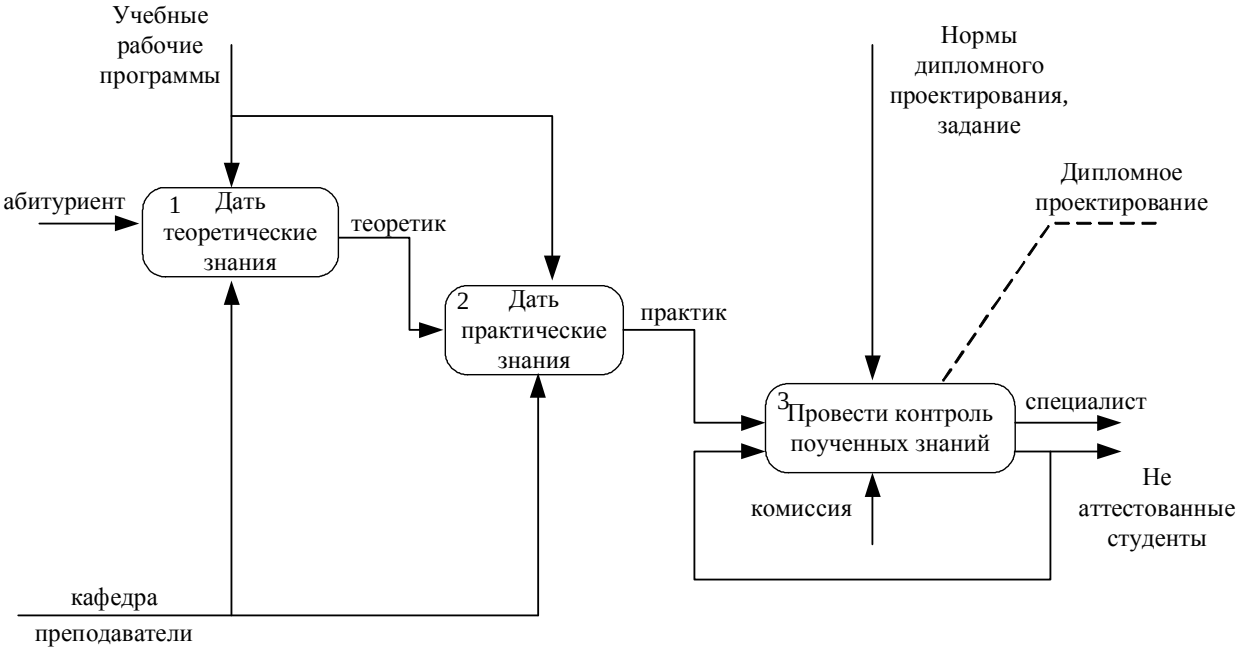
фицированных и стандартизованных процессов. Недостаток – диаграммы не выразительны и неудобны для анализа и проектирования многофункциональных информационных систем.

В РФ разработан специальный стандарт Р 50.1.28-2001 – Методология функционального моделирования IDEF0.2001

Пример диаграммы IDEF0 «Описание процесса – обучение студента» в упрощенном виде приведен на рис. 10.



Контекстная диаграмма



Детализирующая диаграмма

Рис. 10. Пример IDEFO

6.2 Моделирование потоков данных (процессов)

Диаграмма потоков данных (ДПД или DFD) – основное средство моделирования функциональных требований к системе. Главная цель DFD диаграмм – продемонстрировать, как каждый процесс системы преобразует свои входные данные в выходные, а также выявить отношения между процессами.

Идея DFD основана на следующем: источники информации (внешние сущности) порождают информационные потоки (потоки данных), переносящие информацию к подсистемам или процессам; подсистемы или процессы преобразуют информацию и порождают новые потоки, которые переносят информацию к другим процессам или подсистемам, накопителям данных или внешним сущностям – потребителям информации.

Таким образом, основными компонентами диаграмм потоков данных являются:

– внешние сущности;
– системы/подсистемы (для сложных систем);
– процессы;
– накопители данных;
– потоки данных.

Обозначения перечисленных элементов в нотации Гейна-Сарсона приведены на рис.11.

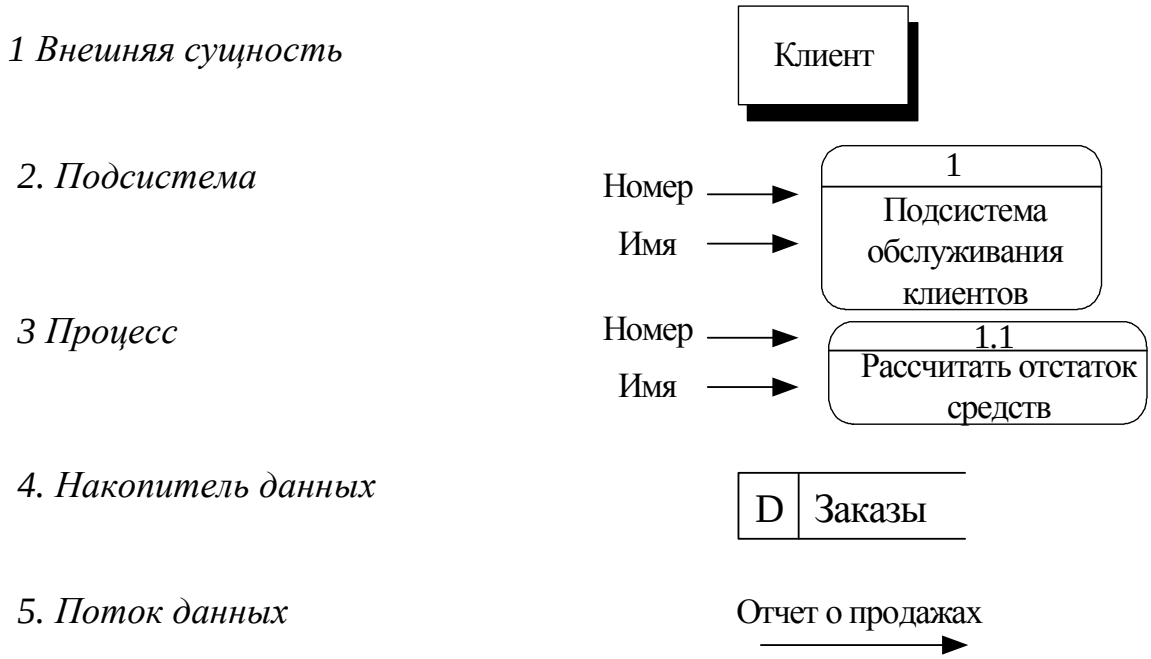


Рис.11. Нотации DFD Гейна-Сарсона

Внешняя сущность – это материальный предмет или физическое лицо, представляющее собой источник или приемник информации. Внешняя сущность всегда находится за переделами границ анализируемой ИС. Но внешние сущности всегда имеют к системе непосредственное отношение. Имя ее – всегда имя существительное. Объекты, представленные такими узлами, не должны участвовать ни в какой обработке информации.

Декомпозиция системы на подсистемы осуществляется при необходимости, если предназначение системы трудно отразить в одном основном процессе на диаграмме самого верхнего уровня. При этом каждая подсистема –

это тоже определенный процесс, выполняющий конкретную задачу в системе. Подсистемы выделяются для крупных ИС, территориально распределенных, включающих в себя несколько более мелких ИС. Номер подсистемы служит для ее идентификации. В поле имени вводится наименование подсистемы в виде предложения с подлежащим и соответствующими определениями и дополнениями.

Процесс – преобразование входных потоков данных в выходные в соответствии с действием, задаваемым именем процесса. Номер процесса служит для его идентификации. В поле имени вводится наименование процесса в виде предложения с глаголом в неопределенной форме (вычислить, рассчитать, проверить, определить, создать, получить), за которым следуют существительные в винительном падеже.

Например:

«Ввести …», «Выдать информацию ……..», «Составить ……..».

Физически процесс может быть реализован различными способами (отдел организации, программа, устройство, выполняющее заданную функцию).

Накопитель данных – абстрактное устройство для хранения информации, которую можно в любой момент поместить в накопитель и через некоторое время извлечь, причем способы помещения и извлечения могут быть любыми.

Накопитель данных может быть реализован физически в виде микрофиши, ящика в картотеке, таблицы в оперативной памяти, файла на магнитном носителе.

Накопитель данных идентифицируется буквой «D» или «БД» и произвольным числом. Имя накопителя должно отображать его содержание. Имя хранилища – имя существительное.

Накопитель данных в общем случае является прообразом будущей базы данных и описание хранящихся в нем данных должно быть увязано с информационной моделью.

Поток данных определяет информацию, передаваемую через некоторое соединение от источника к приемнику. Реальный поток данных может быть информацией, передаваемой по кабелю между двумя устройствами, пересылаемыми по почте письмами, магнитными лентами или дискетами, переносимыми с одного компьютера на другой.

Поток данных на диаграмме изображается стрелкой, которая показывает направление потока. Каждый поток данных имеет имя, отражающее его содержание.

Лекция 7

Функциональные модели

7.1 Построение модели с помощью ДПД

Главная цель построения модели – создание ясных и понятных требований к системе на каждом уровне детализации. Декомпозиция DFD осуществляется на основе процессов, каждый процесс раскрывается на основе DFD нижнего уровня.

Для этого рекомендуется следующее:

- на каждой диаграмме размещать от 3 до 7 процессов;
- не использовать аббревиатуры;
- декомпозицию поток данных проводить совместно с декомпозицией процессов.

Процесс построения модели сводится к следующему:

1. Множество требований к системе разбивается на основные функциональные группы.
2. Идентифицируются внешние объекты.
3. Идентифицируются основные потоки информации между внешними объектами и системой.
4. Разрабатывается предварительная контекстная диаграмма, на которой основные функциональные группы представлены процессами, внешние объекты – внешними сущностями, основные потоки информации – потоками данных.

Контекстная диаграмма КД – моделирует систему наиболее общим видом, отражает интерфейс системы с внешними сущностями, т. е. информационные потоки, которыми она связывается с внешним миром. На контекстной диаграмме отображается, как правило, единственный основной процесс в системе, отражающий ее основную задачу, и внешние сущности.

Каждый проект имеет одну контекстную диаграмму со звездообразной топологией.

Номер единственному основному процессу на контекстной диаграмме первого уровня, как правило, не присваивается.

В центре КД находится так называемый главный процесс, соединенный с приемниками и источниками информации, посредством которых с системой взаимодействуют пользователи и другие внешние системы.

Для сложной ИС строится контекстная диаграмма, которая включает в себя подсистемы, соединенные между собой и внешними сущностями информационными потоками.

Признаками сложности ИС могут быть:

- наличие большого количества внешних сущностей (десять и более);
- распределенная природа системы;
- многофункциональность системы с группировкой функций в отдельные подсистемы.

Для каждой подсистемы, присутствующей на контекстных диаграммах, либо же для одного главного процесса для простой ИС выполняется ее детализация при помощи DFD. Каждый процесс на DFD, в свою очередь, может

быть детализирован при помощи DFD или миниспецификации – когда полученный процесс не имеет смысла дальше детализировать. При детализации должны выполняться следующие правила:

– *правило балансировки* – означает, что при детализации подсистемы или процесса детализирующая диаграмма в качестве внешних источников/приемников данных может иметь только те компоненты (подсистемы, процессы, внешние сущности, накопители данных), с которыми имеет информационную связь детализируемая подсистема или процесс на родительской диаграмме;

– *правило нумерации* – означает, что при детализации процессов должна поддерживаться их иерархическая нумерация. Например, процессы, детализирующие процесс с номером 12, получают номера 12.1, 12.2, 12.3 и т. д.

Миниспецификация (спецификация процесса) – описание логики процесса, конечная ветка иерархии DFD, по которой можно разработать соответствующую программу или реализацию. Решение о завершении детализации процесса и использовании миниспецификации принимается аналитиком исходя из следующих критериев:

– возможности описания преобразования данных процессом в виде последовательного алгоритма;

– выполнения процессом единственной логической функции преобразования входной информации в выходную;

– возможности описания логики процесса при помощи миниспецификации небольшого объема (не более 20–30 строк).

После построения законченной модели системы ее необходимо верифицировать (проверить на полноту и согласованность).

В согласованной модели для всех потоков данных и накопителей данных должно выполняться правило сохранения информации: все поступающие куда-либо данные должны быть считаны, а все считываемые данные должны быть кем-либо получены или куда-либо записаны.

Пример построения модели для АИС пропуска автотранспорта на проходной (рис. 12)

Транспорт идентифицируется по номеру. Номер имеет цифровой и буквенный код. Система находится в ожидании ввода данных. При подъезде автотранспорта система должна идентифицировать номер, при соответствии его имеющемуся в базе – открыть шлагбаум.

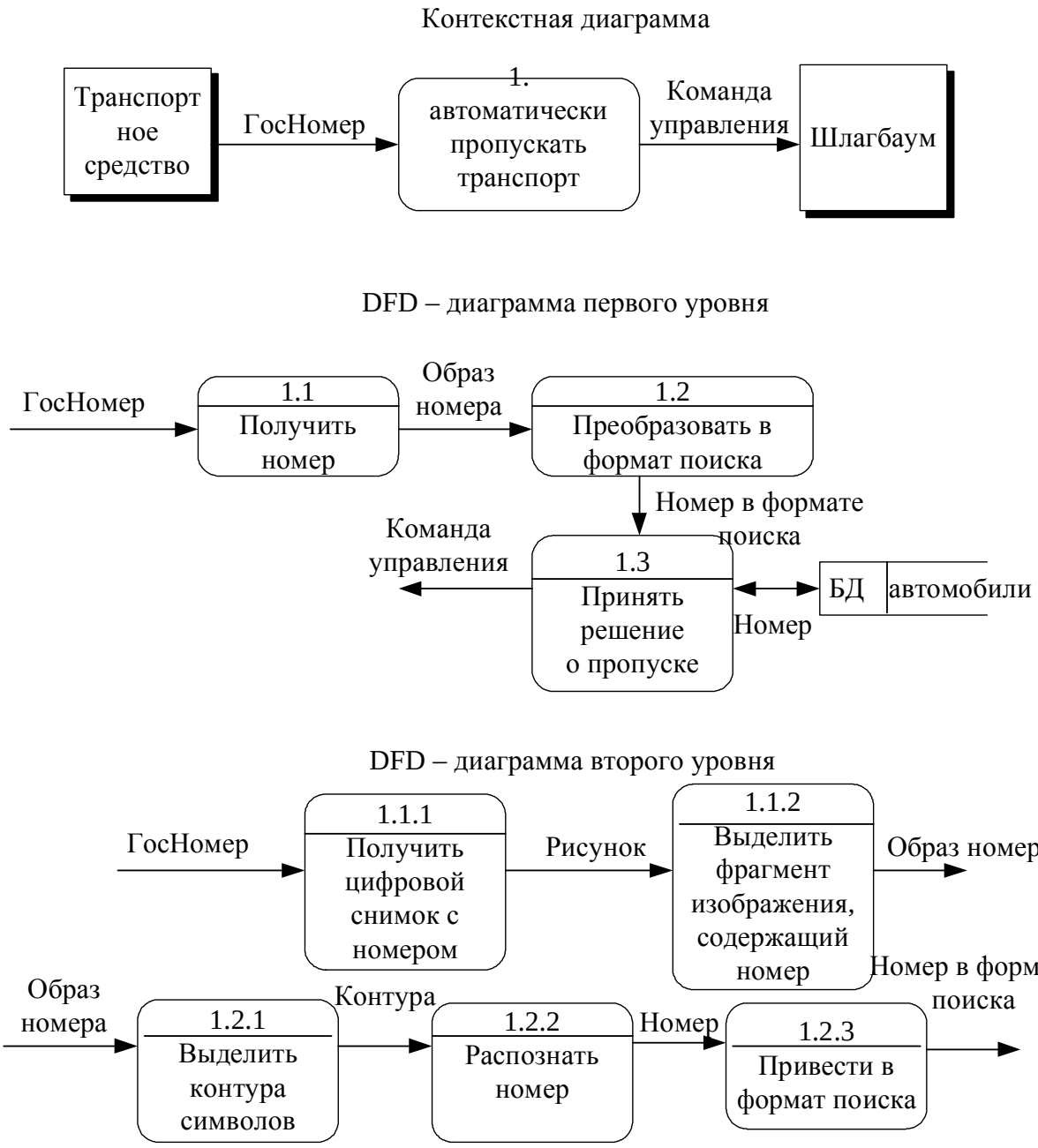


Рис. 12. Пример построения функциональной модели (DFD Гейна-Сарсона)

7.2 Словарь данных

Недостатком диаграмм данных является то, что они не всегда показывают, какая информация преобразуется процессами и как она изменяется. Задача описания информации, преобразуемой процессами, реализуется в словаре данных.

Словарь данных – это определенным образом организованный список всех элементов данных системы с их точными определениями. Это дает всем проектировщикам возможность иметь представление о входных и выходных потоках и хранилищах данных.

Словарь данных – «метабаза», в которой хранится информация о базе данных системы. Его основная задача – документирование данных, а также

однозначное толкование элементов данных, управление данными на всех этапах разработки и эффективное взаимодействие между разработчиками.

Словарь хранит описания потоков данных в системе. Потоки данных на диаграммах могут иметь разные структуры, поэтому на DFD для работы с потоками данных имеются расширения диаграммы потоком данных.

Структуры потоков данных, включающих в себя несколько более простых потоков, описываются с помощью формы Бэкуса-Наура (БНФ) в словаре данных. Этот прием применяется для читабельности диаграмм.

Пример: поток ФИО может объединять в себе фамилию, имя, отчество

Для каждого потока в словаре данных надо хранить его имя, его тип и его атрибут. Перед каждым элементом ставится @.

Типы потока:

1. Простой или групповой.
2. Внутренний или внешний.
3. Поток данных или поток управления.
4. Непрерывный или дискретный.

Атрибуты потока:

1. БНФ – определение.
2. Единицы измерения.

Наиболее часто используется групповой поток, описываемый БНФ-нотацией в словаре данных.

Синтаксис: @БНФ = /простой оператор/ или БНФ – выражение

простой оператор – это текстовое выражение, заключенное в «/».

БНФ - выражение – выражение в форме Бекуса-Наура, допускающее следующие операции отношений:

– = – означает «композиция из»;

– + – означает «*И*»;

– [!] – означает «*ИЛИ*»;

– ( ) – означает, что компонент в скобках не обязателен;

– { } – означает итерацию компонента в скобках;

– «» – означает литерал.

