Основы проектирования ИС

Классификация ИС, определяющая функциональные возможности и особенности построения систем

В настоящее время существует много разнотипных информационных систем (ИС), которые предназначены для решения определенных задач, и, соответственно, отличаются функциональностью, техническими характеристиками, принципами построения, заложенными в них правилами обработки информации и т.д. ИС можно классифицировать по разным признакам. С точки зрения проектирования классификация ИС позволяет отнести создаваемую систему к определенному классу, использовать опыт создания подобных систем, точнее определить требования, характеристики проекта, функциональность, особенности построения. В основе существующей классификации используются наиболее существенные признаки, определяющие функциональные возможности и особенности построения современных систем.

В зависимости от типа и объема решаемых задач, обрабатываемых данных, уровня автоматизации, сфере применения, способов обработки данных, организации функционирования, ИС делятся на ряд классов, которые указаны на рис. 1.1.

Рис. 1.1. Классификация информационных систем

По степени автоматизации информационных процессов предприятия (организации) информационные системы делятся на ручные (неавтоматизированные), автоматические и автоматизированные. Ручные ИС не используют современные аппаратно-программные средства для сбора, обработки, хранения и передачи данных. Все операции выполняются человеком. К таким системам можно отнести библиотечную систему без применения компьютеров. В автоматических ИС все операции по переработке информации выполняются без участия человека. Автоматизированные ИС предполагают участие в процессе обработки информации и человека, и технических средств, причем главная роль в выполнении рутинных операций обработки данных отводится компьютеру. Именно этот класс систем соответствует современному представлению понятия "информационная система".

По типу хранимых данных ИС делятся на фактографические и документальные. Фактографические системы предназначены для хранения и обработки структурированных данных в виде чисел и текстов. Над такими данными можно выполнять различные операции. В документальных системах информация представлена в виде документов, состоящих из наименований, описаний, рефератов и текстов. Поиск по неструктурированным данным осуществляется с использованием семантических признаков. Отобранные документы предоставляются пользователю, а обработка данных в таких системах практически не производится.

В зависимости от характера обработки данных ИС делятся на информационно-поисковые и информационно-решающие. Информационно-поисковые системы производят ввод, систематизацию, хранение, выдачу информации по запросу пользователя без сложных преобразований данных (например, ИС библиотечного обслуживания, резервирования и продажи билетов на транспорте, бронирования мест в гостиницах и пр.). Информационно-решающие системы осуществляют, кроме того, операции переработки информации по *определенному алгоритму*. По характеру использования выходной информации такие *системы принято* делить на управляющие и советующие. Результирующая информация управляющих ИС непосредственно трансформируется в принимаемые человеком решения. Для этих систем характерны задачи расчетного характера и обработка больших объемов данных (например, ИС планирования производства или заказов, бухгалтерского учета). Советующие ИС вырабатывают информацию, которая принимается человеком к сведению и учитывается при формировании управленческих решений, а не инициирует конкретные действия. Эти системы имитируют интеллектуальные процессы обработки знаний, а не данных. (например, *экспертные системы*).

В зависимости от сферы применения различают следующие классы ИС.

Информационные системы организационного управления, которые предназначены для автоматизации функций управленческого персонала предприятий и организаций (гостиниц, учебных учреждений, банков, магазинов и др.). Основными функциями этих систем являются: оперативный контроль и регулирование, оперативный учет и анализ, перспективное и оперативное планирование, бухгалтерский учет, управление сбытом, снабжением и другие экономические и организационные задачи.

ИС управления технологическими процессами (ТП) или (АСУТП)- служат для автоматизации функций производственного персонала по контролю и управлению производственными операциями. В таких системах обычно предусматривается наличие развитых средств измерения параметров технологических процессов (температуры, давления, химического состава и т.п.), процедур контроля допустимости значений параметров и регулирования технологических процессов.

ИС автоматизированного проектирования (САПР) - предназначены для автоматизации функций инженеров-проектировщиков, конструкторов, архитекторов, дизайнеров при создании новой техники или технологии. Основными функциями этих систем являются: инженерные расчеты, создание графической документации (чертежей, схем, планов), создание проектной документации, моделирование проектируемых объектов.

Интегрированные (корпоративные) ИС - используются для автоматизации бизнес-процессов предприятия (организации) и охватывают весь цикл работ от планирования деятельности до сбыта продукции. Они включают в себя ряд модулей (подсистем), работающих в едином информационном пространстве и выполняющих функции поддержки соответствующих направлений деятельности. Типовые задачи, решаемые модулями интегрированной (корпоративной) системы, приведены в таблице 1.1.

Таблица 1.1. Функциональное назначение модулей интегрированной (корпоративной) ИС

Подсистема маркетинга	**Производственные подсистемы**	**Финансовые и учетные подсистемы**	**Подсистема кадров**	**Прочие подсистемы (диспетчера, руководства)**
Исследование рынка и прогнозирование продаж	Планирование объемов работ и разработка календарных планов	Управление портфелем заказов	Анализ и прогнозирование потребности в трудовых ресурсах	Контроль за деятельностью фирмы
Управление	Оперативный контроль	Управление	Ведение архивов	Выявление

продажами	и управление производством	кредитной политикой	записей о персонале	оперативных проблем
Рекомендации по производству новой продукции	*Анализ работы* оборудования	Разработка финансового плана	Анализ и планирование подготовки кадров	Анализ управлен-ческих и стратегичес-ких ситуаций
Анализ и установление цены	Участие в формировании заказов поставщикам	*Финансовый анализ* и прогнозирование		Обеспечение процесса выработки управленчес-ких решений
Учет заказов	Управление запасами	Контроль бюджета, бухгалтерский учет и расчет зарплаты		

Анализ современного состояния рынка ИС показывает устойчивую тенденцию роста спроса на информационные системы организационного управления. Причем спрос продолжает расти именно на интегрированные системы управления. Автоматизация отдельной функции, например, бухгалтерского учета или сбыта готовой продукции, считается уже пройденным этапом для многих предприятий.

В таблице 1.2 приведен перечень наиболее популярных в настоящее время программных продуктов для реализации ИС организационного управления различных классов.

Таблица 1.2. Классификация современного рынка информационных систем

Локальные системы	**Малые интегрированные системы**	**Средние интегрированные системы**	**Большие интегрированные системы (IC)**
БЭСТ Инотек Инфософт Супер-Менеджер Турбо-Бухгалтер Инфо-Бухгалтер	Concorde XAL *Exact* NS-2000 Platinum PRO/*MIS* Scala SunSystems БЭСТ-ПРО Парус Ресурс Эталон	Microsoft-Business Solutions - Navision, Axapta J D Edwards (Robertson & Blums) MFG-Pro (QAD/BMS) SyteLine (СОКАП/SYMIX) 1С-Предприятие БОСС-Корпорация Галактика	*SAP*/R3 (*SAP* AG) Baan (Baan) BPCS (ITS/*SSA*) OEBS (Oracle E-Business Suite)

Существует *классификация ИС* в зависимости от уровня управления, на котором система используется.

Информационная система оперативного уровня - поддерживает исполнителей, обрабатывая данные о сделках и событиях (счета, накладные, зарплата, кредиты, поток сырья и материалов). Информационная система оперативного уровня является связующим звеном между организацией и внешней средой. Задачи, цели, источники информации и алгоритмы обработки на оперативном уровне заранее определены и в высокой степени структурированы.

Информационные системы специалистов - поддерживают работу с данными и знаниями определенной предметной области, повышают продуктивность и производительность работы специалистов

предприятия (организации), например, технологов, проектировщиков, диспетчеров. Основная задача этих информационных систем заключается в информационной и интеллектуальной поддержке в ходе решения производственных задач (интерактивных вычислениях, распознавании возникающих производственных ситуаций, выявлении отклонений в состоянии наблюдаемого объекта).

Информационные системы уровня менеджмента - используются работниками среднего управленческого звена для мониторинга, контроля, принятия решений и администрирования. Основные функции этих информационных систем:

- сравнение текущих показателей с прошлыми;
- составление периодических отчетов за определенное время;
- обеспечение доступа к архивной информации и т.д.

Стратегическая информационная система - обеспечивает поддержку принятия решений по реализации стратегических целей развития организации. Информационные системы стратегического уровня помогают высшему звену управленцев решать слабо *неструктурированные задачи,* осуществлять долгосрочное планирование, выявлять скрытые закономерности в поведении объекта управления на основе обработки больших объемов разнородной информации. Основная задача - сравнение происходящих во внешнем окружении изменений с существующим потенциалом предприятия (организации). Информационные системы стратегического уровня предназначены также для компьютерной поддержки согласования и принятия решений. Эти системы включают мощный аналитический компонент, который позволяет собирать информацию из различных источников и обрабатывать большие объемы данных.