Пример:

@ ИМЯ = ГОСНОМЕР

@ ТИП = групповой поток

@ БНФ = буквенный код + цифровой код + код региона

@ ИМЯ = КОМАНДА ШЛАГБАУМУ

@ ТИП = дискретный поток

@ БНФ = [«0» ! «1»]

Ниже также приведены различные примеры описания данных:

@ ИМЯ = ЗАЯВЛЕНИЕ О ПРИЕМЕ

@ ТИП = управляющий поток

@ БНФ = /указывает на готовность человека стать сотрудником и предоставить личные данные/

@ ИМЯ = ФОРМА ВЫВОДА ДАННЫХ
@ ТИП = дискретный поток
@ БНФ = /указывает форму вывода найденных по ЗАПРОСУ данных/

Лекция 8

Методы задания спецификация процессов

Если нет необходимости в дальнейшей детализации процесса с помощью DFD, то используют спецификацию процесса. *Спецификация процесса* – это алгоритм описания задачи, выполняемой процессом. Множество всех СП представляет собой спецификацию системы.

Классификация методов задания СП

1. Текстовое описание.
2. Структурированный естественный язык.
3. Таблица решений.
4. Дерево решений.
5. Визуальный язык.
6. Язык программирования.

Независимо от метода задания процесса его описание всегда имеет стандартное начало:

@ВХОД = <имя_данных1>
@ВЫХОД = <имя_данных2>
(или @ВХОДВЫХОД = <имя символа данных> если входной и выходной символы одинаковы)
<имя _ данных n> – соответствующее имя, определенное в словаре данных
@СПЕЦПРОЦ
<тело спецификации>
@КОНЕЦСПЕЦПРОЦ

Спецификации должны удовлетворять следующим требованиям:

- для каждого процесса нижнего уровня должна существовать только одна спецификация;
- спецификация должна определять способ преобразования входных потоков в выходные;
- на данном этапе не нужно определять метод реализации этого преобразования;

– набор конструкций для построения спецификации должен быть стандартным.

Набор конструкций является типовым для всех языков и включает в себя следующие типовые элементы: последовательную обработку, условие, CASE-выбор и три вида циклов.

8.1 Структурированный естественный язык

В состав языка входят глаголы, предлоги и союзы, используемые в логических отношениях, математические, физические и технические термины, арифметические уравнения, таблицы, диаграммы, графы.

Управляющие структуры языка имеют один вход и один выход. К ним относятся:

1. Последовательная конструкция.

ВЫПОЛНИТЬ функция 1
ВЫПОЛНИТЬ функция 2
ВЫПОЛНИТЬ функция 3

2. Конструкция выбора.

ЕСЛИ <условие> ТО
ВЫПОЛНИТЬ функция 1
ИНАЧЕ
ВЫПОЛНИТЬ функция 2
КОНЕЦЕСЛИ

3. Итерация.

ДЛЯ <условие>
ВЫПОЛНИТЬ функция 1
КОНЕЦДЛЯ
или
ПОКА <условие>
ВЫПОЛНИТЬ функция 1
КОНЕЦПОКА

При использовании структурированного естественного языка приняты следующие соглашения:

1. Логика процесса выражается в виде комбинации последовательных конструкций, конструкций выбора и итераций.
2. Ключевые слова должны быть написаны заглавными буквами.
3. Слова или фразы, определенные в словаре данных, должны быть написаны заглавными буквами.
4. Глаголы должны быть активными и ориентированными на целевое действие (заполнить, вычислить, извлечь, а не модернизировать, обработать).

Пример составления спецификации процесса на примере нашей АИС:

Спецификация процесса 1.3
@ВХОД = Распознанный код
@ВХОД = Номер
@ВЫХОД = Команда открытия
@СПЕЦПРОЦ 1.3 Принять решение о пропуске
ПОКА не перебраны все номера в БД Номера
ВЫПОЛНИТЬ получить Номер
ЕСЛИ Номер = Распознанный код ТО
ВЫПОЛНИТЬ выдать Команда открытия
КОНЕЦЕСЛИ
КОНЕЦПОКА
@ КОНЕЦСПЕЦПРОЦ 1.3

8.2 Таблицы и деревья решений

Структурированный естественный язык неприемлем для некоторых типов преобразований. Например, если действие зависит от нескольких переменных, которые в совокупности могут продуцировать большое число комбинаций. Для описания подобных действий используются таблицы и деревья решений.

Таблица решений (ТР) спецификации процесса – это задание матрицы, отображающей множество входных условий во множество действий.

ТР состоит из двух частей и, как правило, из одной конструкции «ЕСЛИ–ТО». Верхняя часть таблицы используется для определения условий. Нижняя часть ТР используется для определения действий. В ней показывается, какие конкретно действия и в какой последовательности выполняются, когда определенная комбинация условий имеет место. В конструкции «ЕСЛИ номер найден, ТО открыть шлагбаум», «номер найден» является условием, а «открыть шлагбаум» – действием.

Рассмотрим пример для нашей системы, слегка усложнив исходную задачу дополнительными условиями:

1. Совпал номер: зажечь зеленую лампу и открыть шлагбаум.
2. Не совпал номер: зажечь красную лампу.

Пример приведен в таблице 3.

Таблица 3

У1	Совпал номер	д	н
	Условия	1	2
У2	Не совпал номер	н	д

	Действия		
Д1	зажечь зеленую лампу	1	
Д2	открыть шлагбаум	2	
Д3	зажечь красную лампу		1

Вариантом таблицы решений является дерево решений (ДР), позволяющее взглянуть на процесс условного выбора с позиции схемы (рис. 13). *Пример дерева решений*

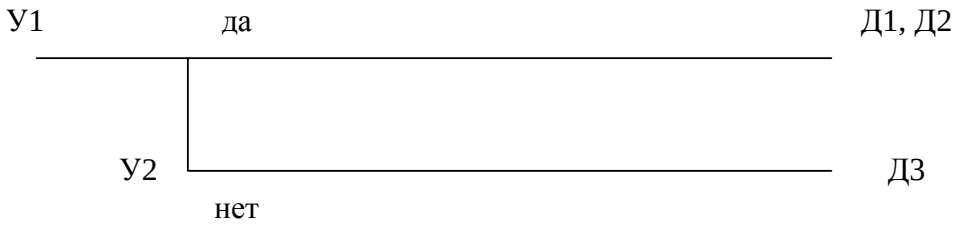


Рис. 13. Дерево решений

8.3 Визуальные языки проектирования спецификаций

Визуальные языки проектирования базируются на основных идеях структурного программирования и позволяют определять потоки управления с помощью специальных иерархически организованных схем.

Одним из наиболее известных подходов к визуальному проектированию спецификаций является подход с использованием FLOW-форм. Каждый символ FLOW-формы имеет вид прямоугольника и может быть вписан в любой внутренний прямоугольник любого другого символа. Каждый символ является блоком обработки. Каждый прямоугольник внутри любого символа также представляет собой блок обработки (рис. 14).

Дальнейшее развитие FLOW-формы получили в диаграммах Насси-Шнейдермана. На этих диаграммах символы последовательной обработки и цикла изображаются также, как и соответствующие символы FLOW-форм. В символах условного выбора и case-выбора собственно условие располагается в верхнем треугольнике, выбираемые варианты - на нижних сторонах треугольника, а блоки обработки - под выбираемыми вариантами (рис. 15).

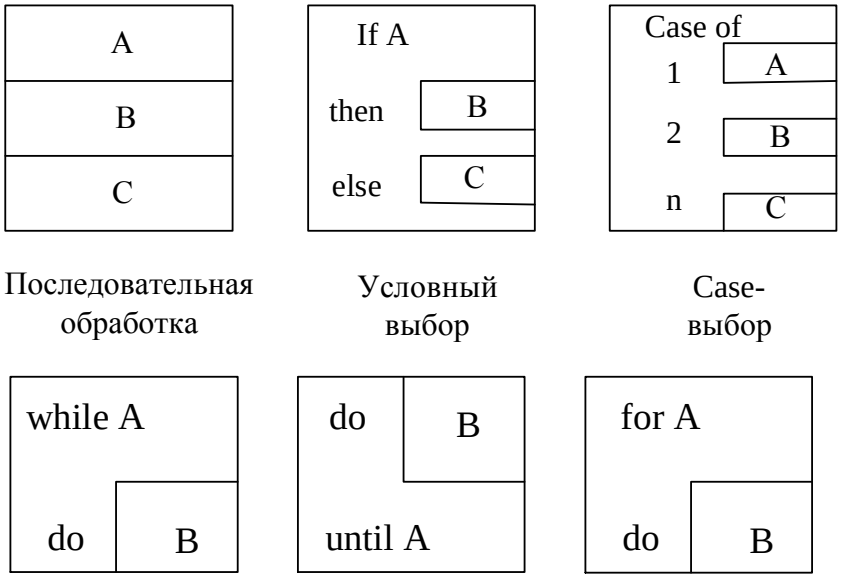


Рис. 14. Символы FLOW-форм.

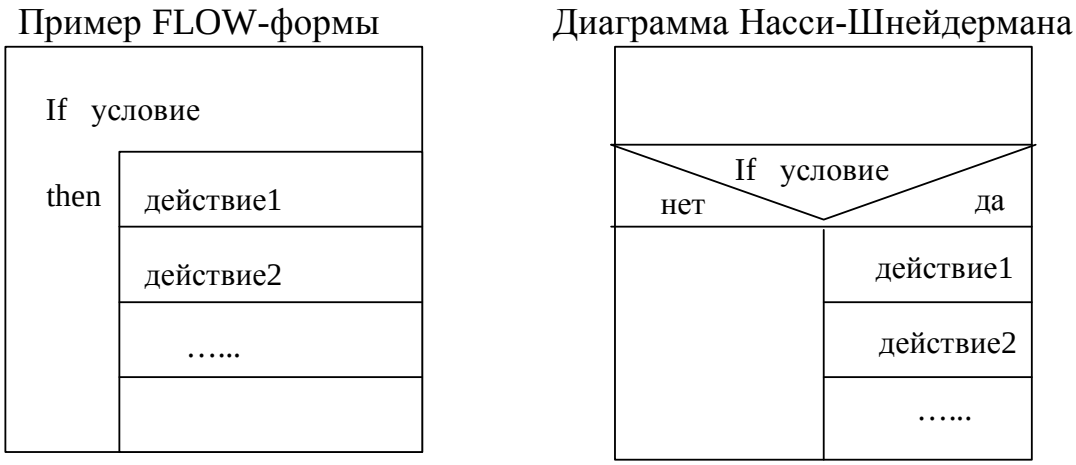


Рис. 15. Блоки ветвления FLOW-формы и диаграммы Насси-Шнейдермана

8.4 Сравнение методов задания спецификаций процессов

1. Языки программирования – наиболее трудный метод задания СП. Языки программирования привязаны к деталям реализации и трудно согласуются с DFD.

2. Структурированный естественный язык применяется в случаях, когда детали СП известны не полностью.

Достоинство: быстрое проектирование СП, прост в использовании, легко понимаем.

Недостатки: отсутствие процедурных возможностей и неспособность к автоматической кодогенерации.

3. Таблицы и деревья решений.

Достоинство: позволяют управлять сложными комбинациями условий и действий, обеспечивают визуальное (табличное и графическое, соответственно) представление СП и легко понимаемы конечным пользователем, таблицы решений позволяют легко идентифицировать несущественности и бреши в СП.

Недостаток: отсутствие процедурных возможностей.

4. Визуальные языки проектирования.

Достоинство: поддерживаются автоматической кодогенерацией, позволяют осуществлять декомпозицию СП.

Недостаток: трудность модификации СП при изменении деталей.

Лекция 9

Модели данных, событийные модели

9.1 Модели данных, ER-диаграммы

Диаграммы «сущность-связь» предназначены для построения информационной модели системы (модели данных). ER-диаграммы обеспечивают способы определения данных и показывают связи между ними. ERD непосредственно используются для проектирования реляционных баз данных.

С помощью ER-диаграммы детализируются хранилища данных системы и ее сущности, а также способы их взаимодействия. По полученной схеме можно сгенерировать логическую модель БД в любой СУБД.

ER-диаграммы идентифицируют объекты предметной области (сущности), их свойства (атрибуты) и отношения с другими объектами – связи. Нотация ERD была впервые введена П. Ченом (Chen) и получила дальнейшее развитие Баркером. Изначально была ориентирована на реляционные данные, но может применяться для сетевых и иерархических структур.

Сущность – множество экземпляров реальных или абстрактных объектов (событий предметов, состояний, людей), обладающих общими характеристиками или атрибутами. Любой объект системы должен быть представлен только одной сущностью. Имя сущности должно отражать тип или класс объекта, а не его единичный экземпляр. Сущность изображается в виде прямоугольника, вверху которого располагается имя сущности. В прямоугольнике могут быть перечислены атрибуты сущности (рис.16).

Каждая сущность должна обладать следующими свойствами:

- каждая сущность должна иметь уникальное имя;
- сущность обладает одним или несколькими атрибутами, которые либо принадлежат сущности, либо наследуются через связь;
- сущность обладает одним или несколькими атрибутами, которые однозначно идентифицируют каждый экземпляр сущности;
- каждая сущность может обладать любым количеством связей с другими сущностями модели.

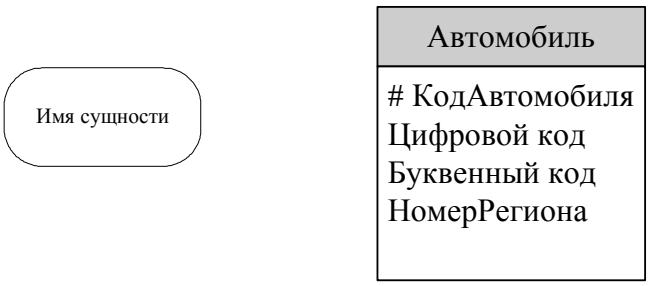


Рис. 16. Графическое обозначение сущности. Пример

Атрибут – любая характеристика сущности, значимая для рассматриваемой предметной области и предназначенная для квалификации, идентификации, классификации, количественной характеристики или выражения состояния сущности. Экземпляр сущности должен обладать единственным определенным значением для ассоциированного атрибута. Атрибут может быть либо обязательным, либо необязательным. Обязательность означает, что атрибут не может принимать неопределенных значений (null values). Атрибут может быть либо описательным (т.е. обычным дескриптором сущности), либо входить в состав уникального идентификатора (первичного ключа).

Уникальный идентификатор – это атрибут или совокупность атрибутов и/или связей, предназначенная для уникальной идентификации каждого экземпляра данного типа сущности. Уникальный идентификатор – это ключевой атрибут. Ключевой атрибут в нотации Баркера определяется решеточкой #.

Связь (Relationship) или Отношение – поименованная ассоциация между двумя сущностями, значимая для рассматриваемой предметной области. Связи может даваться имя. Все связи являются бинарными – это линии с двумя концами. Для каждой связи определяется степень множественности и обязательность.

Степень связи и обязательность графически изображаются следующим образом (рис. 17).

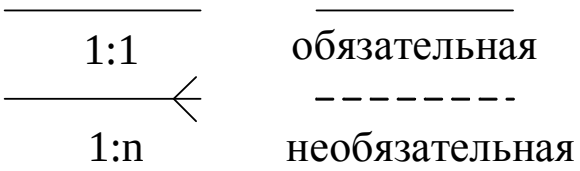


Рис. 17. Виды связей в нотации Баркера

Распространены следующие типы отношений:

a. 1*1 (один к одному);
b. 1*m (один ко многим);
c. n*m (многие ко многим).

Одиночная линия означает «один», «птичья лапка» – «многие».

Построение модели ERD включает в себя следующие основные этапы:

1) идентификация сущностей, атрибутов и первичных ключей;

2) идентификация отношений между сущностями и указание типов отношений;

3) разрешение неспецифичных отношений.

Чтобы не допустить аномалий при обработке данных, используют нормализацию (разрешение неспецифичных отношений). Как правило, при нормализации используются концепции Кодда. Обычно используют три типа нормализованных схем. Чаще всего реализуют схемы в ЗНФ, в которой должны быть устранены все отношения «многие ко многим» – третий этап построения информационной модели. Неспецифичные отношения разрешаются введением третьей сущности – ассоциативной – представляющей из себя пары реальных объектов, взаимодействующих между собой.

9.1 Спецификации управления

На функциональной диаграмме может использоваться так называемое расширение реального времени – это дополнение модели функционирования средствами описания управляющих событий в системах реального времени. При этом существуют управляющие процессы, потоки и хранилища (рис. 18).

Управляющий процесс – это процесс, выполняющий некоторую управленческую деятельность в системе.

Управляющий поток – некоторая управляющая информация. Имя его всегда имя существительное или с прилагательными, но без глаголов. Обычно имеет дискретное, а не непрерывное значение.

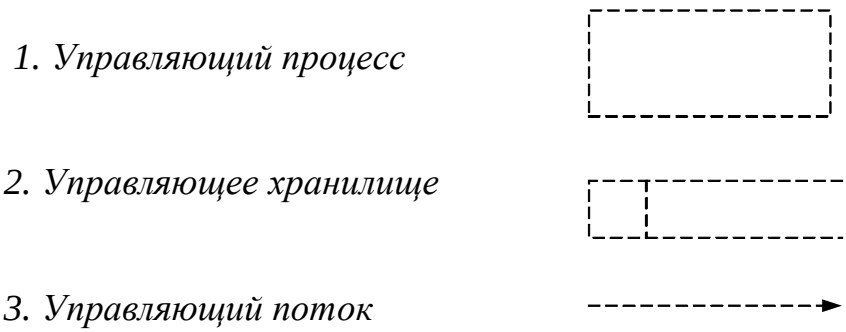


Рис. 18. Элементы управления (расширения DFD диаграмм)

Режим выполнения процесса, запускаемого управляющим потоком, зависит от типа управляющего потока. Имеются следующие типы управляющих потоков:

1. T-поток (trigger flow). Запускает выполнение процесса.

2. A-поток (activator flow). Является потоком управления процессом, который может изменять выполнение отдельного процесса. Пока поток есть, процесс выполняется. Поток пропадает – процесс останавливается.

3. E/D-поток (enable/disable flow). Является потоком управления процессом, который может переключать выполнение отдельного процесса. Тече-

ние по E-линии вызывает выполнение процесса, которое продолжается до тех пор, пока не возбуждается течение по D-линии.

Декомпозицию управляющих процессов осуществляют с помощью STD – диаграмм (Рис. 19).

Спецификации управления – предназначены для моделирования и документирования аспектов системы, зависящих от времени или реакции на событие, т. е. для описания управляющих процессов.

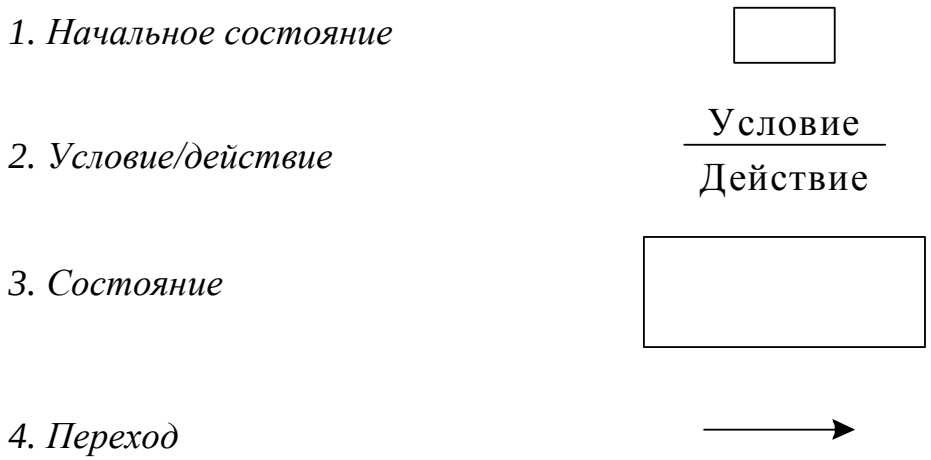


Рис. 19. Элементы STD диаграмм

С помощью STD может моделироваться функционирование системы на основе ее предыдущего и текущего состояния. STD включает следующие нотации:

Состояние – рассматривается как условие устойчивости для системы. Имя состояния должно отражать реальную ситуацию, в которой находится система.

Начальное состояние – узел STD, являющийся стартовой точкой для начального системного перехода. Начальное состояние всегда одно единственное, соответствующее состоянию системы после ее инсталляции, но перед началом реальной обработки, а также любое число завершающих состояний.

Переход – перемещение моделируемой системы из одного состояния в другое. Имя перехода идентифицирует событие являющееся причиной перехода и управляющее им. Как правило, это управляющий поток из внешнего мира или внутри системы, возникающий при выполнении некоторого условия.

Условие – событие, вызывающее переход и идентифицируемое именем перехода. Если в условии участвует входной управляющий поток, то его имя внутри условия заключается в кавычки. Условие с переходом может связывать действие, выполняющееся, если будет переход.

Действие – операция, которая может иметь место при выполнении перехода. Условие – это есть некоторое событие в системе, которое она способна обнаружить и на которое должна отреагировать определенным образом,

изменяя свое состояние, а действие – это отклик системы, посылаемый во внешнее окружение в ответ на обнаружение события.

С учетом данной информации проведем доработку функциональной диаграммы системы автоматического управления шлагбаумом. Добавим процесс «Управление считыванием номера», который запускает выполнение процесса «Получить номер». Активизация этого процесса осуществляется каждый раз после принятия решения о пропуске.

Составим диаграмму спецификации управления с помощью STD-диаграммы (рис. 20).

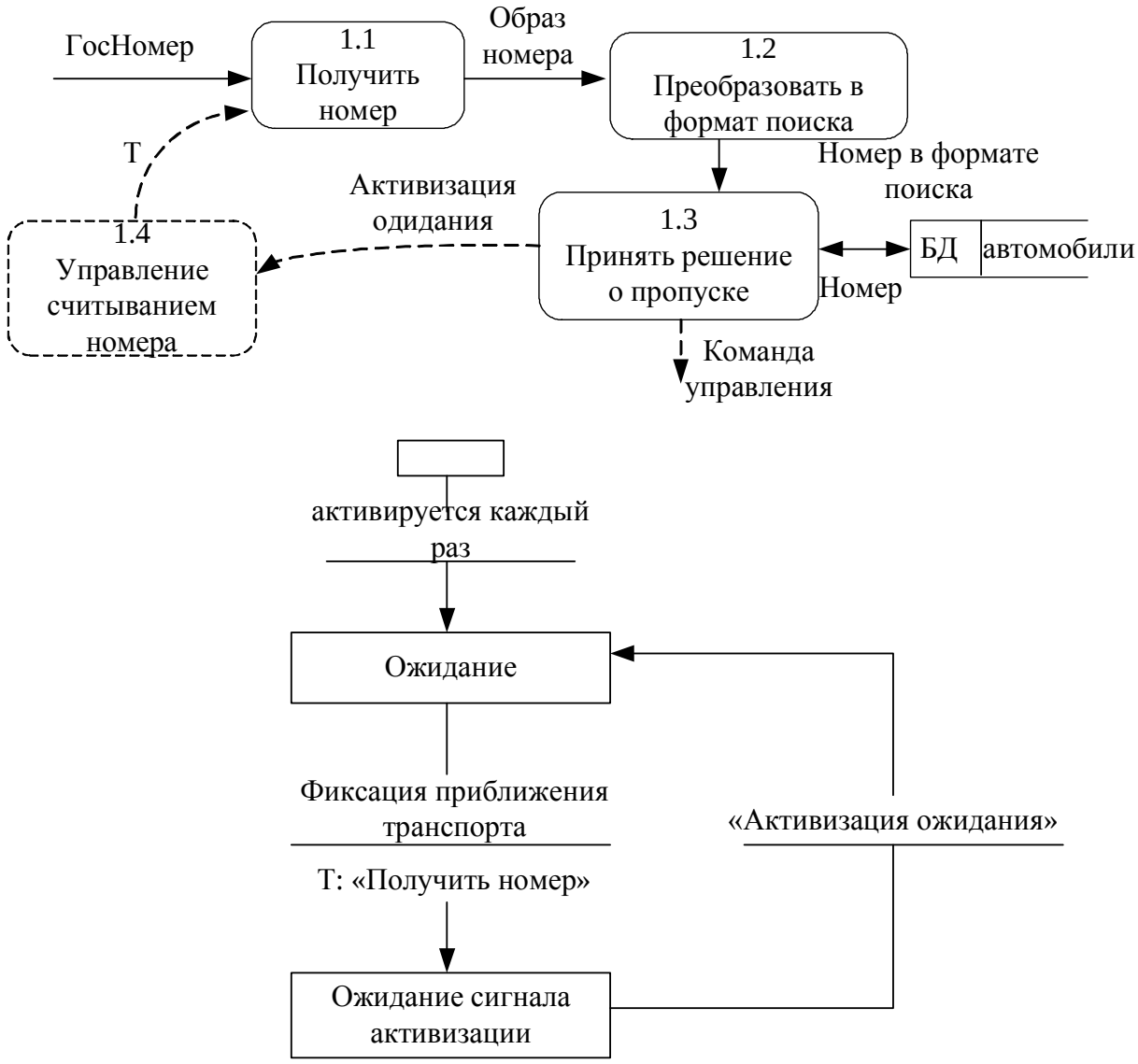


Рис. 20. Пример STD диаграммы

Лекция 10

Структурное проектирование

На этапе структурного анализа строится модель требований, состоящая из множества взаимоувязанных диаграмм, текстов и словаря данных. Модель требований описывает, что должна делать система, без детализации, как это будет осуществляться. Для осуществления детализации реализации используется фаза «Проектирование» ЖЦ, на которой определяется, как реализуются требования, зафиксированные на фазе анализа.

На данном этапе строится модель реализации, демонстрирующая, как система будет удовлетворять предъявленным к ней требованиям (без технических подробностей). Структурное проектирование – мост между анализом и реализацией. Модель реализации представляет собой структурные карты, которые показывают, как системные требования будут отображаться комбинацией программных структур.

Структурные карты – инструмент для демонстрации структуры системы и составляющих ее программных модулей, а также их связей друг с другом.

При построении структурных карт используются две техники:

1. Структурные карты Константайна.
2. Структурные карты Джексона.

Структурные карты Константайна предназначены для описания отношений между модулями, построения межмодульной иерархии.

Используют фундаментальные элементы, утвержденные IBM, ISO и ANSI. Базовый элемент структурных карт – модуль.

Модули – это базовые строительные блоки программной системы. Все виды модулей имеют ряд общих свойств, основные из которых:

1. Модуль состоит их множества операторов ЯП, записанных последовательно.
2. Модуль имеет имя, по которому к нему можно ссылаться, как к единому фрагменту
3. Модуль может принимать и передавать данные как параметры в вызывающей последовательности или связывать данные через фиксированные ячейки и области памяти.

В структурных картах Константайна используются четыре типа модулей (рис. 21):

1. Модуль. Используется для представления обрабатывающего фрагмента.
2. Подсистема. Ранее определенный модуль. Может повторно использоваться любое число раз на любых структурных картах.
3. Библиотека. Отличается от подсистемы тем, что определена вне проекта данной системы.

4. Область данных. Используется для указания модулей, содержащих исключительно области глобальных/распределенных данных.

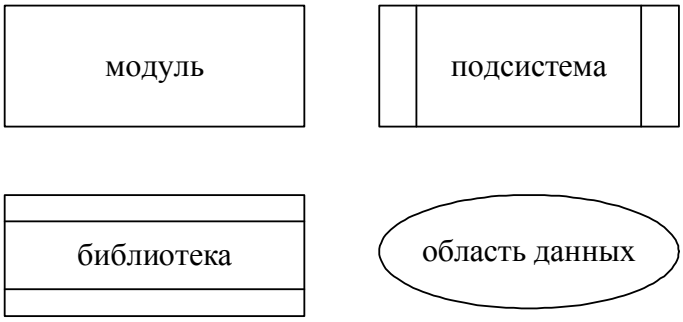


Рис. 21. Программные модули по Константайну

Связи по данным и управлению между модулями раскрываются в потоках-вызовах.

Структурные карты Джексона заключаются в создании диаграмм (структурных карт) для описания внутримодульных (а иногда и межмодульных) связей. Техника близка к традиционным блок-схемам.

Диаграммы Джексона имеют базовый компонент – структурный блок, а также процедурный и библиотечный как разновидности структурного (рис. 22).

1. Структурный блок – функция или блок кодов с одним входом и одним выходом.
2. Процедурный блок – специальный вид структурного блока, представляющий вызов ранее определенной процедуры.
3. Библиотечный блок – аналог процедурного и представляет вызов библиотечного модуля.

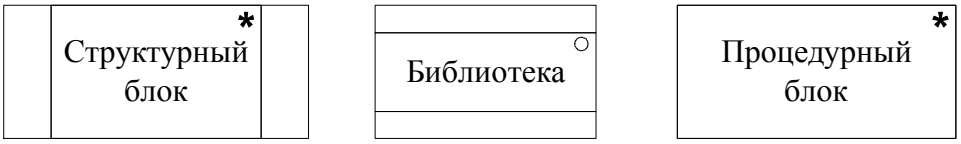


Рис. 22. Программные модули по Джексону

Для взаимоувязывания блоков используются связи следующих типов:

1. Последовательная связь, обеспечивающая последовательное выполнение слева направо.
2. Параллельная связь, обеспечивающая одновременное выполнение блоков
3. Условная связь, обеспечивающая выбор одной из альтернатив.
4. Итерационная связь, обеспечивающая выполнение блока в цикле.

Лекция 11

Характеристики модели реализации

Один из фундаментальных принципов структурного проектирования – большая система должна быть расчленена на обозримые модули. При этом разделение системы на модули должно осуществляться с выполнением следующих требований:

1. Чтобы модули были как можно более независимы (критерий сцепления - coupling).
2. Чтобы каждый модуль выполнял единственную (связанную с общей задачей) функцию (критерий связности - cohesion).

Сцепление надо уменьшать, связность надо увеличивать.

11.1 Сцепление

Сцепление является мерой взаимозависимости модуля, т. е. насколько хорошо модули отделены друг от друга. В хорошем проекте сцепления должны быть минимизированы, модули должны быть слабозависимыми или вообще независимыми по возможности. Слабое сцепление обеспечивает:

- уменьшение вероятности появления «волнового эффекта» (ошибка в одном модуле влияет на работу других модулей);
- уменьшает риск появления «эффекта ряби» (внесение изменений, например, при исправлении ошибки приводит к появлению новых ошибок);
- упрощение сопровождения модуля;
- упрощение системы для понимания.

Слабое сцепление может быть достигнуто за счет комбинирования трех следующих способов действий:

- удаления необязательных связей;
- уменьшения количества необходимых связей;
- упрощения необходимых связей.

На практике существуют три основных типа сцепления, используемые системными проектировщиками для связи модулей (рис.23): нормальное сцепление, сцепление по общей области и сцепление по содержимому.

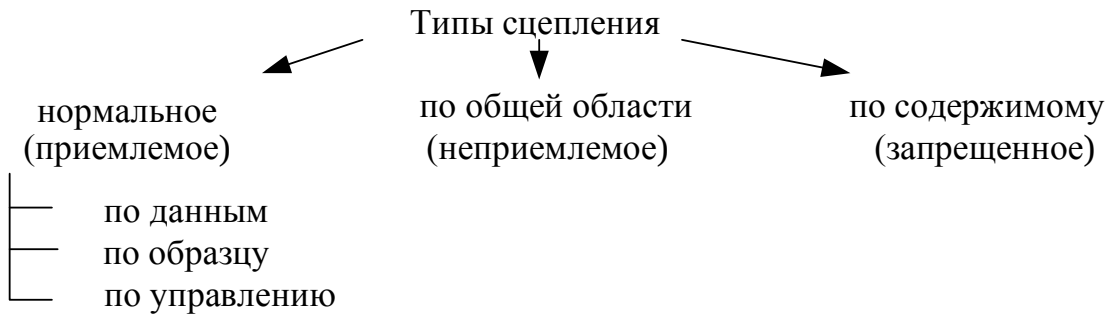


Рис. 23. Типы сцепления

Два модуля являются нормально сцепленными, если: первый модуль вызывает второй; второй возвращает управление первому; вся информация, передаваемая между модулями, представляется значениями параметров при вызове.

Сцепление по данным (data coupling). Два модуля сцеплены по данным, если они взаимодействуют через передачу параметров и при этом каждый параметр является элементарным информационным объектом. В случае небольшого количества передаваемых параметров сцепление по данным обладает наилучшими характеристиками.

Два модуля сцеплены по образцу (stamp coupling), если один посылает другому составной информационный объект, т.е. объект, имеющий внутреннюю структуру. Примером составного объекта может быть объект *Данные о клиенте*, включающий в себе поля: *Название организации*, *Почтовый адрес*, *Номер счета* и т.п.

Два модуля сцеплены по управлению (control coupling), если один посылает другому информационный объект – флаг, предназначенный для управления его внутренней логикой. Существует два типа флагов: описательный и управляющий. Описательный флаг описывает произошедшую ситуацию, управляющий флаг используется для декларирования определенных действий в вызываемом модуле. Управляющие флаги усиливают сцепление.

Два модуля являются сцепленными по общей области (common coupling), если они ссылаются к одной и той же области глобальных данных. Сцепление по общей области является плохим, хотя и может использоваться.

Два модуля являются сцепленными по содержимому (content coupling), если один модуль изменяет значения информационных объектов в другом модуле или если один модуль изменяет код другого модуля. Любые два модуля могут быть сцеплены более чем одним способом. В этом случае тип их сцепления определяется худшим типом сцепления.

11.2 Связность

Связность – это мера функциональной зависимости объектов внутри одного модуля. Размещение сильно связанных объектов в одном и том же модуле уменьшает межмодульные взаимосвязи и взаимовлияния. Типы связности приведены на рис. 24.

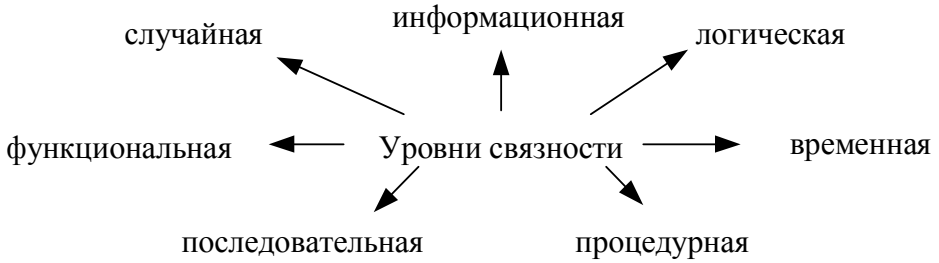


Рис. 24. Типы связности

Функционально связный модуль содержит объекты, предназначенные для выполнения только одной задачи, например: *Расчет заработанной пла-*

ты, Считывание данных кредитной карты. Каждый модуль имеет одну четко определенную цель, при его вызове выполняется только одна задача (рис. 25).

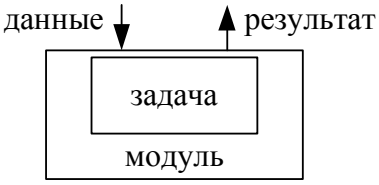


Рис. 25. Функциональная связность

Модуль имеет последовательную связность, если его объекты охватывают подзадачи, для которых выходные данные одной из подзадач служат входными данными для следующей (рис. 26), например: *Открыть файл - Прочитать запись - Закрыть файл*.

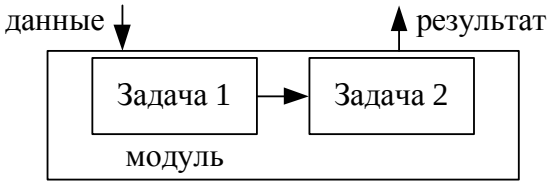


Рис. 26. Последовательная связность

Информационно связный модуль содержит объекты, использующие одни и те же входные или выходные данные (рис. 27).

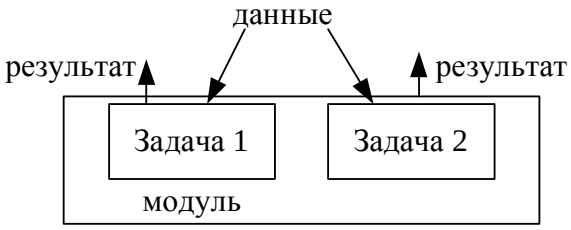


Рис. 27. Информационная связность

Процедурно связный модуль является модулем, объекты которого включены в различные (и возможно несвязные) подзадачи, в которых управление переходит от каждой подзадачи к последующей (рис. 28).

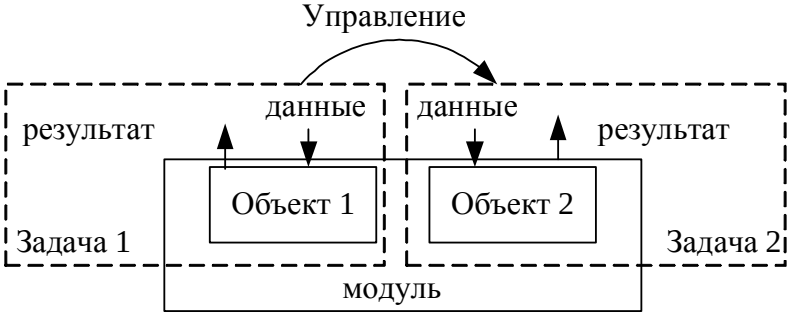


Рис. 28. Процедурная связность

Временно связным является модуль, объекты которого включены в подзадачи, связанные временем исполнения (рис. 29).