С точки зрения программно-аппаратной реализации можно выделить ряд типовых архитектур ИС. Существуют архитектурные решения, которые основаны на использовании выделенных файл-серверов или серверов баз данных. Существуют также варианты архитектур корпоративных информационных систем, которые базируются на технологии Internet (*Intranet*). Следующая разновидность архитектуры информационной системы основывается на концепции "хранилища данных" (DataWarehouse) - интегрированной *информационной среды*, включающей разнородные информационные ресурсы. В последнее время используются облачные технологии для реализации информационных сервисов. Облачные вычисления представляют собой комплексное решение, которое предоставляет ИТ-ресурсы в виде сервисов. Это решение основано на интернет-технологиях. Модель облачных технологий состоит из внутренней и внешней частей, которые взаимодействуют через Интернет. В качестве самого облака выступает внутренняя часть. Внешняя часть включает в себя клиентский компьютер или сети компьютеров организации и приложений, которые используются для доступа в облако. Внутренняя часть предоставляет компьютеры, приложения, хранилища данных и серверы, реализующие облачные сервисы. Различаются следующие модели сервисов облачных вычислений, которые имеют свою определенную функциональность и уровень сервисов: 1. *IaaS модель* (англ. Infrastructure-as-a-Service, инфраструктура как услуга) предоставляется как возможность использования облачной инфраструктуры для самостоятельного управления ресурсами обработки, хранения, сетей и другими вычислительными ресурсами. Это первый уровень, который является основой облачных вычислений. Он состоит из физических активов – сетевых устройств, серверов, дисков и т. д., т.е. пользователю предоставляются услуги по аппаратному обеспечению облачных технологий. Пользователи самостоятельно разворачивают операционные системы, приложения, хранилища данных, сетевые компоненты и т.д.; 2. *PaaS модель* (PaaS, англ. Platform-as-a-Service, платформа как услуга), когда потребителю услуг предоставляется доступ к операционной системе и системе управления базами данных с возможностью последующего размещения приложений в облаке при помощи инструментальных средств. Модель относится к промежуточному уровню (между аппаратным обеспечением и приложениями пользователей); 3. *SaaS модель* (англ. Software-as-a-Service, программное обеспечение как услуга), в которой потребителю предоставляется возможность использования прикладного программного обеспечения провайдера, работающего в облачной инфраструктуре и доступного из различных клиентских устройств или

посредством тонкого клиента, из браузера или интерфейс программы. Это верхний уровень – уровень приложений, при котором поставщик услуг разрабатывает веб-приложение и самостоятельно управляет им, предоставляя пользователям доступ к программному обеспечению через Интернет.

Этапы проектирования ИС

На первом этапе основным подходом в *проектировании ИС* был метод "снизу-вверх", когда система создавалась как набор отдельных приложений, наиболее важных в данный момент для поддержки деятельности предприятия. С помощью этого подхода определяются взаимосвязи между элементами системы по горизонтальным и вертикальным уровням. Основной целью этих проектов было не создание тиражируемых продуктов, а обслуживание текущих потребностей конкретного учреждения. Такой подход отчасти сохраняется и сегодня. В рамках "лоскутной автоматизации" достаточно хорошо обеспечивается поддержка отдельных функций, но практически полностью отсутствует стратегия развития комплексной системы автоматизации, а объединение функциональных подсистем превращается в самостоятельную и достаточно сложную проблему. Создавая свои отделы и управления автоматизации, предприятия пытались "обустроиться" своими силами. Однако периодические изменения технологий работы и должностных инструкций, сложности, связанные с разными представлениями пользователей об одних и тех же данных, приводили к непрерывным доработкам программных продуктов для удовлетворения все новых и новых пожеланий отдельных работников. Как следствие - и работа программистов, и создаваемые ИС вызывали недовольство руководителей и пользователей системы.

Следующий этап в области проектирования ИС связан с осознанием того факта, что существует потребность в достаточно *стандартных программных средствах* автоматизации деятельности различных предприятий и организаций. Системы начали проектировать "сверху-вниз", т.е. в предположении, что одна программа должна удовлетворять потребности многих пользователей. Обобщающий подход "сверху-вниз", называемый целевым, целенаправленным, системно-целевым, основан на структуризации или декомпозиции системы. Этот подход позволяет расчленить исходную большую неопределенность на более обозримые компоненты и выбрать методы их анализа и проектирования, сохраняя целостность представления об исследуемой системе или решаемой проблеме на основе иерархической структуры (древовидной, стратифицированной). Подход основан на структуризации в пространстве. Подход применялся при разработке ИС для крупных предприятий (организаций). Сама идея использования универсальной программы накладывает существенные ограничения на возможности разработчиков по формированию *структуры базы данных, экранных форм,* по выбору алгоритмов расчета. Заложенные "сверху" жесткие рамки не дают возможности гибко адаптировать систему к специфике деятельности конкретного предприятия: учесть необходимую глубину аналитического и производственно-технологического учета, включить необходимые процедуры обработки данных, обеспечить интерфейс каждого рабочего места с учетом функций и технологии работы конкретного пользователя. Решение этих задач требует серьезных доработок системы. Таким образом, материальные и временные затраты на внедрение системы и ее доводку под требования заказчика обычно значительно превышают запланированные показатели.

Подходы "снизу" и "сверху" называют также *каузальным* и *аксиологическим* соответственно. На практике обычно эти подходы сочетают.

Каузальное представление системы – описание системы в терминах влияния одних переменных на другие, без употребления понятий цели и средств достижения целей. Этот термин происходит от понятия "cause" – причина, т.е. подразумевает причинно-следственные отношения. При каузальном представлении будущее состояние системы определяется предыдущими состояниями и воздействиями среды. Такое представление является развитием отображения системы в виде "пространства состояний", характерного для большинства математических методов моделирования. Каузальный подход применяется на этапе предварительного описания системы, когда цель еще не сформулирована. Аксиологическое представление системы – отображение системы в терминах целей и целевых функционалов. Этот подход используется на начальном этапе моделирования, когда уже определены цели и средств достижения целей.

Кроме обобщенных подходов "снизу" и "сверху", существуют и специальные подходы к проектированию систем: информационный, кибернетический, когнитивный, ситуационный, структурно-лингвистический подход (основан идее постепенной формализации модели принятия решения) и др.

Согласно статистическим данным, собранным Standish Group (США), из 8380 проектов, обследованных в США в 1994 году, неудачными оказались более 30% проектов, общая стоимость которых превышала 80 миллиардов долларов. При этом оказались выполненными в срок лишь 16% от общего числа проектов, а перерасход средств составил 189% от запланированного бюджета.

В то же время, заказчики ИС стали выдвигать все больше требований, направленных на обеспечение возможности комплексного использования корпоративных данных в управлении и планировании своей деятельности. Таким образом, возникла необходимость формирования новой методологии построения информационных систем, которая позволяет использовать методы моделирования и автоматизации в процессе проектирования. В настоящее время для проектирования информационных систем широкое применение нашел подход, основанный на анализе бизнес-процессов, кратко называемый процессным.

Процессный подход основан на структуризации во времени, на представлении процессов в форме графов. Применение данного подхода долгое время было практически нереализуемым из-за большой трудоемкости, отсутствия правил и средств автоматизации формирования графов, отображающих процессы в системах.

В 1990-е гг. была разработана методология SADT – (Structured Analysis and Design – структурный анализ и проектирование, которая предложена **Дугласом Россом. Методология** SADT представляет собой совокупность методов, правил и процедур, предназначенных для построения функциональной модели объекта исследуемой предметной области. На ее основе разработаны и стали широко применяться функционально-ориентированные и объектно-ориентированные технологии. Компьютерная реализация методологии SADT получила название IDEF (Icain Definition). Основными структурными моделями являются модели процессов IDEFO и IDEF3, модель данных IDEF1X. Созданы стандарты IDEF и DFD, ориентированные на анализ процессов, в том числе бизнес-процессов. Для разработки моделей применяются автоматизированные средства – BPWin, ARIS, язык UML (Unified Modeling Language – Унифицированный язык моделирования) и др.

Основными задачами, решению которых должна способствовать любая методология проектирования ИС, являются следующие:

– обеспечивать создание ИС, отвечающей целям и задачам организации, а также предъявляемым требованиям по автоматизации деловых процессов заказчика;
– обеспечивать создание системы с заданным качеством, в заданные сроки и в рамках установленного *бюджета проекта*;
– поддерживать эффективные механизмы сопровождения, модификации и наращивания системы;
– обеспечивать преемственность разработки, т.е. использование в разрабатываемой ИС существующей информационной инфраструктуры организации (задела в области информационных технологий).

Проектирование ИС охватывает три основные области:

1) проектирование объектов данных, которые будут реализованы в базе данных;
2) проектирование программ работы системы, *экранных форм*, отчетов, которые будут обеспечивать выполнение запросов к данным;

3) учет конкретной среды или технологии, а именно: топологии сети, конфигурации аппаратных средств, используемой *архитектуры (файл-сервер,* клиент-сервер, облачных технологий), параллельной обработки, распределенной обработки данных и т.п.

Проектирование информационных систем всегда начинается с ***определения цели проекта***. В общем виде цель проекта можно определить как решение ряда взаимосвязанных задач, включающих в себя обеспечение на момент *запуска системы* и в течение всего времени ее эксплуатации:

- обеспечение требуемой функциональности системы и уровня ее адаптивности к изменяющимся условиям функционирования;
- обеспечение требуемой пропускной способности системы;
- обеспечение требуемого времени реакции системы на запрос;
- обеспечение *безотказной* работы системы;
- обеспечение необходимого уровня безопасности;
- обеспечение простоты эксплуатации и поддержки системы.

Процесс создания ИС делится на ряд ***этапов (стадий***), ограниченных некоторыми временными рамками и заканчивающихся выпуском конкретного продукта (моделей, программных продуктов, документации и пр.). Реально сложившаяся практика проектирования ИС предусматривает следующие этапы (стадии) проектирования:

- предпроектное обследование (формирование требований к системе),
- проектирование,
- реализация,
- тестирование,
- ввод в действие,
- эксплуатация и сопровождение.

Предпроектное обследование (формирование требований к системе)

Выполняется на начальном этапе процесса создания ИС, который также называется этапом предпроектного анализа объекта автоматизации (анализ предметной области). К предметной области может относиться деятельность организации, отдельные бизнес-процессы, отдельные производственные функции или процессы и т.п. Целью начального *этапа* является формирование требований к ИС, корректно и точно отражающих цели и задачи организации-заказчика. Чтобы специфицировать процесс создания ИС, отвечающей потребностям заказчика, нужно выяснить и четко сформулировать, в чем заключаются эти потребности. *Для этого необходимо определить требования заказчика к ИС и отобразить их на языке моделей в требования к разработке проекта ИС так, чтобы обеспечить соответствие целям и задачам организации.* На начальном этапе процесса создания ИС выполняется моделирование бизнес-процессов, протекающих в организации и реализующих ее цели и задачи. Модель организации, описанная в терминах бизнес-процессов и бизнес-функций, позволяет сформулировать основные требования к ИС. Это фундаментальное положение современной методологии обеспечивает объективность в выработке требований к проектированию системы. Множество моделей описания требований к ИС затем преобразуется в систему моделей, описывающих концептуальный проект ИС. Формируются модели архитектуры ИС, модели требований к программному обеспечению (ПО) и информационному обеспечению (ИО). Затем формируется архитектура ПО и ИО, выделяются отдельные приложения, определяются структура используемых документов и данных, информационные процессы,

формируются модели требований к приложениям и проводится их разработка, тестирование и интеграция.

Задача формирования требований к ИС является одной из наиболее ответственных, трудно формализуемых и наиболее дорогих и тяжелых для исправления в случае ошибки. Современные инструментальные средства и программные продукты позволяют достаточно быстро создавать ИС по готовым требованиям. Но зачастую эти системы не удовлетворяют заказчиков, требуют многочисленных доработок, что приводит к резкому удорожанию *фактической стоимости* ИС. Основной причиной такого положения является неправильное, неточное или неполное определение требований к ИС на этапе анализа.

Этап предпроектного обследования завершается разработкой технического задания на создание ИС.

Этап проектирования ИС

На этапе проектирования, прежде всего, разрабатываются модели данных. Проектировщики в качестве исходной информации получают результаты предпроектного обследования. Построение логической и физической моделей данных является основной частью *проектирования базы данных ИС*. Полученная в процессе предпроектного анализа структура документов и данных сначала преобразуется в логическую, а затем в *физическую модель данных*. Параллельно с проектированием *схемы базы данных* выполняется проектирование процессов, чтобы получить спецификации (описания) всех модулей ИС. Оба эти процесса проектирования тесно связаны, поскольку часть бизнес-логики обычно реализуется в базе данных (ограничения, триггеры, хранимые процедуры). Главная цель проектирования процессов заключается в отображении функций, полученных на этапе анализа, в модули информационной системы. При проектировании модулей определяют интерфейсы программ: структура меню, вид окон, горячие клавиши и связанные с ними вызовы и т.д. Конечными продуктами этапа проектирования являются:

- схема базы данных (на основании ER-модели);
- набор *спецификаций модулей* системы (строятся на базе моделей функций).

На этапе проектирования осуществляется также разработка архитектуры ИС, включающая в себя выбор платформы (платформ) ИС и операционной системы (операционных систем). В неоднородной ИС могут работать несколько компьютеров на разных аппаратных платформах и под управлением различных операционных систем. Кроме выбора платформы, на этапе проектирования определяются следующие характеристики архитектуры:

- будет ли это архитектура "файл-сервер", "клиент-сервер", сервис-ориентированная архитектура;
- будет ли это 3-уровневая архитектура со следующими слоями: сервер, ПО промежуточного слоя (сервер приложений), клиентское ПО;
- будет ли база данных централизованной или распределенной. Если база данных будет распределенной, то какие механизмы поддержки согласованности и актуальности данных будут использоваться;
- будет ли база данных однородной, то есть, будут ли все серверы баз данных продуктами одного и того же производителя (например, все серверы только Oracle или все серверы только *DB2* UDB). Если база данных не будет однородной, то какое ПО будет использовано для обмена данными между СУБД разных производителей (уже существующее или разработанное специально как часть проекта);
- будут ли для достижения нужной производительности использоваться параллельные серверы баз данных (например, Oracle Parallel Server, *DB2* UDB и т.п.).

Этап проектирования завершается разработкой *технического проекта* ИС.

Этап реализации

На этапе реализации осуществляется создание программного обеспечения системы, установка технических средств, разработка эксплуатационной документации, выполняется тестирование отдельных подсистем и готовой ИС в целом. Этап тестирования обычно оказывается распределенным во времени. После завершения разработки отдельного модуля системы выполняют автономный тест, который преследует две основные цели:

- обнаружение отказов модуля (жестких сбоев);
- соответствие модуля спецификации (наличие всех необходимых функций, отсутствие лишних функций, корректность алгоритмов обработки данных).

После того как автономный тест успешно пройден, модуль включается в состав разработанной части системы и группа сгенерированных модулей проходит тесты связей, которые должны отследить их совместное функционирование.

Далее группа модулей тестируется на надежность работы, то есть проходят, во-первых, тесты имитации отказов системы, а во-вторых, тесты наработки на отказ. Первая группа тестов показывает, насколько хорошо система восстанавливается после сбоев программного обеспечения и отказов аппаратного обеспечения. Вторая группа тестов определяет степень *устойчивости системы* при штатной работе и позволяет оценить время *безотказной* работы системы. В комплект тестов устойчивости должны входить тесты, имитирующие пиковую нагрузку на систему и обработку пороговых (наименьших и наибольших) значений используемых данных.

Затем весь комплект модулей проходит системный тест - тест внутренней приемки продукта, показывающий уровень его качества. Сюда входят тесты функциональности и тесты надежности системы.

Последний тест информационной системы - приемо-сдаточные испытания. Такой тест предусматривает показ информационной системы заказчику и должен содержать группу тестов, моделирующих реальные бизнес-процессы, чтобы показать *соответствие реализации требованиям заказчика*.

Этап реализации завершается разработкой рабочего проекта ИС (рабочей документации).

Ввод в действие ИС

Ввод в действие ИС включает в себя следующие виды работ:

-подготовка объекта к внедрению проекта - осуществляется комплекс работ по подготовке предприятия к внедрению разработанного проекта ИС;

-опытное внедрение проекта - осуществляют проверку правильности работы некоторых частей проекта и получают исправленную проектную документацию и "Акт о проведении опытного внедрения";

-сдача ИС в промышленную эксплуатацию - осуществляют комплексную системную проверку всех частей проекта, в результате которой получают доработанный "Техно-рабочий проект" и "Акт приемки проекта в промышленную эксплуатацию".

Эксплуатация и сопровождение

Эксплуатация и сопровождение ИС включает в себя следующие виды работ:

-эксплуатация ИС - получают информацию о работе всей системы в целом и отдельных ее компонентов и собирают статистику о сбоях системы в виде рекламаций и замечаний, которые накапливаются для выполнения следующего этапа;

-сопровождение и модернизация ИС - ликвидируются последствия сбоев в работе системы и исправляются ошибки, не выявленные при внедрении. В процессе модернизации ИС либо дорабатывается, т.е. расширяется по составу подсистем и задач, либо производится перенос системы на другую программную или техническую платформу с целью адаптации ее к изменяющимся внешним и внутренним условиям функционирования, в результате чего получают документы модернизированного "Техно-рабочего проекта".

Таким образом, процесс проектирования ИС представляет собой *процесс построения, последовательного преобразования и согласования моделей на всех этапах жизненного цикла* (ЖЦ) ИС. На каждом этапе ЖЦ создаются соответствующие этому этапу модели (организационной структуры, требований к ИС, проекта ИС, требований к отдельным приложениям и т.д.). Модели по мере из разработки сохраняются и накапливаются в *репозитории* проекта. Построение моделей, их контроль, преобразование и предоставление в коллективное пользование *команды проекта* осуществляется с использованием специальных программных инструментов - CASE-средств (*Computer-Aided Software Engineering*- средства автоматизации разработки программ).