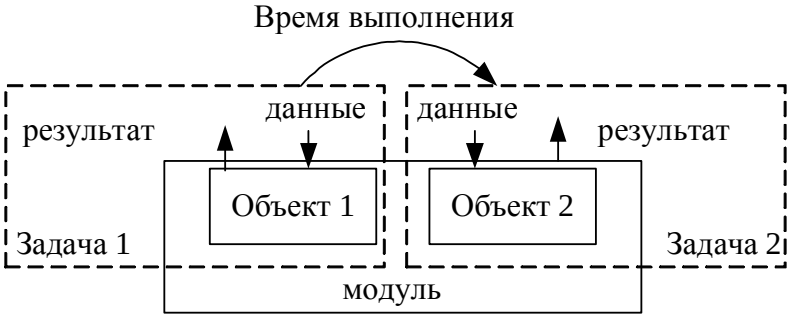


Рис.29. Временная связность

Модулем с логической связностью является модуль, объекты которого содействуют решению одной общей подзадачи (рис. 30).

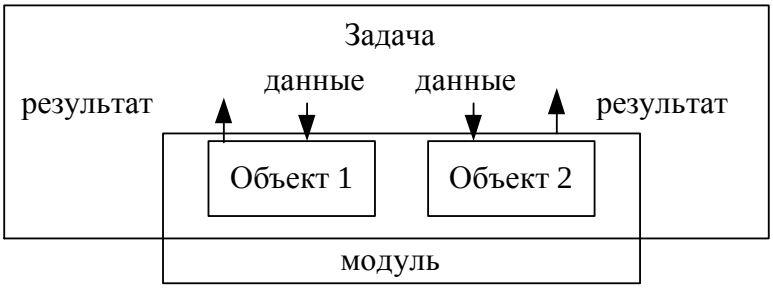


Рис. 30. Логическая связность

Случайно связным является модуль, объекты которого соответствуют подзадачам, незначительно связанным друг с другом (рис. 31).

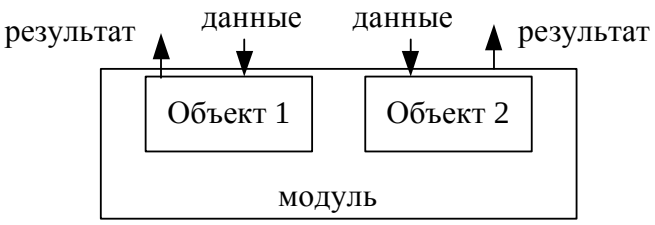


Рис. 31. Случайная связность

Лекция 12

Понятие бизнес-процесса

12.1 Понятие бизнес-процесса

Обычный подход к анализу предметной области, к которой можно отнести часто встречающиеся задачи анализа деятельности предприятий, для которых планируется разработка ИС, включает в себя деление этой деятельности на технологические единицы, которые в дальнейшем будут выделены как функции и далее на функциональной структуре представлены соответствующим образом. Но если рассматривать деятельность предприятий как экономических структур, то их деятельность можно разделить не только на технологические единицы, но и на экономические.

Экономические отношения подразумевают, как правило, покупателя-продавца или производителя-потребителя. Таким образом, продукты деятельности предприятий должны иметь своего потребителя как внутри предприятия, так и за его пределами. Проектирование систем на основе не только технологического, но и экономического анализа позволило рассматривать структуру деятельности предприятий с точки зрения новой категории – бизнес-процесса.

Бизнес-процесс – это набор операций, которые, вместе взятые, образуют результат, имеющий ценность для потребителя. При этом бизнес-процесс представляет собой некоторую деятельность, получающую входные данные одного или нескольких типов и выдающую результат, имеющий ценность для клиента.

Пример: разработка нового продукта, обучение нового специалиста. Потребителем данных продуктов не обязательно являются сторонние люди и организации. В рамках предприятия также могут быть отделы, которые в дальнейшем новый продукт будут, например, выдавать на рынок потребителю, что тоже можно рассматривать как бизнес-процесс. Процесс выполнения заказа на входе получает заказ и выдает в качестве результата заказанные товары.

Таким образом, деятельность предприятия может быть представлена в виде иерархической структуры бизнес-процессов. Бизнес-процесс рассматривается как совокупность различных видов деятельности, на входе которой имеется один вид ресурса или более, а на выходе получается продукт, имеющий ценность для потребителя внутри предприятия или за его пределами. Как правило, бизнес-процессы реализуются на предприятии целыми подразделениями.

При разработке проектов для предприятия производится анализ его в виде бизнес-процессов и составляется модель предприятия «Как есть!».

Для каждого бизнес-процесса дается ответ на вопросы:

1. Что в него поступает на входе?
2. Результат работы бизнес-процесса?

3. Чем он руководствуется?
4. Кто его выполняет?
5. Какие функции выполняются в рамках бизнес-процесса.

12.2 Методология Мартина

Мартин Фаулер разработал IE-методологию (рис. 32), представляющую общую стратегию разработки ИС. Основное внимание уделяется планированию и этапу анализа. Поскольку ПО является частью ИС вообще, то и разработка ПО входит как часть разработки системы в целом.

Основные идеи методологии:

1. Поэтапный подход к разработке приложений, базирующийся на плане развития ИС.
2. Направленность на моделирование данных, а затем на функциональное моделирование.

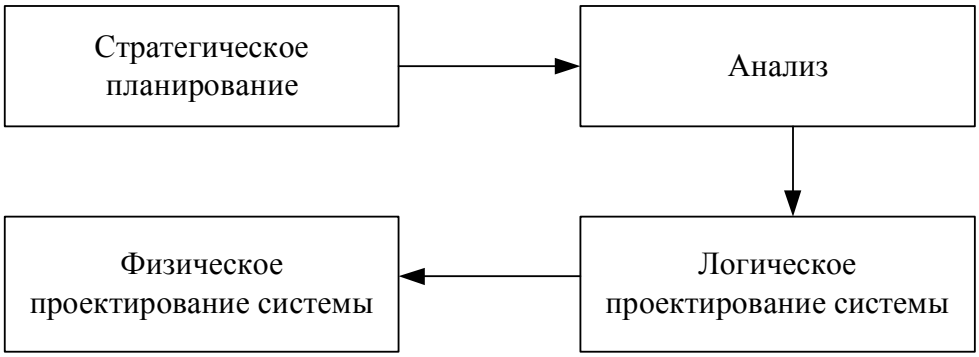


Рис. 32. Основные этапы методологии Мартина

Первый этап – стратегическое информационное планирование – направлен на решение следующих задач:

1) определение основной цели системы и способов ее достижения;
2) составление модели предметной области «как есть», отражающей существующую специфику и бизнес-процессы;
3) выделение основных задач системы;
4) составление модели данных предметной области «Сущность – связь»;
5) определение порядка разработки ИС.

Бизнес-процессы представляются иерархическими древовидными структурными диаграммами, структуры данных в нотации ERD.

На втором этапе прорабатываются частные выделенные задачи, позволяющие решить основные задачи, поставленные на первом этапе. Декомпозиция процессов осуществляется в виде DFD, проводится нормализация ERD, составляются матрицы «Сущность-процесс» для соотнесения данных и процессов. На третьем этапе выполняется логическое проектирование, включающее проектирование спецификаций процессов, проект пользовательских

интерфейсов, преобразование модели ERD в логическую модель данных. Этот этап аналогичен разработке ПО в школе SE.

На последнем этапе осуществляется непосредственная реализация системы – программирование.

12.3 Собственные методологии фирм разработчиков ПО

На основе классического структурного анализа и проектирования крупными фирмами были разработаны свои собственные методологии, ориентированные на конкретный программный продукт этих фирм и также базирующиеся на понятии бизнес-процесса. Это известные методологии:

1. CDM фирмы ORACLE для продукта Designer/2000.
2. DATARUN фирмы Computer Systems Advisers для продукта SILVERRUN.

CDM включает в себя 11 основных процессов разработки:

1) постановка задачи;
2) исследование существующих систем;
3) определение архитектуры системы;
4) построение базы данных;
5) проектирование и реализация модулей;
6) преобразование данных;
7) документирование;
8) тестирование;
9) обучение;
10) внедрение;
11) поддержка.

CDM для проектирования информационных систем предлагает три варианта в зависимости от сложности создания системы:

1. Классический – включает все 11 процессов для наиболее сложных систем.
2. Быстрая разработка – для средних проектов. Включает только 1, 3, 4, 5, 8, 10 процессы.
3. Облегченная разработка – для небольших систем. Основывается на использовании прототипов (ранее созданных решений).

Одной из наиболее распространенных в мире электронных методологий является методология DATARUN. В соответствии с методологией DATARUN ЖЦ ПО разбивается на стадии, которые связываются с результатами выполнения основных процессов, определяемых стандартом ISO 12207. Сущность методологии DATARUN – построение комплекса взаимосвязанных моделей будущей системы. Данная методология – информационно-ориентированная, т. е. на первом месте осуществляется построение хранилища данных, затем идет проработка функций обработки этой информации.

Методология DATARUN опирается на две модели или на два представления:

– модель организации;
– модель ИС.

Модель организации – то, от чего мы отталкиваемся при проектировании, модель ИС – то, что мы получаем в результате проектирования.

Информационная система создается последовательным построением ряда промежуточных моделей, начиная с модели бизнес-процессов и заканчивая моделью программы, автоматизирующей эти процессы. В упрощенном виде это представляется как последовательность шагов проектирования системы по DATARUN:

– BPM (Business Process Model) – модель бизнес-процессов;
– PDS (Primary Data Structure) – структура первичных данных;
– CDM (Conceptual Data Model) – концептуальная модель данных;
– SPM (System Process Model) – модель процессов системы;
– ISA (Information System Architecture) – архитектура информационной системы.

Для пользовательских приложений разрабатывается:

– ADM (Application Data Model) – модель данных приложения;
– IPM (Interface Presentation Model) – модель представления интерфейса;
– ISM (Interface Specification Model) – модель спецификации интерфейса.

Лекция 13

CASE-технологии

13.1 Понятие CASE-средства

Современные CASE-средства охватывают обширную область поддержки многочисленных технологий проектирования ИС: от простых средств анализа и документирования до полномасштабных средств автоматизации, покрывающих весь жизненный цикл ПО. CASE-технологии не могут считаться самостоятельными методологиями, они только автоматизируют процесс разработки ПО для информационных систем.

CASE-средство – любое программное средство, автоматизирующее ту или иную совокупность процессов жизненного цикла программного обеспечения. CASE-средства развивались в два этапа:

1. CASE-средства на первом этапе создавались для системных аналитиков и проектировщиков (поддерживают графические модели, словари данных, проектирование спецификаций). Основное внимание уделяется этапу анализа, определению требований, проектированию БД (инструмент для поддержки методов структурно анализа и проектирования).

2. CASE-средства второго этапа ориентированы на полную поддержки ЖЦПО. В первую очередь – кодогенерации. CASE-средства второго поколения имеют набор компонент, каждая из которых отвечает за определенный

этап ЖЦПО. Набор компонент может варьироваться в зависимости от требований проекта.

Стоимость CASE-средств меняется в зависимости от набора их функций.

К настоящему моменту наиболее интенсивное развитие получили два главных направления применения CASE-средств:

1. BPR (business process reengineering) – перепроектирование бизнес-процессов. Под перепроектированием понимается «фундаментальное переосмысление и радикальное перепланирование критических бизнес-процессов, имеющее целью резко улучшить их выполнение по отношению к затратам, качеству обслуживания и скорости».

Реинженеринг – это перепроектирование существующей системы. На основе имеющейся системы получается модель, которая дорабатывается и реорганизуется в новую систему.

2. Системный анализ и проектирование или *инженеринг*. Включает функциональное, информационное и событийное моделирование как вновь создаваемой, так и существующей системы.

Использование CASE-средств изменяет фазы ЖЦПО. Получается так называемая CASE-модель ЖЦПО (рис. 33). Фаза анализа заменяется на фазу прототипирования. Наиболее автоматизируемые фазы – контроль проекта и кодогенерация. Основное внимание должно уделяться фазе прототипирования и проектирования спецификаций.

Прототип системы можно использовать в качестве концептуальной модели для дальнейшего проектирования новой системы. Прототипы уменьшают затраты на проектирование и подготовку производства за счёт выявления возможных ошибок на ранних стадиях.

Первая стадия в CASE-модели отвечает за анализ и выбор прототипов.

Интегрированное CASE-средство (или комплекс средств, поддерживающих полный жизненный цикл программного обеспечения) содержит следующие компоненты:

- репозитарий;
- графические средства анализа и проектирования для построения моделей информационных систем;
- средства разработки приложений;
- средства конфигурационного управления (для управления процессом разработки информационных систем);
- средства документирования;
- средства тестирования;
- средства управления проектом;
- средства реинжиниринга.

Репозитарий – место для хранения моделей, интерфейсов и программных реализаций; часть окружения для манипулирования артефактами проектирования.

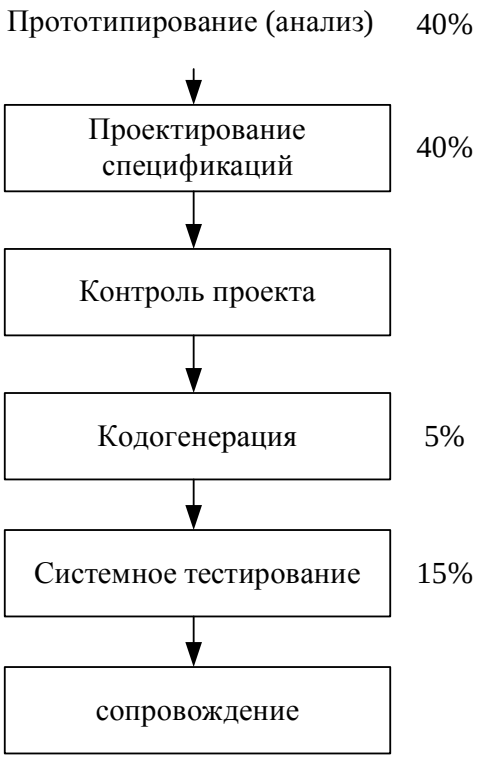


Рис. 33. Case-модель ЖЦПО

Артефакт – это элемент информации, используемый или порождаемый в процессе разработки программного обеспечения. Артефактом может быть модель, описание или программный продукт. Репозитарий – это основа CASE-средства. Он должен обеспечивать:

– инкрементный режим ввода описаний объектов;
– измененное описание должно распространяться на весь проект;
– синхронизация информации;
– контроль информации;
– хранение версий проекта.

Структура репозитария приведена на рис. 34.

Описание объекта	Идентификатор
	Имя
	Тип
	Текстовое описание
	Компоненты
	Файл-хранилище
	Область значений
Отношения с другими объектами	
Контрольная информация	
Правила обработки	

Рис. 34. Структура репозитария

Пример хранимой информации: структурные диаграммы, проекты отчетов, описание данных, определения экранов, меню, программные коды. На основе репозитария генерируются отчеты по проекту: по содержимому, по анализу объектов, по декомпозиции. Репозитарий – база для стандартизации документации по проекту.

Основные функции CASE-средства:

1. Поддержка графических моделей (характерно 4 типа: DFD, ERD, STD и структурные карты).

2. Контроль ошибок (синтаксис диаграмм, балансирование диаграмм, контроль декомпозиции).

3. Организация и поддержка репозитария (обновление, доступ, анализ).

4. Поддержка процесса проектирования и разработки (прототипирование, кодогенерация, структурные методологии).

Кодогенерация включает средства генерации каркаса ПО и средства генерации полного продукта.

13.2 Классификация CASE-средств

Все современные CASE-средства могут быть классифицированы в основном по типам и категориям.

Тип отражает функциональную ориентацию CASE-средств на те или иные процессы ЖЦ. Классификация по типам в основном совпадает с компонентным составом CASE-средств и включает следующие основные типы:

- средства анализа;
- средства анализа и проектирования;
- средства проектирования баз данных;
- средства разработки приложений (программирование);
- средства реинжиниринга.

Вспомогательные типы включают:

- средства планирования и управления проектом;
- средства конфигурационного управления;
- средства тестирования;
- средства документирования.

Классификация по категориям CASE-средств определяет степень интегрированности по выполняемым функциям и включает:

- tools – отдельные локальные средства, решающие небольшие автономные задачи;
- toolkit – набор частично интегрированных средств, ориентирующихся на одну фазу ЖЦПО;
- workbench – средства, поддерживающие весь ЖЦИС и связанные общим репозиторием.

Лекция 14

Технология внедрения CASE-средств

Приведенная в данном разделе технология базируется в основном на стандартах IEEE (IEEE – Institute of Electrical and Electronics Engineers – Институт инженеров по электротехнике и электронике).

Внедрение включает все действия от оценки первоначальных потребностей до полномасштабного использования CASE-средств в различных подразделениях организации-пользователя. Процесс внедрения CASE-средств состоит из следующих этапов:

1. Определение потребностей в CASE-средствах.
2. Оценка и выбор CASE-средств.
3. Выполнение пилотного проекта.
4. Практическое внедрение CASE-средств.

Технология внедрения представлена на рис. 35.

14.1 Определение потребностей в CASE-средствах

Данный этап включает достижение понимания потребностей организации и технологии последующего процесса внедрения CASE-средств. Он должен привести к выделению тех областей деятельности организации, в которых применение CASE-средств может принести реальную пользу. Результатом данного этапа является документ, определяющий стратегию внедрения CASE-средств.

Анализ возможностей организации в отношении ее технологической базы, персонала и используемого ПО, как правило, может быть формальным и неформальным. Формальный – через модели зрелости технологических процессов предприятия CMM и согласно стандартам ИСО 9000. Неформальный – путем анкетирования, опросов с целью сбора информации о проектах, ведущихся на предприятии, их характеристиках, документации, которая выпускается при проектировании, технологической базе, персонале. Оценка готовности организации к внедрению CASE-технологии должна быть объективной, иначе результат внедрения будет отрицательным. Определение потребностей должно выполняться в сочетании с обзором рынка CASE-средств, поскольку информация о технологиях, доступных на рынке в данный момент, может оказать влияние на потребности.

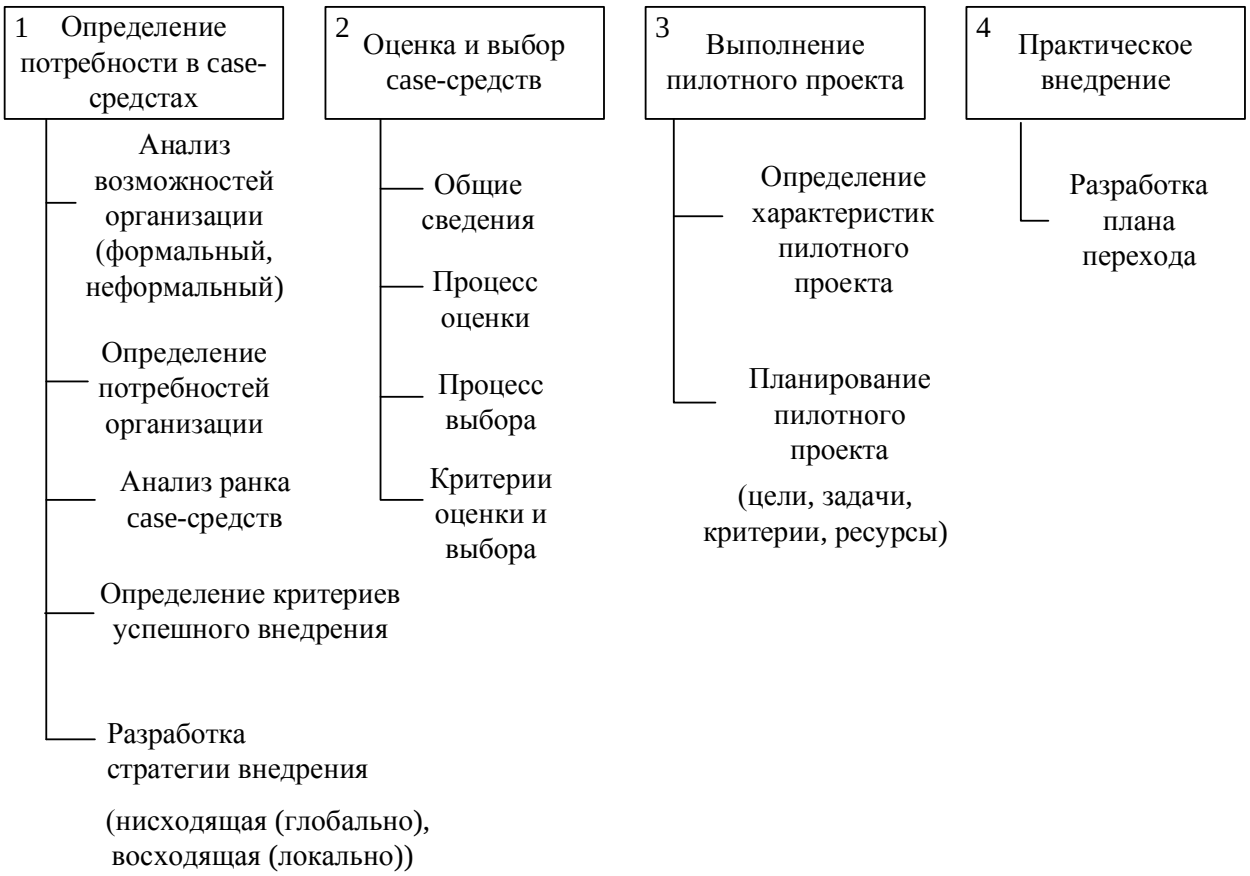


Рис. 35. Технология внедрения CASE-средств

Определению потребностей организации могут помочь ответы на следующие вопросы:

1. Каким образом продуктивность и качество деятельности организации сравниваются с аналогичными показателями подобных организаций.

2. Какие процессы ЖЦПО дают наилучшую или наихудшую отдачу; существуют ли конкретные процессы, которые могут быть усовершенствованы путем использования новых методов и средств.

Возможный эффект от внедрения CASE-средств – это поддержка реинжиниринга бизнес-процессов, ускорение разработки приложений, снижение доли ручного труда в процессе разработки и/или эксплуатации, повышение качества документирования.

Потребности организации в CASE-средствах должны соразмеряться с реальной ситуацией на рынке или собственными возможностями разработки. Результатом этапа определения потребности в CASE-средствах становится набор критериев, на базе которых производится выбор конкретного средства. Определяемые критерии должны позволять количественно оценивать степень удовлетворения каждой из потребностей, связанных с внедрением. При этом оценка должна выражаться в количественной форме – метрике.

Стратегия внедрения должна обеспечивать удовлетворение потребностей и критериев, определенных ранее. Выделяется три подхода к разработке стратегии внедрения:

1. Нисходящий – общий анализ процесса создания и сопровождения ПО на предприятии до того, как определяются требования к CASE-средствам.

2. Восходящий – определение некоторого средства или типа средств, которые потенциально могут помочь организации в улучшении выполнения текущей работы предприятии. Рекомендуется для зрелых предприятий.

3. Комбинированный.

14.2. Оценка и выбор CASE-средств

Модель процесса оценки и выбора, описывает наиболее общую ситуацию оценки и выбора, а также показывает зависимость между ними. Как можно видеть, оценка и выбор могут выполняться независимо друг от друга или вместе, каждый из этих процессов требует применения определенных критериев. Процесс оценки и выбора может преследовать несколько целей:

- оценка нескольких CASE-средств и выбор одного или более из них;
- оценка одного или более CASE-средств и сохранение результатов для последующего использования;
- выбор одного или более CASE-средств с использованием результатов предыдущих оценок.

Входной информацией для процесса оценки является:

- определение пользовательских потребностей;
- цели и ограничения проекта;
- данные о доступных CASE-средствах;
- список критериев, используемых в процессе оценки.

Результаты оценки могут включать результаты предыдущих оценок. Процесс оценки и/или выбора может быть начат только тогда, когда лицо, группа или организация полностью определили для себя конкретные потребности и формализовали их.

Оценка и накопление соответствующих данных может выполняться следующими способами:

- анализ CASE-средств и документации поставщика;
- опрос реальных пользователей;
- анализ результатов проектов, использовавших данные CASE-средства;
- просмотр демонстраций и опрос демонстраторов;
- выполнение тестовых примеров;
- применение CASE-средств в пилотных проектах;
- анализ любых доступных результатов предыдущих оценок.

Выбор CASE-средств на основе проведенного анализа и списка пользовательских критериев осуществляется методами многокритериального анализа.

14.3. Выполнение пилотного проекта

Пилотный проект – проект, цель которого состоит в экспериментальной проверке правильности решений, принятых на предыдущих этапах, и подготовка к внедрению.

Пилотный проект представляет собой первоначальное реальное использование CASE-средства в предназначенной для этого среде. Пилотный проект должен обладать многими из характеристик реальных проектов, для которых предназначено данное средство. Он преследует следующие цели:

- подтвердить достоверность результатов оценки и выбора;
- определить, действительно ли CASE-средство годится для использования в данной организации, и если да, то определить наиболее подходящую область его применения;
- собрать информацию, необходимую для разработки плана практического внедрения;
- приобрести собственный опыт использования CASE-средства.

Важной функцией пилотного проекта является принятие решения относительно приобретения или отказа от использования CASE-средства. План пилотного проекта должен содержать:

- цели, задачи и критерии оценки;
- персонал;
- процедуры и соглашения;
- обучение;
- график и ресурсы.

14.4. Практическое внедрение CASE-средств

Для перехода к практическому использованию CASE-средств разрабатывается план перехода. План перехода должен включать следующее:

- информацию относительно целей, критериев оценки, графика и возможных рисков, связанных с реализацией плана;
- информацию по приобретению, установке и настройки CASE-средств;
- информацию по интеграции каждого средства с существующими, включая их интеграцию в процессы разработки и эксплуатации ПО, существующие в организации;
- ожидаемые потребности в обучении и ресурсах;
- где будет использоваться средство и как.

Кроме того, при реализации плана перехода необходим постоянный мониторинг использования CASE-средств, обеспечения текущей поддержки, сопровождения и обновления средств по мере необходимости.

14.5 Критерии выбора готовой информационной системы

Если стоит задача выбора системы из имеющихся программных средств, то основные параметры выбора информационной системы, как правило, следующие:

1. Функциональность. Система должна решать только те задачи, которые актуальны для организации.
2. Возможность интегрировать систему с другими приложениями. На каждом предприятии существует определенный набор систем и приложений, в которых накоплена важная информация, поэтому новой системе необходимо уметь работать с уже имеющимися данными.
3. Модульность системы. Функциональность системы, распределенная по модулям, которые нет нужды приобретать и устанавливать сразу, а можно приобретать по мере необходимости.
4. Веб-интерфейс. Веб-интерфейс в настоящее время можно назвать наиболее приемлемым вариантом создания рабочих мест в системах масштаба предприятия. Достоинства: кроссплатформенность, масштабируемость системы и простое обновление функционала.
5. Надежность. Сопровождение, обеспечение сохранности данных и безопасности системы.
6. Стоимость. Один из решающих факторов.
7. Соответствие текущей инфраструктуре. Новая система должна подходить к имеющейся корпоративной платформе.

Лекция 15

Оценка качества разработанной программной продукции

15.1 Качество ПО и его характеристики

Любое ПО информационной системы, представляющее собой продукцию, может быть оценено с точки зрения качества.

При оценке качества решают следующие задачи:

- планируют уровень качества;
- контролируют показатели качества и их значения в процессе разработки и испытаний ПО;
- осуществляют эксплуатационный контроль заданного уровня качества.

Основополагающими стандартами в этой области являются:

- ISO 9000:2000 Quality management systems – Fundamentals and vocabulary. Системы управления качеством – Основы и словарь (аналог ГОСТ Р-2001);
- ISO 9001:2000 Quality management systems – Requirements. Models for quality assurance in design, development, production, installation, and servicing. Системы управления качеством – Требования. Модели для обеспечения качества при проектировании, разработке, коммерциализации, установке и обслуживании. Определяет общие правила обеспечения качества результатов во всех процессах жизненного цикла (аналог ГОСТ Р-2001) ;

– ISO 9002:1994 Quality systems – Model for quality assurance in production, installation and servicing. Системы качества – Модель для обеспечения качества при коммерциализации, установке и обслуживании;
– ISO 9003:1994 Quality systems – Model for quality assurance in final inspection and test. Системы качества – Модель обеспечения качества при финальном инспектировании и тестировании;
– ISO 9004:2000 Quality management systems – Guidelines for performance improvements. Системы управления качеством. Руководство по улучшению деятельности (аналог ГОСТ Р-2001);
– ISO 9000-3:1997. Стандарты в области административного управления качеством и обеспечения качества. Часть 3. Руководящие положения по применению стандарта ISO 9001 при разработке, поставке и обслуживании программного обеспечения.

Понятие качества ПО в виде совокупности атрибутов качества определяется стандартом ISO 9126 (аналог ГОСТ Р ИСО/МЭК 9126-93). Стандарт определяет шесть характеристик (*показателей качества ПО*), каждая из которых отражает определенные аспекты качества ПО. Но для применения данного стандарта необходимо иметь четкие требования к анализируемой программе, для того чтобы оценить ее качество на всех этапах ЖЦ.

Качество ПО – это весь объем признаков и характеристик программной продукции, который относится к их способности удовлетворять установленным или предполагаемым потребностям пользователя.

Качество оценивается с помощью критериев оценки качества ПО.

Критерий оценки качества ПО – это набор определенных и задокументированных правил и условий, которые используются для принятия решения о приемлемости общего качества конкретной программной продукции. Качество представляется набором установленных уровней – некоторой степени, в которой удовлетворяются установленные потребности.

Для определения показателей качества ПО существуют следующие методы:

1) по способу получения информации:
– измерительный (получение информации о свойствах и характеристиках ПО через измерительные инструменты);
– регистрационный (в процессе испытаний или функционирования ПО);
– расчетный (использование теории и эмпирических зависимостей данных);
– органолептический (на восприятии информации органами чувств).

2) по источнику получения информации:
– экспертный (мнение экспертов или экспертные оценки);
– социологический (анкеты-опросники).

Для оценки программной продукции у нее выделяются *характеристики* – набор свойств (атрибутов), по которым ее качество описывается и оценивается с помощью вышеперечисленных методов.

Качество ПО может быть оценено следующими характеристиками и их комплексными показателями (КП) или атрибутами:

1. Функциональные возможности (Functionality) – набор атрибутов, относящийся к сути набора функций и их конкретным свойствам. Функции – то, что реализует установленные или предполагаемые возможности.

Комплексные показатели:

- пригодность (соответствие набора функций конкретным назначениям);
- правильность (правильность результатов);
- способность к взаимодействию (совместимость);
- согласованность (придерживание стандартов);
- защищенность (способность предотвращать несанкционированный доступ).

2. Надежность (Reliability) – набор атрибутов, относящихся к способности ПО сохранять свой уровень качества функционирования за определенное время и при определенных условиях. ПО не может стареть или изнашиваться.

Комплексные показатели:

- стабильность (частота отказов и ошибок);
- устойчивость к ошибке;
- восстанавливаемость.

3. Практичность (Usability) – набор атрибутов, относящихся к объему работ, требуемых для использования или оценки этого использования в кругу пользователей продукции. Пользователи – это операторы, конечные пользователи.

Комплексные показатели:

- понятность;
- обучаемость;
- привлекательность;
- простота использования.

4. Эффективность (Efficiencies) – набор атрибутов, относящихся к соотношению между уровнем качества функционирования ПО и объемом используемых ресурсов при установленных условиях. Ресурсы могут включать другие программные продукты, технические средства, материалы, услуги сопровождающего персонала.

Комплексные показатели:

- характер изменения во времени (скорость выполнения функций, время отклика);
- характер изменения ресурсов (объем захватываемых ресурсов).

5. Сопровождаемость (Maintainability) – набор атрибутов, относящихся к объему работ, требуемых для проведения конкретных изменений (модификаций). Изменение – это исправления, усовершенствования или адаптация ПО.

Комплексные показатели:

- анализируемость;

– изменяемость;
– устойчивость;
– тестируемость.

6. Мобильность (Portability) – набор атрибутов, относящихся к способности ПО быть перенесенным из одного окружения в другое. Окружение может включать организационное, техническое и программное окружение.

Комплексные показатели:

– адаптируемость;
– простота внедрения;
– соответствие;
– взаимозаменяемость.

Оценка качества программной продукции определяется классом ПО (СРВ, ИС, военные системы). Каждая характеристика отражается набором *метрик (показателей качества)* – количественного масштаба значений этой характеристики. У разных категорий пользователей свои представления о качестве ПО.

1. Представление пользователя.

Основной интерес в применении ПО, его производительности и результатах использования. Интересуют вопросы: «Имеются ли требуемые функции? Надежность? Эффективность? Удобство использования? Переносимость?» Представление пользователя – решающее.

2. Представление разработчика.

Так как разработчики создают продукт и проходят все этапы ЖЦПО, то их интересуют не только конечные результаты продукции, но и промежуточные. И хотя разработчики и пользователи оперируют одними и теми же характеристиками, но метрики для измерения этих характеристик у них разные (пример: эффективность – реакция системы или время ожидания и доступа, маршрут, быстродействие, надежность, сопровождаемость).

3. Представление руководителя.

Руководителя интересуют коммерческие требования. Его также может интересовать оптимизация качества в пределах ограниченной стоимости, времени, трудовых ресурсов. Для систем, внедряемых на предприятии, главным критерием качества у руководства остается повышение коммерческой эффективности всего предприятия.

15.2 Модель процесса оценивания качества

Основные этапы модели – установление требований к качеству ПО, подготовка к оцениванию и оценивание (рис. 36). В свою очередь подготовка к оцениванию включает выбор метрик – количественных показателей программной продукции, которые могут быть измерены для дальнейшего использования, и определение уровней ранжирования, в которые попадут метрики.

Примеры метрик:

– полнота реализации функций. Используется для измерения пригодности;
– корректность реализации функций. Используется для измерения пригодности;
– отношение числа обнаруженных дефектов к прогнозируемому. Используется для определения зрелости;
– отношение числа проведенных тестов к общему их числу. Используется для определения зрелости;
– отношение числа доступных проектных документов к указанному в их списке. Используется для измерения анализируемости.

Далее начинается непосредственно процедура оценивания с измерениями и оценкой. Оценка – это последний этап процесса оценивания, на котором обобщается множество установленных уровней.

Результат – заключение о качестве программной продукции и затем решение руководства по выпуску или отбраковке ее.

Уровень ранжирования – установление диапазонов значений.

Рис. 36. Процесс оценки качества

Контроль качества ПО системы можно получать с помощью процессов верификации и валидации.

1. Верификация – проверка того, что продукт делался правильно, т. е. проверка того, что он разрабатывался в соответствии со всеми требованиями по отношению к процессу и этапам разработки.

2. Валидация – проверка того, что сам продукт правилен, т.е. установление того, что он удовлетворяет требованиям, ожиданиям пользователя, заказчика и других заинтересованных сторон.

Тестирование – это частный вариант верификации и валидации.

Лекция 16

Объектно-ориентированный подход к разработке программного обеспечения

16.1 Возникновение объектно-ориентированного подхода

Для того чтобы рассматривать особенности объектно-ориентированного подхода к разработке программного обеспечения, необходимо вспомнить об истории развития данного подхода и всех его основных принципов, а также рассмотреть терминологию, которая применяется в объектно-ориентированном программировании, поскольку данные вещи неразрывно связаны между собой.

Изначально первым этапом развития программирования считается создание структурного подхода (основатель Э. Дейкстра). Структурное программирование – первая методология программирования. Базируется на двух основных принципах: первый – это использование процедурного стиля программирования, второй – последовательная декомпозиция решения задачи сверху вниз. ИДЕЯ – каждую сложную систему можно разделить на более мелкие подсистемы, каждую из которых можно совершенствовать независимо!

Любую задачу можно решить путем осуществления последовательности действий. Это *алгоритмическая декомпозиция*. При этом на ее вход подаются какие-либо данные, обрабатываются и выходят. Далее начинается детальное разложение действий внутри программы. Последовательность выполнения действий осуществляется последовательным вызовом процедур, каждая из которых выполняет определенный кусочек задачи. При этом контролировать правильность выполнения алгоритма можно путем установки заглушек на каждом уровне (имитация выход процедур нижнего уровня), например, проверить правильность выполнения каждой процедуры путем проверки, что она возвращает.

Такой структурный подход широко поддерживался следующими языками программирования конца 60-х г.г.:

- Паскаль;
- Алгол-68;
- Симула;

Эволюция языков выглядит следующим образом:

1. (1954 – 1958 г.) введены математические формулы:

- Fortran 1;
- Algol-58;
- Flowmatic.

2. (1959 – 1961 г.) реализованы подпрограммы, типы данных, списки, указатели:

- Fortran 1;

– Algol-60;
– Cobol;
– Lisp.

3. (1962 – 1970 г.) созданы классы, абстрактные данные:

– Algol-68 (приемник алгола);
– PL/1;
– Pascal (приемник алгола, более мягкий);
– Simula.

Структурное программирование показало преимущество *модульного построения программ*, хотя модульность как таковая в них не поддерживается. Структура программы представляла собой набор вызовов функций. Контроль правильности их вызовов, передачи параметров осуществляется только на стадии выполнения.