Автоматизация проектирования ИС

Тенденции развития современных информационных технологий приводят к постоянному возрастанию сложности информационных систем, создаваемых в различных областях экономики и управления. Современные крупные проекты ИС характеризуются, как правило, следующими особенностями:

- сложность описания процессов автоматизации (достаточно большое количество функций, процессов, элементов данных и сложные взаимосвязи между ними), что приводит к необходимости моделирования и анализа данных и процессов;
- сложность структуры создаваемой системы, включающей совокупность тесно взаимодействующих компонентов (подсистем), которые предназначены для решения определенных локальных задач (например, приложения для аналитической обработки данных, использующих нерегламентированные запросы к данным большого объема; подсистемы поддержки принятия решений, использующие в основе обработку экспертных знаний), что приводит к необходимости формализации специальных областей знаний;
- отсутствие подобных аналогов, что ограничивает возможность использования типовых проектных решений или существующих приложений;
- необходимость интеграции существующих и вновь разрабатываемых приложений, что требует тщательной проверки и тестирования на уровне данных, чтобы не было потери или их искажения;
- функционирование распределенной системы в неоднородной среде на различных платформах;
- разобщенность и разнородность отдельных групп разработчиков по уровню квалификации и сложившимся традициям использования тех или иных инструментальных средств;
- существенная временная протяженность проекта, обусловленная, с одной стороны, ограниченными возможностями коллектива разработчиков, и, с другой стороны, масштабами организации-заказчика и различной степенью готовности отдельных ее подразделений к внедрению ИС.

Для успешной реализации проекта на создание ИС объект автоматизации должен быть, прежде всего, адекватно описан, должны быть построены полные и непротиворечивые функциональные, информационные, алгоритмические, математические модели ИС. Накопленный к настоящему времени опыт проектирования ИС показывает, что описание объекта автоматизации представляет собой логически сложную, трудоемкую и длительную по времени работу, которая требует высокой квалификации участвующих в ней специалистов. С учетом этого проектирование сложной ИС на интуитивном уровне с применением неформализованных методов, основанных на искусстве, практическом опыте, экспертных оценках и дорогостоящих экспериментальных проверках качества функционирования ИС, не приводит к нужному или ожидаемому результату. Кроме того, в процессе создания и функционирования ИС информационные потребности пользователей могут изменяться или уточняться, что еще более усложняет разработку и сопровождение больших систем. Для эффективного решения задач проектирования необходимо использовать инструментальные средства, которые основаны на применении определенной методологии. Методология представляет собой учение о методах, способах и стратегиях исследования предмета. Разные методологии различаются технологией выполнения работ и решения задач, т.е. используют разные методы, способы, приемы, средства.

В 70-х и 80-х годах при разработке ИС достаточно широко применялась структурная методология, предоставляющая в распоряжение разработчиков строгие формализованные методы описания ИС и принимаемых технических решений. Структурная методология основана на наглядной графической технике, которая позволяет описать различные модели ИС с использованием схем и диаграмм. Наглядность и строгость средств структурного анализа позволяла разработчикам и будущим пользователям системы с самого начала неформально участвовать в создании ИС, обсуждать технические решения и участвовать в принятии проектных решений. Однако, широкое применение этой методологии и следование ее рекомендациям при разработке конкретных ИС без средств автоматизации было крайне затруднено или практически невозможно. Вручную очень трудно разработать и графически представить строгие формальные спецификации сложной системы, проверить их на полноту и непротиворечивость, вносить изменения. Ручная разработка обычно порождала следующие проблемы:

- неадекватная спецификация требований;
- неспособность обнаруживать ошибки в проектных решениях;
- низкое качество документации, снижающее эксплуатационные качества ИС;
- затяжной цикл и неудовлетворительные результаты тестирования ИС.

Перечисленные факторы способствовали появлению программных средств специального класса - CASE-средств (Computer Aided Software Engineering), позволяющих автоматизировать процесс создания и сопровождения ИС (компьютерная поддержка программной инженерии). Первоначальное значение термина CASE подразумевало автоматизацию разработки только лишь программного обеспечения (ПО), в настоящее время значение термина расширилось и охватывает процесс разработки ИС на всех этапах. Таким образом, под термином CASE-средства понимаются программные средства, поддерживающие процессы создания и сопровождения ИС, включая анализ и формулировку требований, проектирование прикладного ПО (приложений) и баз данных, генерацию кода, тестирование, документирование, обеспечение качества, конфигурационное управление и управление проектом, а также другие процессы.

Появлению CASE-средств предшествовали исследования в области методологии программирования. Программирование обрело черты системного подхода с разработкой и внедрением языков высокого уровня, методов структурного и модульного программирования, языков проектирования и средств их поддержки, формальных и неформальных языков описаний системных требований и спецификаций и т.д. Кроме того, появлению CASE-средств способствовали и такие факторы, как:

- подготовка аналитиков и программистов, применяющих методы и средства модульного и структурного программирования;
- широкое внедрение и постоянный рост производительности компьютеров, позволившие использовать эффективные графические средства и автоматизировать большинство этапов проектирования;

- внедрение сетевых технологий, позволяющих объединить усилия отдельных исполнителей в единый процесс проектирования путем использования общей базы данных, содержащей необходимую информацию о проекте.

Согласно обзору передовых технологий (Survey of Advanced Technology), составленному фирмой Systems Development Inc. в 1996 г. по результатам анкетирования более 1000 американских фирм, CASE-технология в настоящее время попала в разряд наиболее стабильных информационных технологий (ее использовала половина всех опрошенных пользователей более чем в трети своих проектов, из них 85% завершились успешно). Однако, несмотря на все потенциальные возможности CASE-средств, существует множество примеров их неудачного внедрения, в результате которых CASE-средства становятся "полочным" ПО (shelfware). В связи с этим необходимо отметить следующие особенности их применения:

- CASE-средства не обязательно дают немедленный эффект; он может быть получен только спустя какое-то время;
- реальные затраты на внедрение CASE-средств обычно намного превышают затраты на их приобретение;
- CASE-средства обеспечивают возможности для получения существенной выгоды только после успешного завершения процесса их внедрения.

Различные виды CASE-средств с учетом условий их применения позволяют достигать разный эффект от их применения. Ниже указываются основные факторы, которые усложняют определение возможного эффекта от использования CASE-средств:

- широкое разнообразие качества и возможностей CASE-средств;
- недостаток опыта применения CASE-средств в организациях;
- широкий диапазон предметных областей проектов;
- различная степень интеграции CASE-средств в различных проектах.

Вследствие указанных причин доступная информация о реальных внедрениях крайне ограничена и противоречива. Она зависит от типа средств, характеристик проектов, уровня сопровождения, опыта разработчиков и пользователей и др. Некоторые аналитики считают, что реальная выгода от использования некоторых типов CASE-средств может быть получена только после одно- или двухлетнего опыта. Другие считают, что эффект может реально проявиться только на этапе эксплуатации ИС, когда технологические улучшения могут привести к снижению эксплуатационных затрат.

Для успешного внедрения CASE-средств организация должна обладать следующими качествами:

- *Технология.* Организация должна быть готова принять новую технологию в силу объективных причин, связанных с ограниченными существующими возможностями;
- *Культура.* Организация должна быть готова к внедрению новых процессов и взаимоотношений между разработчиками и пользователями;
- *Управление.* Должно быть четкое руководство и высокая организованность по отношению к наиболее важным этапам и процессам внедрения.

Если организация не обладает хотя бы одним из перечисленных качеств, то внедрение CASE-средств может закончиться неудачей независимо от степени тщательности следования различным рекомендациям по их внедрению.

Для того, чтобы принять взвешенное решение относительно инвестиций в CASE-технологию, пользователи вынуждены производить оценку отдельных CASE-средств, опираясь на неполные и противоречивые данные. Эта проблема зачастую усугубляется недостаточным знанием всех возможных "подводных камней" использования CASE-средств. Среди наиболее важных проблем выделяются следующие:

- достоверная оценка отдачи от инвестиций в CASE-средства затруднительна ввиду отсутствия приемлемых метрик и данных по проектам и процессам разработки ИС;
- внедрение CASE-средств может представлять собой достаточно длительный процесс и может не принести немедленной отдачи. Возможно даже краткосрочное снижение продуктивности в результате усилий, затрачиваемых на внедрение. Вследствие этого руководство организации-пользователя может утратить интерес к CASE-средствам и прекратить поддержку их внедрения;

- отсутствие полного соответствия между теми процессами и методами, которые поддерживаются CASE-средствами, и теми, которые используются в данной организации, может привести к дополнительным трудностям;
- CASE-средства зачастую трудно использовать в комплексе с другими подобными средствами. Это объясняется как различными методологиями, поддерживаемыми различными средствами, так и проблемами передачи данных и управления от одного средства к другому;
- некоторые CASE-средства требуют слишком много усилий для того, чтобы оправдать их использование в небольшом проекте, хотя даже при этом можно извлечь выгоду в той части, в которой и предназначено их применение;
- негативное отношение персонала к внедрению новой CASE-технологии может быть главной причиной провала проекта.

Пользователи CASE-средств должны быть готовы к необходимости долгосрочных затрат на эксплуатацию, частому появлению новых версий и возможному быстрому моральному старению средств, а также постоянным затратам на обучение и повышение квалификации персонала.

Однако при грамотном и разумном подходе к использованию CASE-средств предприятия преодолевают перечисленные трудности и достигают достаточно высокой эффективности в создании сложных ИС. Успешное внедрение CASE-средств позволяет получить такие выгоды как:

- высокий уровень технологической поддержки процессов разработки и сопровождения ПО;
- положительное воздействие на следующие факторы: производительность, качество продукции, соблюдение стандартов, документирование;
- приемлемый уровень отдачи от инвестиций в CASE-средства.