Развитие объектно-ориентированного подхода стимулировалось, во-первых, быстро растущими потребностями пользователей, что требовало многократного использования уже написанных кодов; во-вторых, необходимостью сопровождения и модификации программ; в-третьих, упрощением проектирования (некоторые задачи достаточно трудно описать последовательной алгоритмизацией).

ООП – это альтернатива алгоритмической декомпозиции, которая не делит задачу на последовательность шагов – как ее решить, а представляет задачу в виде взаимодействия объектов, принимающих в ней участие и взятых из словаря предметной области. Каждый объект обеспечивает некоторое поведение, мы можем его попросить что-то сделать. Объект – моделирует предмет из реального мира.

Объектно-ориентированный метод разработки ПО произошел на основе развития всех предыдущих методов и вылился в создании объектно-ориентированных языков программирования, которые представляют собой не более чем инструмент для написания объектно-ориентированных программ.

Наиболее распространены среди них:

– C++;
– Java;
– Смолтолк (Smalltalk).

Достоинства ООП перед алгоритмическим:

1) уменьшается размер программных систем за счет использования готовых кодов;

2) в результате получаются более гибкие системы и более эволюционирующие;

3) уменьшается риск создания системы, потому что строится из проверенных кирпичиков.

16.2 Основные понятия объектно-ориентированного подхода

Основной особенностью объектно-ориентированного подхода является стремление к уменьшению семантического разрыва между языком формулирования задачи и языком программирования, в терминах которого данная задача будет решена. Структурный подход не всегда позволяет решить эту проблему или же предлагает слишком сложное решение. В отличие от него объектно-ориентированный подход позволяет описывать программные системы, моделирующие реальные системы, в виде взаимодействия объектов этих систем.

Например, для системы «банкомат» объектами могут являться: клиент, банкомат, счет (рис. 37). Для системы «шлагбаум» это: транспорт, номер, шлагбаум.

Описать предметную область можно в этом случае в виде взаимодействия объектов.

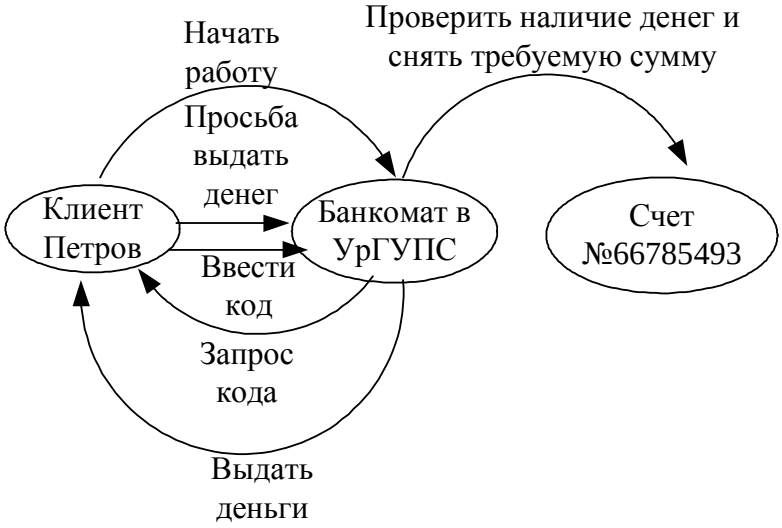


Рис. 37. Пример взаимодействия объектов

Объект в целом – это достаточно общее понятие. Поэтому различные авторы толковали его своими словами по-разному, но, тем не менее, смысл объекта следующий.

Объект – это сущность, имеющая некоторое состояние (информацию) и предоставляющая набор операций, с помощью которых можно изменять или проверять это состояние. В программной среде объект – это модель или абстракция реальной сущности.

Объекты взаимодействуют между собой через связи. Связь обозначает соединение, через которое объект может вызывать операции другого объекта, или один объект перемещает данные к другому объекту.

Чтобы один объект мог послать сообщение другому объекту, он должен его видеть. На этапе анализа вопросы видимости опускаются, и здесь они будут рассмотрены чуть позже.

При этом то, что происходит внутри того или иного объекта, не видно другим участникам системы. Подобный принцип построения систем называется инкапсуляция – упрятывание. При этом описываются только механизмы взаимодействия между объектами, а каким образом это реализовано – не важно.

Инкапсуляция – это обеспечиваемое объектами сокрытие информации.

Те операции, которые может выполнять объект, – это его интерфейс.

Интерфейс – описание того, как объект взаимодействует с окружающим миром.

Состояние объекта характеризуется значениями его атрибутов, которые могут быть как простыми величинами, так и сложными объектами. Все объекты должны отличаться друг от друга, т. е. обладать некоторой уникальностью. Это проявляется в том, что некоторые атрибуты объекта не измены, и их значения, присущие только ему, уникальны.

Объект может состоять из других объектов, являющихся его атрибутами (включение одного объекта в другой) – объект-контейнер, объект-атрибут.

В качестве атрибутов объект может использовать ссылки на другие объекты.

Объекты взаимодействуют между собой с помощью сообщений. Принимая сообщение, объект реагирует на него некоторым действием.

Действия, которые может выполнять объект, называются методами объекта. *Метод* – это предоставляемый сервис.

Совокупность методов представляет собой интерфейс объекта. Некоторые атрибуты объекта могут быть доступны извне через его методы, другие предназначены только для внутреннего пользования. Также и методы могут быть доступны извне, могут быть внутренними. В этом случае они в интерфейс объекта не входят.

Рассмотрим время жизни объектов.

В любой системе объекты создаются и уничтожаются через какое-то время. По мере выполнения программы объекты могут загружаться в оперативную память или стираться из нее. Время жизни объектов при таком подходе определяется временем жизни программы. Когда создаются объекты, в какой последовательности, – определяет разработчик. Он задает их создание либо на стадии разработки, либо на стадии выполнения.

Объекты создаются либо как объявляемые переменные, которые в момент запуска программы создаются компилятором, либо динамически.

При динамическом создании объектов на стадии выполнения программа обращается к неким механизмам создания объектов, в каждом языке реализованным по-разному.

Уничтожаться объекты также могут по-разному: либо явно, либо автоматически, после того как они перестают быть кому-либо нужны. Если все пути обращения к объекту уничтожены, то объект становится недоступным и уничтожается (*уничтожение по достижимости*).

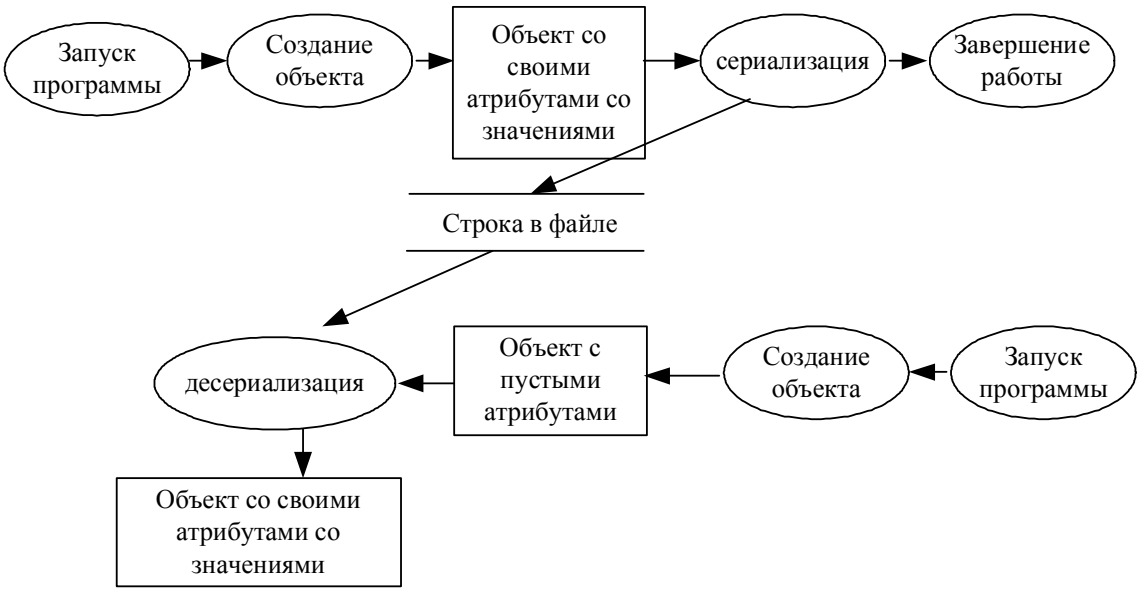


Рис. 38. Метод сериализации

Еще один подход к созданию объектов – создавать и хранить их не в оперативной памяти, как, например, в объектно-ориентированной базе данных или на диске. При запуске программы она берет эти объекты в их последнем сохраненном состоянии. Для того, чтобы сохранить это состояние, применяется метод сериализации (рис.38).

Объекты можно сохранять также в базах данных. При хранении объекта в БД объект при запуске программы он не создается заново, она обращается именно к одному и тому же объекту с его уникальным идентификатором.

Лекция 17

Классы

17.1 Понятие класса

В любой системе, как правило, функционирует достаточно большое количество объектов, многие из которых похожи друг на друга – однотипны.

Однотипные объекты объединяются в классы.

Класс – это шаблон создания объектов.

Все объекты одного и того же класса обладают одинаковым интерфейсом и реализуют его одинаково. Их отличие может быть только в текущем состоянии. Индивидуальные объекты – это экземпляры класса.

Экземпляр – это объект, принадлежащий некоторому классу.

Классы считаются одинаковыми, если у них совпадают все атрибуты, все интерфейсы и реализация этих интерфейсов, т. е. они одинаково реагируют на одни и те же сообщения, их поведение одинаково.

Обозначение классов встречается в следующем виде (рис.39):

Обозначение Румбаха и общеиспользуемое в средствах преоктирования	Обозначение Intel
	

Рис. 39. Обозначение класса

Ранее отмечалось, что состояние объектов характеризуется значениями его атрибутов, которые могут быть как простыми величинами, так и сложными объектами. Таким образом, в классе, по которому создается объект, также имеются атрибуты, присущие объекту и методы, которые объект реализует.

Атрибуты класса могут включать:

- признак видимости;
- имя;
- тип;
- значение по умолчанию.

Записывается последовательно: <признак><имя>:<тип>=<значение по умолчанию>.

Признак видимости может принимать следующие значения:

- общий (public) – атрибут доступен для всех классов;
- защищенный (protected) – атрибут доступен только для друзей (в одном пакете) и для подклассов класса;
- секретный (private) – атрибут доступен только внутри текущего класса.

Атрибуты у экземпляров классов отличаются друг от друга значениями.

Операции – методы класса (или процессы, реализуемые классом). Также, как и у атрибутов, они могут иметь при описании классов следующие характеристики:

- признак видимости – также как и у атрибутов;
- имя – символьная строка;
- список параметров – это необязательные аргументы, передаваемые данной операции (методу);
- тип выражения – определяется конкретным языком программирования.

Записываются <признак видимости><имя> (<список параметров>):<тип выражения, возвращающего значение>.

Например, void (в Си++) – тип выражения, который показывает, что метод ничего не возвращает.

Важнейшее свойство классов – наследование.

Наследование – это отношение между классами, при котором один класс разделяет структуру или поведение одного или нескольких других

классов. Существующий класс называется при этом суперклассом, а производный подклассом или производным классом.

Механизм наследования позволяет выделить общие части классов. Каким образом проводить наследование и создавать иерархию классов, какие классы выделять, решает сам разработчик.

Иерархия классов – это описание отношений наследования между классами.

Некоторые классы, которые просто объединяют общие характеристики других классов, но по которым невозможно создать объекты, называются *абстрактными*.

Классы, экземпляры которых могут существовать в системе, называются *конкретными*. Если класс наследуют методы и атрибуты одного класса – это *простое наследование*, если нескольких – *множественное*. Множественное наследование позволяет объединять характеристики разных классов в одном. Следует учитывать, что не все объектно-ориентированные языки позволяют осуществлять множественное наследование (Smalltalk, Java не поддерживают).

При обращении к классам используется свойство полиморфизма.

Полиморфизм – механизм, позволяющий скрывать различные реализации за общим интерфейсом. Объекты разных классов, выведенные из базового и имеющие похожий интерфейс, могут по-разному реагировать на одно и тоже сообщение (ограниченный полиморфизм). Если независимо от класса объекта он может обработать какое-либо сообщение – это неограниченный полиморфизм. Полиморфизм сводится к разной реакции разных классов или объектов на одно и то же сообщение. Способ реакции описывается в каждом классе, выведенном из базового, индивидуально.

При неограниченном полиморфизме все объекты независимо от класса одинаково реагируют на одно и то же сообщение (например, метод, приводящий значение объекта к строковому виду).

17.2 Виды отношений между классами

Существуют следующие основные отношения между классами:

1) ассоциация;
2) наследование;
3) использование;
4) агрегация.

Ассоциация – двухсторонняя связь между объектами разных классов. *Например*, имеется объект преподаватель и объект студент. Студент может выбрать преподавателя на дипломное руководство, а преподаватель студента. Между ними устанавливается связь – ассоциация «руководит – имеет руководителя». Если связь разрывается, студент меняет руководителя или руководитель студента, то их внутренние атрибуты изменятся, т. е. при изменении ассоциации меняются атрибуты обоих объектов: студент удаляется из списка

преподавателя, а у студента преподаватель исключается из дипломного руководителя. Ассоциации могут иметь множественность 1–1, 1–m, n–m. Аналог – связи в ER-модели базы данных.

Вообще для ассоциаций характерно указание ее мощности:

- 1 – в точности одна связь;
- N – неограниченное число;
- 0..N – ноль или больше;
- 1..N – одна или больше;
- 3..7 – указанный интервал.

Мощность ставится на конце линии ассоциации.

Наследование – это когда один объект наследует все или часть атрибутов и методов другого или других объектов.

Например, существует объект Form – форма, стандартное серое окно, с помощью которого программируют интерфейс пользователя. Мы можем создать свою форму на базе объекта Form. Она унаследует все методы и атрибуты родительской – наследование, но при этом мы можем их переназначить в своей форме, добавить новые методы и атрибуты, изменить ей цвет, размер. Или имеется стандартное линейное уравнение в общем виде: $y=kx+b$. Уравнение, в котором элементам k и b присвоены конкретные значения, будет наследовать все свойства и поведение стандартного линейного уравнения (родителя), но при этом будет представлять конкретную реализацию.

Использование – вызов методов другого класса. Часто встречается в программах. Например, обращение к методам другого класса, которые выполняют какие-то стандартные действия.

Например, вызов метода «определение длины слова» из класса String (Строка) в Java. Программирование в ООЯ заключается в знании стандартных классов каждого языка и методов, присущих каждому классу. Это равнозначно тому, что на каждую болезнь есть лекарство – главное, знать – где применить и что.

Агрегация – это когда один класс включает другой в качестве атрибута (объект–контейнер, объект–атрибут, целое –часть).

Например, автомобиль состоит из двигателя, кузова, шасси. Агрегация в отличие от связей показывает отношение целого и его части. Агрегация может рассматриваться как специализированный случай ассоциации. Агрегация может предусматривать физическое включение (самолет состоит из крыльев, шасси, двигателя), а может и исключать его (акционер владеет акциями, но акции могут существовать и без него).

Объект, являющийся частью другого объекта (агрегата), имеет с ним связь, через которую агрегат может посылать ему сообщения.

Различают агрегацию:

1) по ссылке – целое содержит указатель или ссылку на часть (например, профсоюз имеет членов, но не владеет ими);

2) по значению – целое физически содержит часть (запись о человеке физически содержит его ФИО, адрес, возраст).

При физическом содержании части в агрегате – она создается и уничтожается вместе с агрегатом (время жизни части и целого совпадает).

При ссылке на часть времена жизни агрегата и его части независимы. Агрегация по ссылке поддерживается для ООЯ работающих с указателями.

Также важно понятие *метаданные* – это данные о данных, которые описывают структуру классов и объектов. Наличие метаданных позволяет опрашивать объект, какие у него есть методы и атрибуты. Метаданные могут храниться в виде объектно-ориентированных баз данных.

Обозначения связей, которые рассмотрены выше, выглядят следующим образом (таблица 4):

Таблица 4

	Обозначение Intel	UML Rational Rose
Ассоциация		
Наследование		
Агрегация по ссылке		
Агрегация по значению		
Использование		

Пример описания классов и отношения между ними на Java (рис. 40):

Здесь изображено два класса – record (суперкласс, позволяющий работать с набором записей из базы данных) и database (класс – база данных – наследующий переменную RS и метод next() у класса record).

Класс record имеет две переменных и два метода. При этом переменная lang – это объект, создаваемый на базе класса language – пример включения (агрегация).

Метод next() позволяет переходить к следующей записи в наборе, метод getRecordCount() позволяет получать количество записей в наборе.

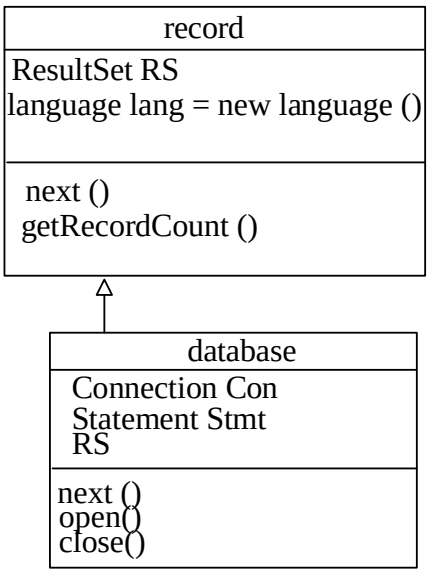


Рис. 40. Пример простого наследования классов

Класс database наследует переменную RS и метод next(). Содержание переменной и поведение метода будут однозначно такие же, как у record. Повторно их описывать нет необходимости.

17.3 Классы и базы данных

По многим причинам использование реляционных структур в объектном программировании имеет широкое применение вследствие ее популярности. Поэтому очень часто встает задача построения объектно-ориентированной оболочки над реляционной схемой.

Для этого были разработаны определенные правила, сводящиеся к следующему:

- каждый класс отображается в одну или несколько таблиц;
- каждое отношение многие-ко-многим отображается в отдельную таблицу;
- каждое отношение один-ко-многим выносится в отдельную таблицу или соотносится с внешним ключом.

При наличии между таблицами отношения многие-ко-многим, оно показывается на диаграммах в виде привязки атрибутов. При этом привязывается только атрибут, объединяющий записи из различных таблиц.