Жизненный цикл программного обеспечения ИС

Методология проектирования информационных систем описывает процесс создания, эксплуатации и развития систем в виде *жизненного цикла* (ЖЦ) ИС, представляя его как некоторую последовательность стадий. Каждая из стадий представляет собой выполнение определенных видов работ. Взаимосвязанная последовательность работ, которая дает определенный выходной результат, представляется в виде *процесса ЖЦ*.

Жизненный цикл (life cycle) системы, продукта, услуги, проекта или других изготовленных человеком объектов – это период, который начинается со стадии создания (например, с разработки концепции) и заканчивается прекращением применения данного объекта. ***Жизненный цикл*** ИС можно представить как ряд событий, происходящих с системой в процессе ее создания и использования. Для каждого этапа ЖЦ определяются состав и последовательность выполняемых работ, получаемые результаты, методы и средства, необходимые для выполнения работ, роли и ответственность участников и другие характеристики. Формальное описание ЖЦ ИС позволяет спланировать и организовать процесс разработки и обеспечить управление этим процессом.

Система (system) согласно стандарту ГОСТ Р ИСО/МЭК 12207-99: Комплекс, состоящий из процессов, технических и программных средств, устройств и персонала, обладающий возможностью удовлетворять установленным потребностям или целям.

Процесс (process) согласно стандарту ГОСТ Р ИСО/МЭК 12207-99: Набор взаимосвязанных работ, которые преобразуют исходные данные в выходные результаты. В стандарте ГОСТ Р ИСО/МЭК 12207-99 определяются процессы жизненного цикла программных средств, которые могут быть реализованы при заказе, поставке, разработке, эксплуатации и сопровождении программных продуктов.

Модель жизненного цикла (life cycle model) согласно стандарту ГОСТ Р ИСО/МЭК 12207-99: 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.

В настоящее время наиболее известны следующие *модели жизненного цикла программного обеспечения информационной системы:*

- ***Каскадная модель*** (также называется ***водопадная, последовательная***) предусматривает последовательное выполнение всех *этапов проекта* в строго фиксированном порядке (рис. 2.1). Переход на следующий этап означает полное завершение работ на предыдущем этапе. Требования, определенные на стадии формирования требований, строго документируются в виде технического задания и фиксируются на все время разработки проекта. Каждая стадия завершается выпуском полного комплекта документации, достаточной для того, чтобы разработка могла быть продолжена другой командой разработчиков;
- ***Итерационная (***также называется ***поэтапная или эволюционная) модель с промежуточным контролем*** (рис. 2.2). Предполагает разбиение жизненного цикла проекта на последовательность итераций, каждая из которых напоминает "мини-проект", включая все процессы разработки в применении к созданию меньших фрагментов функциональности, по сравнению с проектом в целом. Цель каждой *итерации* — получение работающей версии программной системы, включающей функциональность, определённую интегрированным содержанием всех предыдущих и текущей итерации. Результат последней итерации содержит всю требуемую функциональность программного продукта. Таким образом, с завершением каждой итерации продукт получает приращение (*инкремент*) к его возможностям, которые, следовательно, развиваются *эволюционно.* Межэтапные корректировки позволяют учитывать реально существующее взаимовлияние результатов разработки на различных этапах; время жизни каждого из этапов растягивается на весь период разработки;
- ***Спиральная модель*** (рис. 2.3). Основана на классическом цикле Деминга PDCA (plan-do-check-act). При использовании этой модели ПО создается в несколько итераций (витков спирали) методом прототипирования. Каждая итерация соответствует созданию фрагмента или версии ПО, на ней уточняются цели и характеристики проекта, оценивается качество полученных результатов и планируются работы следующей итерации. На каждой итерации оцениваются: риск превышения сроков и стоимости проекта; необходимость выполнения ещё одной итерации; степень полноты и точности понимания требований к системе; целесообразность прекращения проекта. Особое внимание уделяется начальным этапам разработки – анализу и проектированию, где реализуемость тех или иных технических решений проверяется и обосновывается посредством создания прототипов (*макетирования*).

Рис. 2.1. Каскадная модель ЖЦ ИС

Рис. 2.2. Итерационная (поэтапная) модель с промежуточным контролем

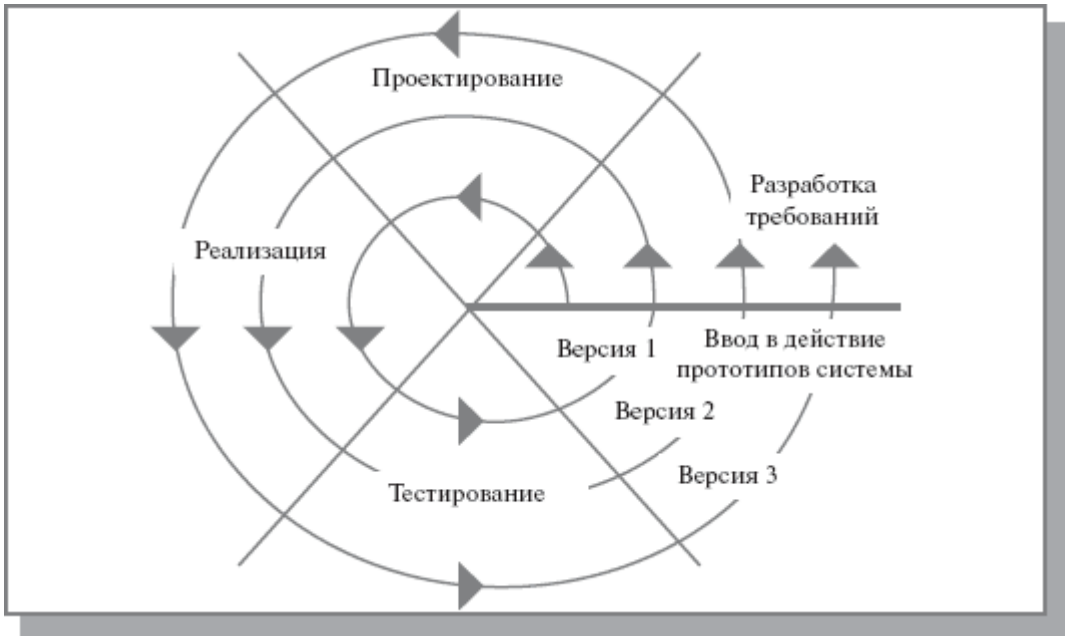


Рис. 2.3. Спиральная модель ЖЦ ИС

Каскадная модель была предложена в 1970 г. Уинстоном Ройсом и характерна для периода 1970-1985 гг., *итерационная модель* стала альтернативой каскадной модели и в 70-е гг. получила название эволюционной модели от Т. Гилба, *спиральная модель* была разработана в середине 1980-х годов Барри Боэмом.

Ранние проекты были ориентированы на создание достаточно простых ИС, каждое приложение представляло собой единый, функционально и информационно независимый блок. Для разработки такого типа приложений эффективным оказался каскадный подход. Каждый этап завершался после полного выполнения и документального оформления всех предусмотренных работ. Этапы проекта в соответствии с каскадной моделью:

1. Формирование требований;
2. Проектирование;
3. Реализация;
4. Тестирование;
5. Внедрение;
6. Эксплуатация и сопровождение.

При этом каскадная модель предполагает разработку законченных продуктов на каждом этапе: технического задания, *технического проекта*, программного продукта, пользовательской документации. Разработанная документация позволяет не только определить требования к продукту следующего этапа, но и определить обязанности сторон, объем работ и сроки.

Каскадный подход хорошо зарекомендовал себя при построении относительно простых ИС, когда в самом начале разработки можно достаточно точно и полно сформулировать все требования к системе.

Недостатки каскадной модели

В каскадной модели переход от одной фазы проекта к другой предполагает полную корректность результата (выхода) предыдущей фазы. Однако неточность какого-либо требования или некорректная его интерпретация в результате приводит к тому, что приходится "откатываться" к ранней фазе проекта и требуемая переработка не просто выбивает проектную команду из графика, но приводит часто к значительному росту затрат, а иногда и к прекращению проекта в той форме, в которой он изначально задумывался. Основным недостатком каскадной модели является то, что реальный процесс создания системы никогда полностью не укладывается в такую жесткую схему, постоянно возникает потребность в возврате к предыдущим этапам и уточнении или пересмотре ранее принятых решений. В результате реальный процесс создания ИС оказывается соответствующим *итерационной* (*поэтапной) модели с промежуточным контролем.* В каскадной модели окончательная оценка сроков и стоимости проекта производится на начальных этапах, после завершения обследования. Очевидно, что если требования к информационной системе меняются в ходе реализации проекта, а качество документов оказывается невысоким (требования неполны и/или противоречивы), то в действительности использование *каскадной модели* создает лишь иллюзию определенности и фактически увеличивает риски. Кроме того, при этом еще и уменьшается ответственность участников проекта, поскольку при формальном подходе *менеджер проекта* реализует только те требования, которые содержатся в спецификации, опирается на документ, а не на реальные потребности бизнеса. Таким образом, каскадная модель для крупных проектов мало реалистична и может быть эффективно использована только для создания небольших систем. Несмотря на настойчивые рекомендации компаний – вендоров и экспертов в области проектирования и разработки ИС, многие компании продолжают использовать *каскадную модель* вместо какого-либо варианта итерационной модели. К основным причинам, по которым *каскадная модель* сохраняет свою популярность, можно отнести следующие:

1. Определенность в приемке/сдаче работ и финансировании проекта;
2. Иллюзия снижения рисков участников проекта (заказчика и исполнителя).

Итерационная модель с промежуточным контролем **эффективна** при создании сложной системы, если она реализуется путем небольших шагов (небольшого объема работ на каждом этапе) и если каждый шаг заключает в себе четко определённый результат, а также при наличии возможности "отката" к предыдущему успешному этапу в случае неудачи.

Недостатки итерационной модели

Итерационная модель также не позволяет оперативно учитывать возникающие изменения и уточнения требований к системе, целостное понимание возможностей и ограничений проекта очень долгое время отсутствует. Согласование результатов разработки с пользователями производится только в точках, планируемых после завершения каждого этапа работ, а общие требования к ИС зафиксированы в виде технического задания на все время ее создания. В этом случае разработчики стремятся не выходить за рамки принятых требований, а заказчики, пока нет системы, не могут дополнить или изменить требования. Таким образом, пользователи зачастую получают систему, не удовлетворяющую их

реальным потребностям. Кроме того, при итерациях приходится отбрасывать часть сделанной ранее работы.

Спиральная модель ЖЦ была предложена для преодоления перечисленных проблем для каскадной и итерационной моделей. На этапах анализа и проектирования реализуемость технических решений и степень удовлетворения потребностей заказчика проверяется путем создания прототипов. Каждый виток спирали соответствует созданию работоспособного фрагмента или версии системы. Это позволяет уточнить требования, цели и характеристики проекта, определить качество разработки, спланировать работы следующего витка спирали. Таким образом, углубляются и последовательно конкретизируются детали проекта и в результате выбирается обоснованный вариант, который удовлетворяет действительным *требованиям заказчика* и доводится до реализации. Спиральная модель позволяет переходить на следующий этап, не дожидаясь полного завершения работы на текущем и решить главную задачу – как можно быстрее показать пользователям системы работоспособный продукт, тем самым активизируя процесс уточнения и дополнения требований.

Отличительной особенностью спиральной модели является специальное внимание, уделяемое рискам, влияющим на организацию жизненного цикла, и контрольным точкам. Также важно понимать, что спиральная модель является не альтернативой эволюционной модели, а специально проработанным вариантом. *Спиральная модель* чаще применяется при разработке информационной системы силами собственного ИТ - отдела предприятия.

Недостатки спиральной модели

Основная проблема спиральной модели – определение момента перехода на следующий этап. Для ее решения вводятся временные ограничения на каждый из этапов *жизненного цикла*, и переход осуществляется в соответствии с планом, даже если не вся запланированная работа закончена. Планирование производится на основе статистических данных, полученных в предыдущих проектах, и личного опыта разработчиков. Иногда возникают проблемы внедрения спиральной модели. В некоторых областях *спиральная модель* не может применяться, поскольку невозможно использование/тестирование продукта, обладающего неполной функциональностью (например, военные разработки, атомная энергетика, диспетчерские системы реального времени и т.п.). Поэтапное итерационное внедрение информационной системы для бизнеса возможно, но сопряжено с технологическими и организационными трудностями (перенос данных, интеграция систем, изменение бизнес-процессов, изменение *учетной политики*, обучение пользователей). Трудозатраты при использовании спиральной модели оказываются на начальном этапе оценки значительно выше по сравнению с другими моделями, а управление проектом требует настоящего искусства. С учетом указанных особенностей, заказчики часто выбирают *каскадную модель*, чтобы повысить определенность в выполнении проекта и его результативности.

Существует ряд стандартов, регламентирующих ЖЦ ПО, в некоторых стандартах регламентируются также процессы разработки.

Методологии и стандарты в области информационных систем

Значительный вклад в теорию проектирования и *разработки информационных систем* внесла компания IBM, предложив еще в середине 1970-х годов методологию *BSP* (Business System Planning - система планирования бизнеса). Метод структурирования информации путем построения матриц пересечения бизнес-процессов, функциональных подразделений, функций систем обработки данных (информационных систем), информационных объектов, документов и баз данных, предложенный в *BSP*, используется сегодня не только в ИТ-проектах, но и проектах по *реинжинирингу бизнес-процессов*, изменению организационной структуры и др. Основные шаги процесса *BSP*, их последовательность

(получить поддержку высшего руководства, определить процессы предприятия, определить классы данных, провести интервью, обработать и организовать данные интервью) можно встретить практически во всех формальных методиках, а также в проектах, реализуемых на практике.

В области управления информационными технологиями выработаны определенные подходы, своды знаний и методологии. Наиболее известными из них являются ITIL (IT Infrastructure Library), COBIT (Control Objectives for Information and Related Technologies), PRINCE2 (PRojects IN Controlled Environments 2), TOGAF (The Open Group Architecture Framework).

ITIL обобщает опыт международных практик, описывает практики управления информационными технологиями на уровне процедур, инструкций и функций. На основе ITIL разработан международный стандарт (ГОСТ Р ИСО/МЭК 20000–2010 Информационная технология. Менеджмент услуг).

Создатели метода COBIT объединили лучшее из международных технических стандартов, стандартов управления качеством, аудиторской деятельности, а также практические требования и опыт многих специалистов. Основная ценность метода COBIT заключается в том, что он предлагает модель, обеспечивающую взаимосвязь между требованиями бизнеса и ИТ – процессами. Стандарт COBIT и библиотека ITIL дополняют друг друга, охватывая разные сферы деятельности и разные уровни управления. Метод COBIT помогает понять, что следует делать для решения поставленной задачи, а ITIL показывает, как этого достичь.

Метод PRINCE2 предназначен для управления проектами, одобрен правительством Великобритании в качестве стандарта управления проектами в социальной сфере. Согласно методологии TOGAF ИТ-архитектура рассматривается как формальное описание системы или детальный план системы на уровне компонент, на основании которого осуществляется реализация системы. ИТ – архитектура отличается от ИТ- стратегии более детальным представлением деятельности.

Существующие стандарты на проектирование и разработку ИС можно сгруппировать следующим образом:

1. По предмету стандартизации – это функциональные стандарты (стандарты на языки программирования, интерфейсы, протоколы) и стандарты на организацию жизненного цикла (ЖЦ), создание и использование автоматизированных систем и программного обеспечения (ПО).
2. По утверждающей организации – это официальные международные стандарты, официальные национальные ведомственные (ГОСТы) стандарты международных консорциумов и комитетов по стандартизации, стандарты "Де-факто" (например, стандарт языка SQL) и фирменные стандарты (например, Microsoft ODBS).
3. По методическому источнику – это методические материалы фирм консультантов, научных центров, консорциумов по стандартизации. Например, методические материалы ORACLE Method корпорации ORACLE.

Среди наиболее известных стандартов можно выделить следующие:

- ISO/*IEC* 12207:2008 Systems and software engineering — Software life cycle processes - стандарт ISO, описывающий процессы жизненного цикла программного обеспечения (ISO - Международная организация по стандартизации). Стандарт устанавливает общую структуру процессов жизненного цикла программных средств, на которую можно ориентироваться в программной индустрии. Стандарт определяет процессы, виды деятельности и задачи, которые используются при приобретении программного продукта или услуги, а также при поставке, разработке, применении по назначению, сопровождении и прекращении применения программных продуктов. Стандарт не содержит описания фаз, стадий и этапов.
- ISO/*IEC* 15288 -2008 Проектирование систем — Процессы жизненного цикла системы.
- ГОСТ Р ИСО/МЭК 12207 – 2010 "Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств" (взамен ГОСТ Р ИСО/МЭК 12207-99 "Информационная технология. Процессы жизненного цикла программных средств"). МЭК (IEC) - Международная электротехническая комиссия. Настоящий стандарт идентичен международному

стандарту ИСО/МЭК 12207-2008* "Системная и программная инженерия. Процессы жизненного цикла программных средств" (ISO/IEC 12207:2008 "System and software engineering - Software life cycle processes"). В настоящем стандарте не детализируются процессы жизненного цикла в терминах методов или процедур, необходимых для удовлетворения требований и достижения результатов процесса. Настоящий стандарт не устанавливает требований к документации в части ее наименований, форматов, определенного содержания и носителей для записи.

- ГОСТ 34.601-90 "Автоматизированные системы. Стадии создания". Настоящий стандарт распространяется на автоматизированные системы (АС), используемые в различных видах деятельности (исследование, проектирование, управление и т.п.), включая их сочетания, создаваемые в организациях, объединениях и на предприятиях (далее - организациях). В стандарте содержится описание содержания работ на каждом этапе. Стадии и этапы создания АС выделяются как части процесса создания по соображениям рационального планирования и организации работ, заканчивающихся заданным результатом. Стадии и этапы работы, закрепленные в стандарте, в большей степени соответствуют каскадной модели жизненного цикла.