Например, имеем реляционную схему (рис. 41):

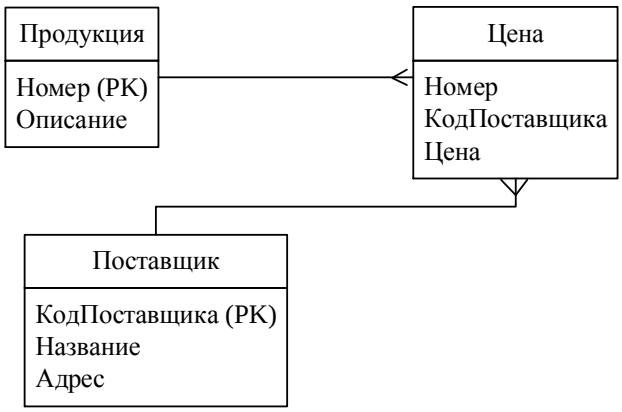


Рис. 41. Пример реляционной схемы

Для этой схемы диаграмма классов будет выглядеть следующим образом (рис.42):

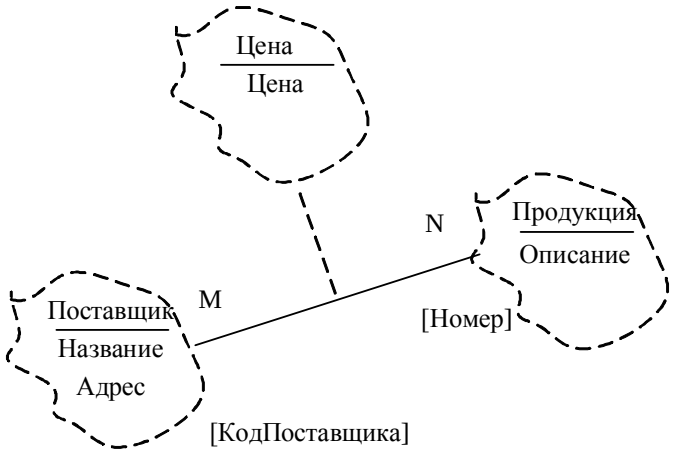


Рис. 42. Соответствующая реляционной схеме объектная надстройка

Лекция 18

Объектно-ориентированный анализ и проектирование ООА/П

18.1 Цель объектного анализа и проектирования

Объектно-ориентированный анализ и проектирование ООА/П – это поуровневый спуск от концептуальной модели использования системы к логической, а затем к физической реализации.

Структура и последовательность создания программных систем в самом общем виде приведена на рис. 43.

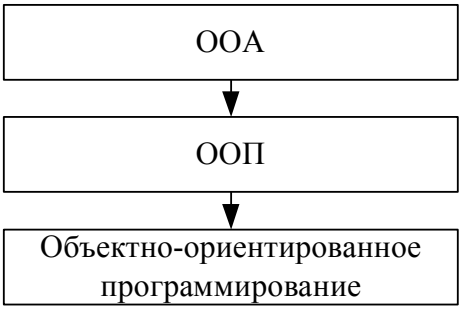


Рис. 43. Общий вид создания объектных программных систем

Объектный анализ с точки зрения ООА/П сводится к исследованию объектов предметной области. Анализ включает написание ТЗ и построение моделей предметной области (концептуальной модели), приближающих разработчика к решению задачи. На данном этапе определяются названия объектов, их логические атрибуты, и иногда – статические связи между объектами. Таким образом, создается модель предметной области. Исследование объектов и построение модели показано на рис. 44.

Рис. 44. Создание модели предметной области

Каждый объект в системе представляется в виде концептуального класса (прямоугольник, внутри имя объекта).

На стадии проектирования, как правило, осуществляется выделение объектов, ответственных за управление, интерфейс. Объекты представляются в виде иерархии классов, которые в конечном счете будут преобразованы в программный код в заданном языке реализации. То есть в процессе ООП устанавливаются динамические отношения между объектами сообщения, которыми они обмениваются, определяются программные объекты, способы их взаимодействия (диаграммы взаимодействия) для выполнения системных требований.

ООА/П тесно связан с таким этапом, как анализ требований. Анализ требований включает в себя описание прецедентов. По сути, с этого начинается анализ и построение любой программной системы. В общем же виде

ООА/П в совокупности с фазой анализа требований предстают следующим образом (рис. 45).

Рис. 45. Объектно-ориентированный анализ и проектирование

Анализ прецедентов связан в первую очередь с тем, что у потребителей и конечных пользователей системы есть свои задачи (потребности), которые должна обеспечить компьютерная система.

Прецедент – это механизм упрощения формулировки требований всех заинтересованных лиц, это рассказы об использовании системы в процессе решения поставленных задач. Прецедент (use-case) означает случай использования.

Для описания прецедента применяются такие понятия, как Исполнитель (актер) и Сценарий.

Исполнитель – это некоторая сущность, обладающая поведением и являющаяся инициатором какого-либо действия в системе. Это может быть человек, организация или другая компьютерная система.

Сценарий – это специальная последовательность действий или взаимодействий между исполнителями и системой. Может использоваться как термин – *экземпляр прецедента.*

Для графической поддержки перечисленных этапов в создании программного обеспечения был разработан специальный язык UML.

UML – это язык для определения, визуализации, конструирования и документирования артефактов программных систем. UML может применяться также и для описания других не программных систем и моделирования экономических процессов.

UML – это по сути стандарт, который появился в 1994 г. Создатели Гради Буч, Джим Румбаха, Айвар Якобсон. С 1997 г. UML был принят группой промышленных стандартов OMG в качестве стандартного языка моделирования, в настоящий момент развивается в редакции OMG UML.

18.2 Диаграммы UML

Всего существует 9 основных диаграмм:

- диаграммы классов (Class diagram);
- диаграммы объектов (Object diagram);
- диаграммы прецедентов (Use case diagram);
- диаграммы последовательностей (Sequence diagram);
- диаграммы кооперации или сотрудничества (Collaboration diagram);
- диаграммы состояний (State diagram);
- диаграммы деятельности (Activity diagram);
- диаграммы компонентов (Component diagram);
- диаграммы развертывания (Deployment diagram).

Для описания системы наиболее часто используется следующий минимальный набор диаграмм:

- вложенная диаграмма вариантов использования (use-case);
- диаграммы взаимодействия (interaction), которые включают: диаграммы сотрудничества (или взаимодействия объектов (collaboration) и диаграммы последовательности действий (sequence);
- диаграмма классов (class);
- диаграмма переходов состояний (statechart).

Данный перечень диаграмм основной. Помимо них существуют, как сказано выше, еще несколько диаграмм, описывающих систему с разных точек зрения (компонентная диаграмма, диаграмма топологии, диаграмма развертывания). Однако на практике наиболее используемыми остаются use-case диаграмма и диаграмма классов.

Система может рассматриваться со статической точки зрения – диаграмма классов, и с динамической – диаграммы взаимодействия объектов и диаграммы последовательности действий.

Диаграмма переходов состояний предназначена для описания поведения конкретного класса в системе. Представляет собой модель конечного автомата, в котором вершины – это возможные состояния класса, и заданы условия перехода из одного состояния в другое.

Физическая модель системы описывается компонентной диаграммой – распределение реализации классов по модулям и диаграммой топологии.

После построения диаграммы классов на ее основе может быть сгенерирован программный код на заданном языке программирования. Этот код можно уточнить, вручную внеся туда изменения. После внесения изменений путем обратного проектирования на основе измененного кода будет сгенерирована новая уточненная модель. Повторение этой процедуры – итерационное моделирование.

Один из опытных проектировщиков предлагает следующий механизм проектирования объектной системы:

«Используем «Use case diagram» для отображения списка операций, которые должна выполнять наша система; иначе говоря, это требования к системе.

Каждый «Use case» – это некоторый процесс (последовательность действий), поэтому мы должны использовать «Sequence diagram» для его детализации. На этой диаграмме мы отображаем объекты из предметной области (объекты, участвующие в бизнес-процессе); таким образом мы получаем экземпляры некоторых классов и их взаимодействие. «Sequence diagram» отображает сам процесс, статическая картина взаимодействия объектов отображается с помощью «Class diagram». Переходим к «Class diagram», на которой изображаются классы нашей ИС. Далее классы объединяются в компоненты, которые отображаются на «Component diagram», где показывается зависимость компонентов между собой. На «Deployment diagram» отображается размещение этих компонентов по компьютерам (узлам сети) для проектируемой ИС».

Лекция 19

Основные виды диаграмм

19.1 Диаграмма прецедентов USE-CASE диаграмма

Концептуальная модель выражается в виде диаграммы прецедентов (вариантов использования). Для составления вариантов использования прецедентов рекомендуется рассмотреть все внешние по отношению к системе события, на которые система должна реагировать. Существуют различные форматы описания прецедентов. В любом случае это – некоторый повествовательный документ о поведении системы. В этом документе должны быть отражены такие моменты, как:

- основные исполнители, заинтересованные лица и их требования;
- результаты работы системы;
- основной процесс в системе (успешный);
- альтернативы – отклонения от основного процесса;
- специальные требования;
- список технологий и типов данных.

Один из шаблонов описания доступен на www.usecases.org.

Одним из распространенных является формат в виде двух колонок от Вирфс-Брок. В первой – действия исполнителя, во второй – отклик системы.

Диаграмма вариантов использования характеризуется рядом свойств:

- вариант использования охватывает некоторую очевидную пользователю функцию;
- вариант может быть небольшим (системные прецеденты) или крупным (бизнес-прецедент).

Основные компоненты диаграммы представлены на рис.46.

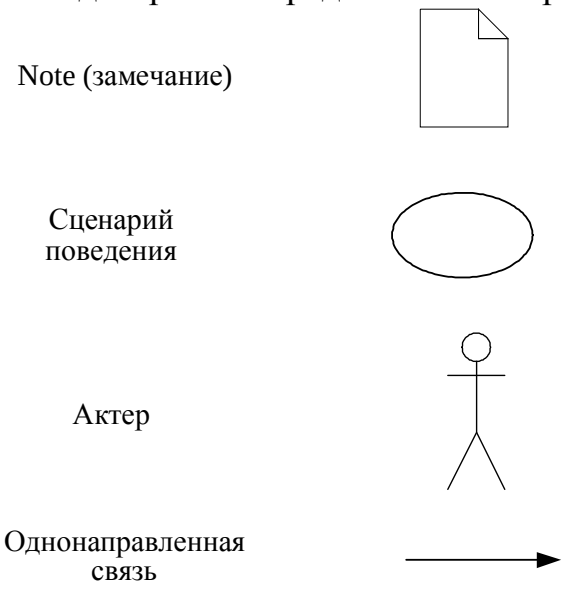


Рис. 46. Элементы Use-Case диаграммы

Сценарии поведения могут отображать:

- образцы поведения для отдельных объектов в системе;
- последовательность связанных транзакций;
- получение некоторой информации объектами.

Актер создает действующих лиц в системе:

- взаимодействует с системой или использует систему;
- передает или принимает информацию в системе;
- является внешним по отношению к системе.

Однонаправленная связь соединяет актеров и варианты использования.

Пример (рис. 47):

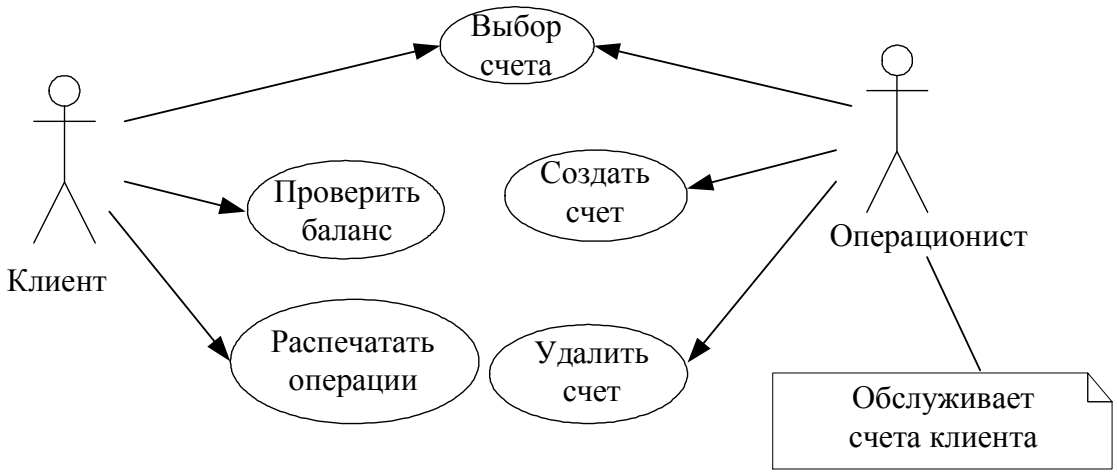


Рис. 47. Диаграмма Use-case

Варианты использования могут быть связаны друг с другом двумя видами связей: связями расширения и использования (рис. 48).

Вариант использования В *расширяет (extends)* вариант использования А, если В определяет дополнительные действия, которые вставляются в некоторое место в сценарии работы А.

Основная разница между расширением и использованием состоит в целях двух находящихся в таком отношении вариантов использования. Если их цели по большому счету одинаковы, пользователь ожидает от них примерно одинаковых результатов, то один из них расширяет другой, если цели существенно различны – то использует.

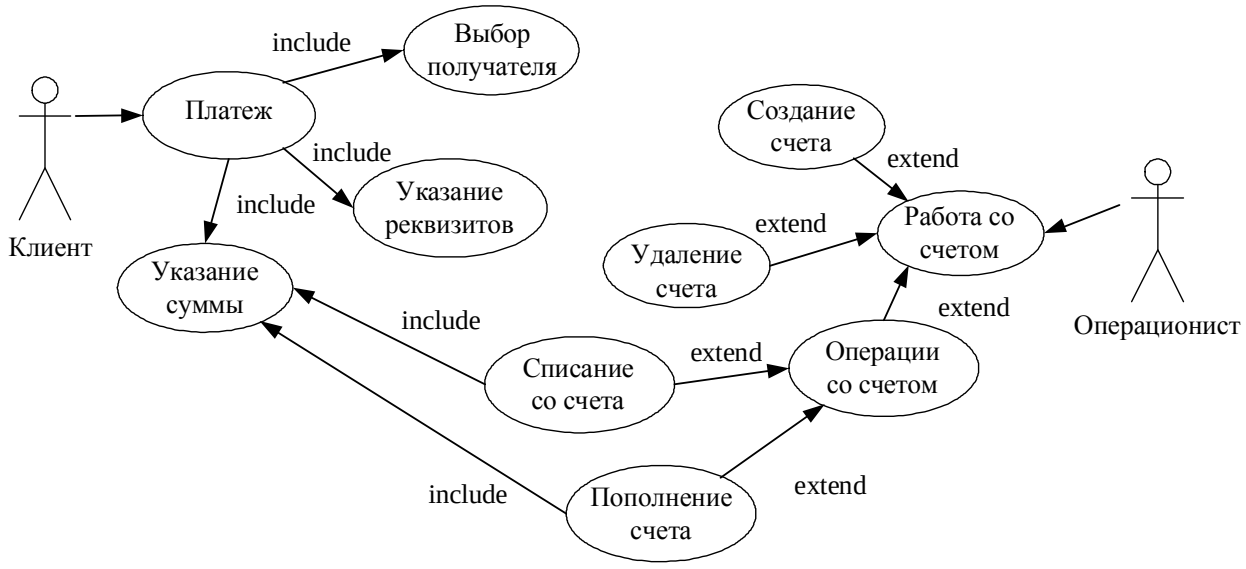


Рис. 48. Доработанная диаграмма использования

19.2 Диаграммы взаимодействия и сотрудничества

Взаимодействие объектов в системе происходит посредством приема и передачи сообщений между объектами. Это диаграммы взаимодействия объектов и диаграммы последовательности действий. Как правило, одна диа-

грамма отображает некоторый процесс в системе (сценарий поведения) или последовательность действий.

Диаграммы последовательности отражает последовательность передачи сообщений между объектами.

Основные элементы диаграммы представлены на рис.49:

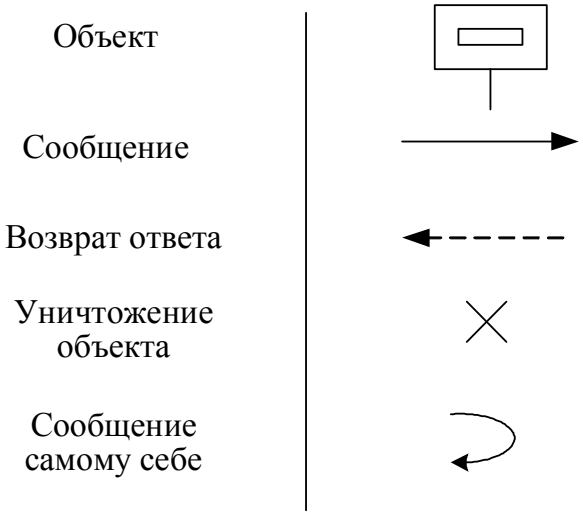


Рис. 49. Элементы диаграммы последовательности

Пунктирная линия под объектом называется линией жизни – фрагмент жизненного цикла в процессе взаимодействия. Каждое сообщение может иметь имя, аргументы или другую управляющую информацию. Прямоугольники на линии жизни – активизации – показывают, когда метод становится активным (рис. 50).

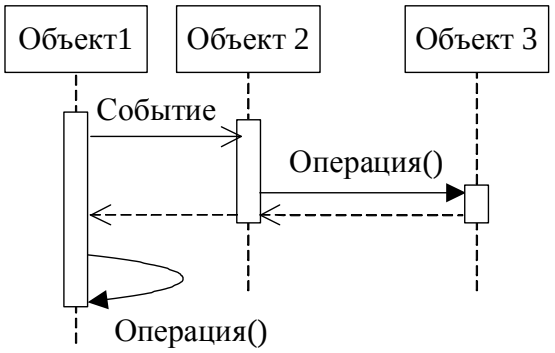


Рис. 50. Пример диаграммы последовательности действий

Другой вид диаграмм – кооперативная диаграмма. Экземпляры объектов соединены линиями, обозначающими сообщения, обмен которыми происходит в рамках данного варианта использования. Все сообщения представляются в компактном виде. Данная диаграмма строится на базе предыдущей и показывает то же самое, но в более компактном виде.

19.3 Диаграммы состояний

Диаграммы состояний описывают поведение системы, определяют ее возможные состояния, а также условия смены состояний.

Рекомендуют строить диаграммы состояний для классов пользовательского интерфейса и управляющих объектов. Пример приведен на рис.52.

Основные элементы диаграммы представлены на рис.51.