- Custom Development Method (методика Oracle) по разработке прикладных информационных систем - технологический материал, детализированный до уровня заготовок проектных документов, рассчитанных на использование в проектах с применением Oracle. Применяется *CDM* для классической модели ЖЦ (предусмотрены все работы/задачи и этапы), а также для технологий "быстрой разработки" (Fast Track) или "облегченного подхода", рекомендуемых в случае малых проектов.
- Rational *Unified* Process (*RUP*) предлагает итеративную модель разработки, включающую четыре фазы: начало, исследование, построение и внедрение. Каждая фаза может быть разбита на этапы (итерации), в результате которых выпускается версия для внутреннего или внешнего использования. Прохождение через четыре основные фазы называется *циклом разработки*, каждый цикл завершается генерацией версии системы. Если после этого работа над проектом не прекращается, то полученный продукт продолжает развиваться и снова минует те же фазы. Суть работы в рамках *RUP* - это создание и сопровождение моделей на базе UML.
- *Microsoft Solution Framework* (*MSF*) сходна с *RUP*, так же включает четыре фазы: анализ, проектирование, разработка, стабилизация, является итерационной, предполагает использование *объектно-ориентированного моделирования*. *MSF* в сравнении с *RUP* в большей степени ориентирована на разработку бизнес-приложений.
- *Extreme Programming* (XP). *Экстремальное программирование* (самая новая среди рассматриваемых методологий) сформировалось в 1996 году. В основе методологии командная работа, эффективная коммуникация между заказчиком и исполнителем в течение всего проекта по разработке ИС, а разработка ведется с использованием последовательно дорабатываемых прототипов.

На отдельных примерах рассмотрим разные уровни стандартов.

ISO/IEC 12207:2008 Systems and software engineering — Software life cycle processes

Стандарт ISO/IEC 12207 был опубликован 1 августа 1995 года и явился первым международным стандартом, содержавшим представительный набор процессов ЖЦ, действий и задач в отношении ПО, которое рассматривалось как часть большей системы, а также применительно к программным продуктам и услугам. За стандартом ISO/IEC 12207 в ноябре 2002 года последовал стандарт ISO/IEC 15288, посвященный процессам ЖЦ систем. Широта применения ПС привела к тому, что ПО и процессы его разработки не могли рассматриваться в отрыве от систем, но только как составная часть системы и процесса её создания. В Дополнениях к стандарту ISO/IEC 12207 были введены цель процесса и его выходы и определена эталонная модель процесса, отвечающая требованиям стандарта ISO/IEC 15504-2. Международный стандарт ISO/IEC 12207:2008, представляет собой переработанные и исправленные дополнения к стандарту ISO/IEC 12207 и является первым шагом в стратегии SC7 по гармонизации

спецификаций, имеющей целью создание полностью интегрированного набора процессов ЖЦ систем и программных средств и руководства по их применению.

Стандарт определяет стратегию и общий порядок в создании и эксплуатации ПО он охватывает жизненный цикл ПО от создания концепции ПО до завершения жизненного цикла. Общая структура стандарта представляет собой набор процессов жизненного цикла. Каждый процесс разделен на набор действий, каждое действие разделено на набор задач. Очень важная особенность стандарта ISO заключается в следующем: каждый процесс, действие или задача инициируется и выполняется другим процессом по мере необходимости, нет заранее определенных последовательностей (кроме логики связей по задачам и др.).

В стандарте ISO/IEC 12207:2008 выделяется 7 групп процессов жизненного цикла:

- процессы соглашения — 2;
- процессы организационного обеспечения проекта — 5;
- процессы проекта — 7;
- технические процессы — 11;
- процессы реализации программных средств — 7;
- процессы поддержки программных средств — 8;
- процессы повторного применения программных средств — 3.

Процессы соглашения

- Поставка
- Приобретение

Процессы организационного обеспечения проекта

- Процесс менеджмента модели жизненного цикла;
- Процесс менеджмента инфраструктуры;
- Процесс менеджмента портфеля проектов;
- Процесс менеджмента людских ресурсов;
- Процесс менеджмента качества.

Процессы проекта

- Процессы менеджмента проекта
  - процесс планирования проекта;
  - процесс управления и оценки проекта.
- Процессы поддержки проекта
  - процесс менеджмента решений;
  - процесс менеджмента рисков;
  - процесс менеджмента конфигурации;
  - процесс менеджмента информации;
  - процесс измерений.

Технические процессы

- Определение требований правообладателей
- Анализ системных требований

- Проектирование архитектуры системы
- Процесс реализации
- Процесс комплексирования системы
- Процесс квалификационного тестирования системы
- Процесс инсталляции программных средств
- Процесс поддержки приемки программных средств
- Процесс функционирования программных средств
- Процесс сопровождения программных средств
- Процесс изъятия из обращения программных средств

Процессы реализации программных средств

- Процесс анализа требований к программным средствам;
- Процесс проектирования архитектуры программных средств;
- Процесс детального проектирования программных средств;
- Процесс конструирования программных средств;
- Процесс комплексирования программных средств;
- Процесс квалификационного тестирования программных средств

Процессы поддержки программных средств

- Процесс менеджмента документации программных средств;
- Процесс менеджмента конфигурации программных средств;
- Процесс обеспечения гарантии качества программных средств;
- Процесс верификации программных средств;
- Процесс валидации программных средств;
- Процесс ревизии программных средств;
- Процесс аудита программных средств;
- Процесс решения проблем в программных средствах.

Процессы повторного применения программных средств

- Процесс проектирования доменов;
- Процесс менеджмента повторного применения активов;
- Процесс менеджмента повторного применения программ.

Для поддержки практического применения стандарта ISO/*IEC* 12207 разработан ряд технологических документов: Руководство для ISO/*IEC* 12207 (ISO/*IEC* TR 15271:1998 Information technology - Guide for ISO/*IEC* 12207); Руководство по применению ISO/*IEC* 12207 к управлению проектами (ISO/*IEC* TR 16326:1999 *Software engineering* - Guide for the application of ISO/*IEC* 12207 to *project management*).

Государственный стандарт РФ ГОСТ Р ИСО/МЭК 12207-99 содержит полный аутентичный текст международного стандарта ИСО/МЭК 12207-95 "Информационная технология. Процессы жизненного цикла программных средств".

В таблице 2.1 приведены ориентировочные описания некоторых процессов ЖЦ программных средств согласно стандарту ISO/*IEC* 12207.

Таблица 2.1. Содержание процессов ЖЦ программных средств ISO/*IEC* 12207

Процесс (исполнитель процесса)	Действия	Вход	Результат
Приобретение (заказчик)	• Инициирование • Подготовка заявочных предложений • Подготовка договора • Контроль деятельности поставщика • Приемка ИС	• Решение о начале работ по внедрению ИС • Результаты обследования деятельности заказчика • Результаты анализа рынка ИС/ тендера • План поставки/ разработки • Комплексный тест ИС	• Технико-экономическое обоснование внедрения ИС • Техническое задание на ИС • Договор на поставку/ разработку • Акты приемки этапов работы • Акт приемно-сдаточных испытаний
Поставка (разработчик ИС)	• Инициирование • Ответ на заявочные предложения • Подготовка договора • Планирование исполнения • Поставка ИС	• Техническое задание на ИС • Решение руководства об участии в разработке • Результаты тендера • Техническое задание на ИС • *План управления проектом* • Разработанная ИС и документация	• Решение об участии в разработке • *Коммерческие предложения*/ конкурсная заявка • Договор на поставку/ разработку • *План управления проектом* • Реализация/ корректировка • Акт приемно-сдаточных испытаний
Реализация (разработчик ИС)	• Подготовка • *Анализ требований* к ИС • Проектирование архитектуры ИС • Разработка требований к ПО • Проектирование архитектуры ПО • Детальное проектирование ПО • Кодирование и тестирование ПО • Интеграция ПО и квалификационное тестирование ПО • Интеграция ИС и квалификационное тестирование ИС	• Техническое задание на ИС • Техническое задание на ИС, модель ЖЦ • Подсистемы ИС • Спецификации требования к компонентам ПО • Архитектура ПО • Материалы детального проектирования ПО • План интеграции ПО, тесты • Архитектура ИС, ПО, документация на ИС, тесты	• Используемая модель ЖЦ, стандарты разработки • План работ • Состав подсистем, компоненты оборудования • Спецификации требования к компонентам ПО • Состав компонентов ПО, интерфейсы с БД, план интеграции ПО • Проект БД, спецификации интерфейсов между компонентами ПО, требования к тестам • Тексты модулей ПО, акты автономного тестирования • Оценка

Процесс (исполнитель процесса)	Действия	Вход	Результат
			соответствия комплекса ПО требованиям ТЗ • Оценка соответствия ПО, БД, технического комплекса и комплекта документации требованиям ТЗ

Особенности стандарта ISO/IEC 12207:2008:

1) Динамический характер стандарта заключается в такой последовательности процессов и задач, при которой один процесс вызывает другой или его часть;

2) Степень адаптивности - максимальное множество процессов и задач сконструированы так, что возможна их адаптация в соответствии с ПО;

3) Стандарт принципиально не содержит конкретные методы действий заготовки решений или документаций. Он описывает архитектуру жизненного цикла ПО, но не конкретизирует её в деталях. Польза стандарта в том, что он содержит наборы задач, характеристик, критериев оценки и др. дающие всесторонний охват проектных ситуаций.