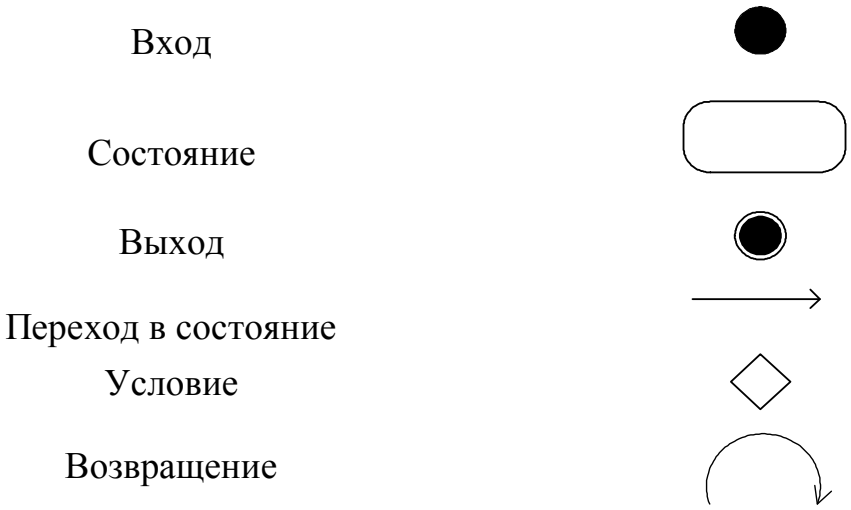


Рис. 51. Элементы диаграммы состояний

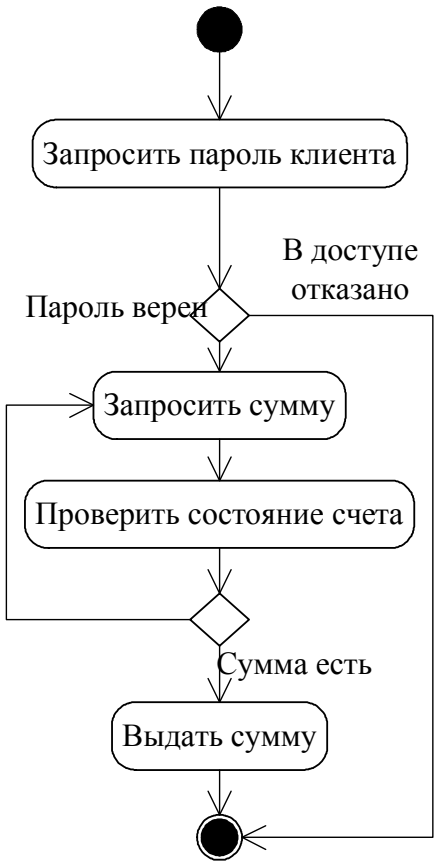


Рис. 52. Пример диаграммы состояний

19.4 Диаграмма классов (этап проектирования)

Диаграмма классов – центральное звено методологии проектирования. На ее базе строится код приложения.

На стадии анализа диаграмма классов используется, чтобы выделить общие роли и обязанности сущностей, обеспечивающих требуемое поведение системы. На стадии проектирования диаграмма классов передает структуру классов, формирующих архитектуру системы. При этом система рассматривается со статической точки зрения.

Основные элементы диаграммы классов рассматривались выше в разделе «Виды связей» в ООА/П.

В проекте все классы уникальны. Каждый класс – это самодостаточная структура, автономная и самоуправляемая, которая может существовать даже после завершения программы.

Для получения классов в программе надо разбить всю задачу на управляемые компоненты, выполняющие отдельные функции.

На диаграмме классов показываются также атрибуты классов и операции. Состояние класса определяется значениями его атрибутов.

В UML на диаграмме классов направленные связи указываются стрелками. Если стрелки нет, то это трактуется: направление неизвестно или связь двунаправлена.

Все основные обозначения отношений между классами рассматривались в лекции 17.

Лекция 20

Архитектура информационных систем

20.1 Многократное использование программных систем

Одно из главных преимуществ объектно-ориентированного программирования является возможность многократного использования разработанного программного обеспечения. Для этой цели применяют библиотеки. В процедурном программировании – это библиотеки функций, в объектно-ориентированном – это библиотеки классов.

Библиотеки функций ориентированы на язык программирования, либо для класса задач. В каждом языке в пакете программирования поставляется набор библиотек с наиболее часто используемыми функциями. Например, функции работы со строками, математические функции, функции преобразования и т.д.

Функция реализует механизм обработки данных, и результат зависит только от входных параметров.

Применение функций весьма ограничено, перенастроить их под конкретные задачи сложно. В этом плане приспособление классов и объектов к задачам намного гибче, поскольку объектно-ориентированный подход позво-

ляет создавать новые классы на основе тех, которые поставляются в библиотеках, с помощью механизма наследования.

В программе с базовых классов создаются конкретные и их приспосабливают под решение текущей задачи. Пример базовых классов библиотека MFC для языка СИ++.

Подключение библиотек также определяется средой программирования.

Например, #include в Си++ или import в Java.

Однако использование библиотек, несмотря на то что оно повышает эффективность разработки программ, требует определенных усилий для их изучения и понимания их устройства. Поэтому появилась концепция компонент.

Компонента – это программный модуль или объект, который готов для использования в качестве составного блока программы и которым можно визуально манипулировать во время разработки.

Компонента объединяет состояние и интерфейс. Компонента должна быть нейтральна к языку программирования, т. е. ее можно использовать в программе на любом языке, который поддерживает компонентный подход.

Компонентное представление используется чаще для построения графического интерфейса пользователя. Однако возможности компоненты не ограничиваются графическим интерфейсом. Она может решать и вычислительные функции.

Наиболее известные системы построения локальных компонент:

1. COM/ActiveX Microsoft.
2. CORBA от OMG.
3. Технология JavaBeans.

20.2 Архитектура информационных систем

Большинство проектируемых систем являются децентрализованными и строятся по принципу «клиент-серверных» технологий. В их основе лежат две основные идеи:

1. Общие для всех пользователей данные на одном или нескольких серверах.
2. Много пользователей на различных машинах, обрабатывающих совместно общие данные.

Поэтому такие системы считаются распределенными только в отношении пользователей (не относятся к по-настоящему распределенным системам) – многопользовательские системы. В основе таких систем лежит СУБД. Как правило, говоря о системах структурного подхода, выделяют следующие компоненты архитектуры таких систем:

1. Компонент представления или интерфейс (реализует функцию ввода и отображения данных). Еще его называют логикой представления.
2. Прикладной компонент (набор запросов, событий, процедур, других вычислительных функций, реализующих предназначение автоматизиро-

ванной информационной системы в конкретной предметной области). Иначе «бизнес-логика».

3. Компонент доступа к данным (реализует функции хранения, извлечения, физического обновления и изменения данных).

По мнению некоторых, компонент доступа к данным следует разделить на логику базы данных – язык SQL – и непосредственно механизмы работы с данными на низком уровне. Но суть от этого не меняется.

Говоря об архитектуре объектно-ориентированной информационной системы, включающей графический интерфейс и взаимодействие с базой данных, выделяют:

1. Интерфейс пользователя (графические элементы и диалоговые окна).
2. Уровень логики приложения или объектов предметной области (включает программное представление объектов предметной области, реализующих требования к системе).
3. Технические службы (объекты и подсистемы общего назначения, которые обеспечивают выполнение вспомогательных функций, например, обмен информацией с базой данных или регистрацию событий).

Объектно-ориентированный анализ и проектирование (ООА/П) относятся, как правило, к уровням логики приложения и технических служб.

При этом, говоря о понятии анализ, необходимо уточнять анализ требований (т.е. исследование требований к системе) и объектный анализ (исследование объектов системы), а говоря о проектировании, необходимо уточнить, об объектном проектировании или проектировании базы данных идет речь.

Тем не менее, из вышесказанного выделяется однозначно три уровня – *трехуровневая архитектура* (логическое деление приложения). Впервые она была описана в 1970 г.г. При такой архитектуре приложение делится по вертикали на три уровня:

1. Логика представления.
2. Бизнес-логика.
3. Механизмы работы с базами данных.

В данной архитектуре логика приложений выносится на отдельный уровень. Уровень интерфейса относительно независим от решения основных задач приложения и обеспечивает функции диалога с пользователем. Его окна и страницы отправляют запросы на средний уровень, а тот в свою очередь отправляет их дальше на нижний уровень хранения данных.

Существует также *двухуровневая архитектура.* В данной архитектуре логика приложений включается в определения окон графического интерфейса.

Объекты уровня графического интерфейса считывают и записывают информацию непосредственно в базу данных. Отдельного уровня логики приложений не существует.

Недостаток двухуровневой архитектуры – невозможность представления логики приложения в виде отдельных компонентов с возможностью их дальнейшего повторного использования.

Преимущество двухуровневой архитектуры: как правило, быстрота разработки. Для простых приложений по обработке данных это бывает очень удобно!

При проектировании систем в настоящее время основной задачей архитектора системы является оптимальное распределение описанных уровней системы по узлам обработки в сети.

Для проектирования систем часто рекомендуют:

- компонент логики представления размещать там же, где и терминал ввода-вывода, т. е. на компьютере конечного пользователя;
- имеет смысл (при мощных компьютерах клиента) размещать там же и часть бизнес-логики;
- при организации пользователей в группы и совместном использовании ими данных, механизм работы с данными и часть бизнес-логики, которые являются общими для всех, размещать на сервере.

Способ размещения компонент выливается в модели доступа к данным.

В зависимости от реализации и расположения в системе трех компонент структуры можно выделить следующие модели доступа к данным в этих системах:

1. Модель файлового сервера (File Server – FS).
2. Модель удаленного доступа к данным (Remote Data Access – RDA).
3. Модель доступа из браузера. Используется в технологиях Internet/Intranet.
4. Модель сервера базы данных (Data Base Server – DBS).
5. Модель сервера приложений (Application Server – AS).

1. Модель файлового сервера характеризуется общим способом взаимодействия компьютеров в локальной сети.

Один из компьютеров выделяется и определяется файловым сервером, т. е. общим хранилищем данных. Все три основных компонента архитектуры (исполняемое приложение) располагаются на клиентской машине.

На FS хранятся только файлы базы данных и некоторые технологические файлы. Операторы обращения к данным обрабатываются ядром СУБД, после чего оно организует доступ к файлам базы данных на FS. По сети идут запрос на чтение/запись, промежуточные и результирующие данные в виде копий файлов базы данных, загружаемых в оперативную память клиентских машин. Никакая часть СУБД на сервере не инсталлируется и не размещается!

Достоинства модели: наиболее простая, недорогая, проста в установке, отсутствие высоких требований к серверу.

Недостатки: высокий сетевой трафик, отсутствие безопасности файлов БД.

Пример: FoxPro, Clipper, Paradox.

2. Удаленный доступ к данным основан на специфике размещения и физического манипулирования данными.

Компонент прикладного доступа к данным размещается на сервере системы, является общим для всех рабочих станций и реализуется в виде самостоятельной программной части СУБД, называемой *SQL- сервером*. Функции SQL-сервера заключаются в низкоуровневых операциях по организации, размещению, хранению и манипулированию данными в дисковой памяти сервера.

В файлах БД помещается также системный каталог БД, где хранятся сведения о зарегистрированных клиентах и их правах.

На клиентских машинах хранятся компонент логики представления (интерфейсная часть) и компонент прикладных функций. Пользователь, входя в систему, регистрируется через клиентскую часть на сервере системы и начинает работу с данными.

Операторы обращения к СУБД не выполняются на рабочих станциях, а пересылаются для обработки на сервер. Результаты обработки запросов отсылаются обратно пользователю на рабочую станцию. Доступ к информационным ресурсам обеспечивается, например, вызовами функций специальной библиотеки (если имеется соответствующий интерфейс прикладного программирования – API). RDA уменьшает загрузку сети, так как по ней передаются от клиента к серверу не запросы на ввод-вывод (как в системах с файловым сервером), а запросы на языке SQL, и их объем существенно меньше.

Достоинство RDA-модели заключается в унификации интерфейса «клиент-сервер» в виде языка SQL.

Недостаток: трудности с администрированием в этой модели, высокий трафик.

3. Модель доступа через Интернет-браузеры (технология Internet-Intranet) может рассматриваться как разновидность RDA.

В настоящее время для доступа к данным в системах, разработанных по данной технологии, используются:

- CGI- или ISAPI-программы, Java-сервлеты; (Для написания CGI-программ применяются языки: Perl, C, TCL. Для написания сервлетов используется Java);
- VBScript, JavaScript-программы (используют языки клиентских скриптов);
- ASP-технология, PHP-технология, JSP-технология (представляют собой также серверные скрипты). *Сервлет* – это программа-посредник между запросами пользователя и базами данных или приложениями на WEB-сервере.

Технология серверных приложений: по запросу WEB-браузера из WEB-сервера читается HTML форма (программа, которая содержит описание полей ввода и имя программы для их обработки). После заполнения полей и нажатия кнопки типа Submit браузер пересылает на сервер значения полей ввода и имя программы для их обработки, после чего сервер запускает эту

программу. Она с помощью запроса получает данные из локальной или удаленной базы данных и с помощью оператора print генерирует новую HTML-страницу. Пользователь видит результаты запроса.

Технология клиентских приложений: браузер при интерпретации страницы доступа и нахождении тега Applet посылает запрос к WEB-серверу на чтение Java-программы апплета. Апплет загружается на машину клиента и выполняется там Java-машиной (JVM). В процессе выполнения на рабочей станции апплет может обращаться к удаленным базам данных через специальный интерфейс доступа к данным JDBC.

Достоинство такой модели: разнообразие клиентских и серверных платформ, распределенная обработка данных, мощная поддержка вычислительных алгоритмов через языки высокого уровня.

Основной недостаток: необходимость средств защиты от несанкционированного доступа.

4. Модель сервера базы данных DBS. На рабочих станциях выполняется приложение, включающее диалог с пользователем. Ядро СУБД общее для всех пользователей хранится на сервере. Операторы обращения к СУБД, закодированные в транзакции, пересылаются для обработки на сервер. Ядро СУБД транслирует запрос, выполняет его и возвращает пользователю результаты обработки оператора.

На сервере могут запускаться также триггеры и хранимые процедуры. *Триггер* – программа, которая позволяет поддерживать целостность таблиц базы данных. Модель DBS поддерживаются следующими СУБД (Oracle, Sybase, Informix и пр.), которые имеют встроенные языки программирования.

Преимущества: высокая производительность таких систем, поскольку по сети передаются только SQL-запросы и результаты их выполнения. Запросы выполняются на высокоскоростных серверах; поддерживается распределенная обработка; большое число сервисных продуктов, облегчающих разработку приложений и создание распределенных систем.

Недостаток: дороговизна, сложность в освоении; высокоскоростные серверы и сети.

На практике часто используются смешанные модели, когда поддержка целостности базы данных и некоторые простейшие прикладные функции выполняются хранимыми процедурами (DBS-модель), а более сложные функции реализуются непосредственно в прикладной программе, которая работает на компьютере-клиенте (RDA-модель).

5. Модель сервера приложений позволяет разгрузить рабочие станции (тонкий клиент) и разнести требования к вычислительным ресурсам в отношении быстродействия и памяти по разным элементам системы (хотя допускается и не разносить). На клиентской машине располагается только компонент представления (интерфейсная часть). Компонент прикладных функций (сервер приложений) и компонент прикладного доступа к данным (СУБД) размещаются на сервере, под которым понимается компьютер, где функционирует СУБД и/или сервер приложений. Сервер приложений представляет

собой специальное средство, называемое менеджером транзакций (набор SQL запросов называется транзакцией), и он, в отличие от клиентов, управляет формированием транзакций к серверу БД. Это оболочка, в рамках которой выполняются прикладные программы (сервисы), написанные на языке высокого уровня. В их числе программы, обеспечивающие информационный обмен с СУБД, программы запускающие приложения на сервере и т.д.

По уровням архитектуры представленные модели делятся следующим образом:

RDA-модель и DBS-модель относят к двухуровневым системам. AS-модель относят к трехуровневым системам.

Библиографический список

1. *ГОСТ* 34.601-90. Стадии проектирования АС.
2. *ГОСТ* 34.602-90. Техническое задание на проектирование АС.
3. *ГОСТ* 34.603-90. Виды приемочных испытаний АС.
4. *ГОСТ* 34.003-90. Информационная технология. Комплекс стандартов на автоматизированные системы. Термины и определения.
5. *ГОСТ* РИСО/МЭК 12207-99. Информационная технология. Процессы жизненного цикла программных средств.
6. *ОРММ ИСЖТ* 5.03-00. Процессы жизненного цикла информационных систем и программных средств.
7. *Калянов Г. Н.* CASE-технологии : Консалтинг в автоматизации бизнес-процессов. – 2-е изд., перераб. и доп. – М. : Горячая линия – Телеком, 2000. – 320 с.
8. *ГОСТ* 24.104-85. Автоматизированные системы управления. Общие требования.
9. *РД* 50-34.698-90. Требования к содержанию документов на АС.
10. *Гайдамакин Н. А.* Автоматизированные информационные системы, базы и банки данных. Вводный курс : учеб. пособие. – М. : Гелиос АРВ, 2002. – 368 с., ил.
11. *Уткин В.Б.* Информационные системы в экономике: учеб. для студ. высш. учеб. заведений. – М. : Издательский центр «Академия», 2004. – 288 с.
12. *Крег Ларман*. Применение UML и шаблонов проектирования. – 2-е изд. / пер. с англ. – М. : Издательский дом «Вильямс», 2002. – 624 с.: ил.
13. *Федотова Д.Э., Семенов Ю.Д., Чижик К.Н.* CASE-технологии : практикум. – М. : Горячая линия – Телеком, 2005. – 160 с., ил.
14. *Черемных С.В., Семенов И.О., Ручкин В.С.* Моделирование и анализ систем. IDEF-технологии : практикум. – М. : Финансы и статистика, 2005. – 192 с., ил.
15. *Фридман А.Л.* Основы объектно-ориентированной разработки программных систем. – М. : Финансы и статистика, 2000. – 192 с., ил.
16. *Трофимов С.А.* Case-технологии: практическая работа в Rational Rose. – Изд. 2-е. – М. : Бином-Пресс, 2002. – 288 с., ил.

17. *Орлов С.А.* Технологии разработки программного обеспечения : учебное пособие. – 2-е изд. / – СПб. : Питер, 2003. – 480 с., ил.

18. *ГОСТ РИСО/МЭК* 9126-93. Информационная технология. Оценка программной продукции. Характеристики качества и руководства по их применению.

19. *Буч Г.* Объектно-ориентированный анализ и проектирование с примерами приложений на языке C++. – 2-е изд. / пер. с англ. – М. : Бином, СПб. : «Невский диалект», 1999 . – 560 с., ил.

Учебное издание

Паршин Константин Анатольевич
Паршина Екатерина Валерьевна

ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ

Конспект лекций
для студентов 5 курса очного обучения
и 6 курса заочного обучения
по специальности – 071900 «Информационные системы
(на железнодорожном транспорте)»

Редактор *С.И. Семухина*

Подписано в печать Формат 60×84 1/16

Бумага оффсетная Усл. печ. л.

Тираж 50 экз. Заказ №

Издательство УрГУПС
620034, Екатеринбург, ул. Колмогорова, 66