ГОСТ Р ИСО/МЭК 12207-2010. Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств

Стандарт устанавливает общую структуру процессов жизненного цикла программных средств, на которую можно ориентироваться в программной индустрии. Стандарт определяет процессы, виды деятельности и задачи, которые используются при приобретении программного продукта или услуги, а также при поставке, разработке, применении по назначению, сопровождении и прекращении применения программных продуктов. Стандарт используется при приобретении систем, программных продуктов и услуг, при их поставке, разработке, применении по назначению, сопровождении и прекращении применения программных продуктов и программных компонентов системы как в самой организации, так и вне ее. Эти аспекты системного определения включаются в настоящий стандарт для обеспечения содержания понятий программных продуктов и услуг. Стандарт устанавливает также процесс, который может использоваться при определении, управлении и совершенствовании процессов жизненного цикла программных средств. Процессы, виды деятельности и задачи настоящего стандарта - самостоятельно либо совместно с ИСО/МЭК 15288 - могут также использоваться во время приобретения системы, содержащей программные средства.

Настоящий стандарт группирует различные виды деятельности, которые могут выполняться в течение жизненного цикла программных систем, в семь групп процессов. Каждый из процессов жизненного цикла в пределах этих групп описывается в терминах цели и желаемых выходов, списков действий и задач, которые необходимо выполнять для достижения этих результатов. Группы процессов (рис. 1.2):

a) процессы соглашения - два процесса;

b) процессы организационного обеспечения проекта - пять процессов;

c) процессы проекта - семь процессов;

d) технические процессы - одиннадцать процессов;

e) процессы реализации программных средств - семь процессов;

f) процессы поддержки программных средств - восемь процессов;

g) процессы повторного применения программных средств - три процесса.

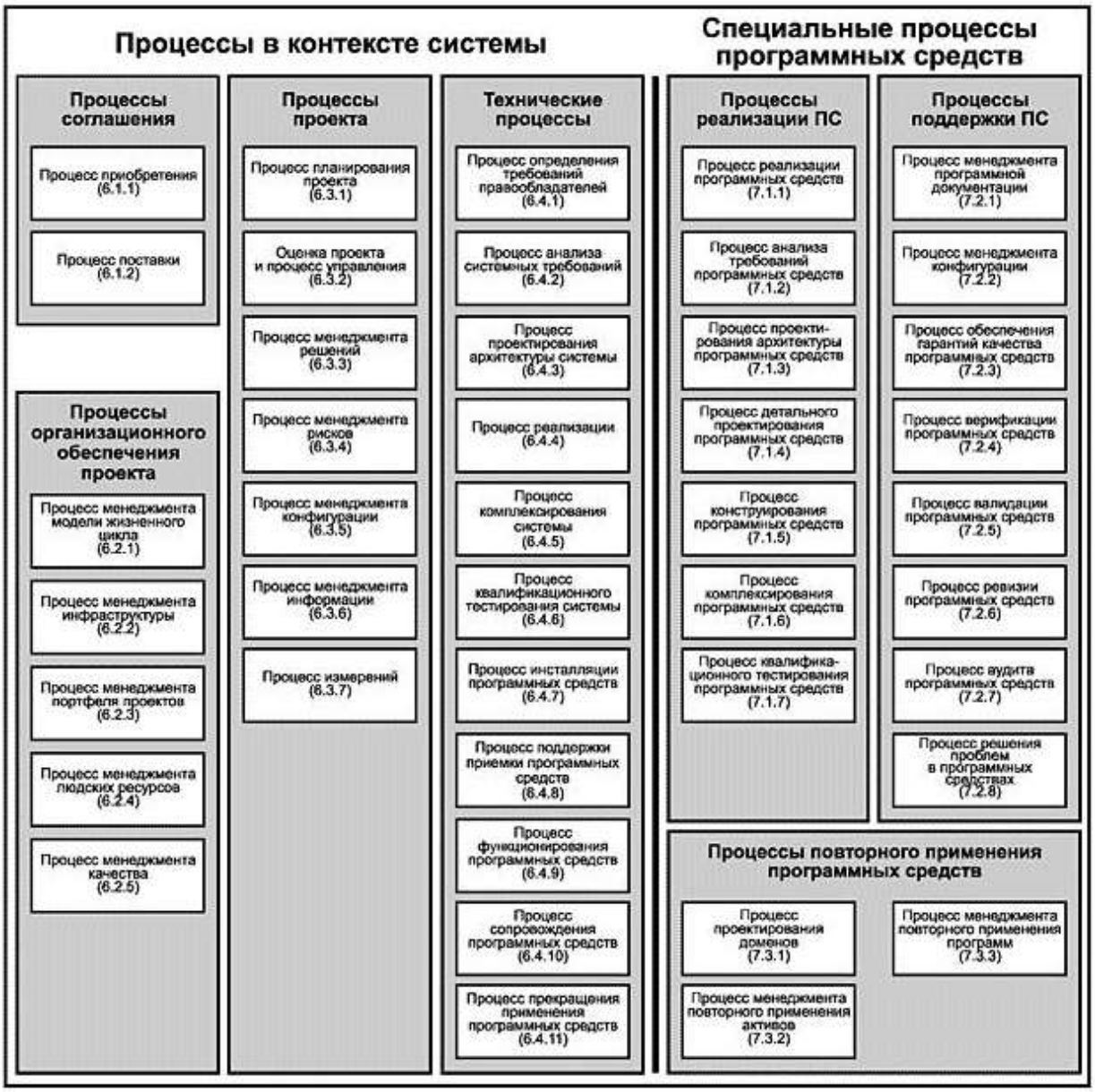


Рисунок 1.2 - Группы процессов жизненного цикла по ГОСТ Р ИСО/МЭК 12207-2010

Цели и прописанные результаты процессов жизненного цикла образуют эталонную модель процессов.

Эталонная модель процесса не представляет конкретного подхода к реализации процесса, как и не предопределяет модель жизненного цикла системы (программного средства), методологию или технологию. Эталонная модель определяет общую структуру процессов ЖЦ ИС и конкретизируется с учетом деловых потребностей организации и области приложений. Планируемые организацией процессы представляются в проекте организации в контексте требований заказчиков.

Результаты процесса используются для демонстрации успешного достижения цели процесса. Анализ результатов выполнения процесса помогает руководителю процесса определять новые возможности и предоставлять исходные материалы для планирования повышения эффективности выполнения процесса.

*ISO/*IEC *15288 -2008 Проектирование систем — Процессы жизненного цикла системы*

Международный стандарт на процессы *жизненного цикла* систем (ISO/*IEC* 15288 *System life cycle processes*)**.** Первое издание было опубликовано в 2002 г., переведен на русский язык в 2005 г. В настоящее время издан ISO/IEC/IEEE 15288:2015 "Systems and software engineering -- System life cycle processes".

К разработке стандарта ISO/*IEC* 15288 -2008 были привлечены специалисты различных областей: системной инженерии, программирования, управления качеством, человеческими ресурсами, безопасностью и пр. Был учтен практический опыт создания систем в правительственных, коммерческих, военных и академических организациях. Стандарт применим для широкого класса систем, но его основное предназначение - поддержка создания компьютеризированных систем.

В стандарте ISO/*IEC* 15288-2008 процессы жизненного цикла описаны в четырех группах процессов:

Процессы соглашения;
Процессы предприятия;
Процессы проекта;

Технические процессы.

Каждый процесс жизненного цикла может быть вызван, по требованию, в любое время в ходе всего жизненного цикла, и нет никакого определенного порядка в их использовании. Любой процесс жизненного цикла может быть выполнен одновременно с любым другим процессом жизненного цикла. Любой процесс жизненного цикла может применяться на любом уровне в иерархическом представлении структуры системы. Поэтому в последующем описании процессов жизненного цикла системы порядок представления процессов и используемые группы процессов не подразумевают строгой последовательности в течение жизненного цикла системы. Группы процессов, отражают основные концепции, используемые в данном международном стандарте. Процессы проекта на создание ИС состоят из следующих процессов:

a) Процесс планирования проекта;
b) Процесс оценивания проекта;
c) Процесс управления проектом;
d) Процесс принятия решений;
e) Процесс управления рисками;
f) Процесс управления конфигурацией;

g) Процесс управления информацией.

Планирование, оценивание и управление являются ключевыми для всех процессов. Они проявляются в управлении любого предприятия, начиная от целой организации до отдельного процесса жизненного цикла и его видов деятельности. В данном международном стандарте, Проект был выбран в качестве контекста для того, чтобы описать процессы, связанные с планированием, оцениванием и управлением.

Системы, которые рассматриваются в данном международном стандарте ISO/*IEC* 15288-2008, создаются и используются для того, чтобы обеспечить информационные сервисы в определенных средах в соответствии с требованиями пользователей. Эти системы могут быть скомпонованы из одного или более из следующих элементов: аппаратные средства, программное обеспечение, люди, процессы (например, процесс анализа), процедуры (например, инструкции оператора), средства и природные объекты (например, вода, организмы, полезные ископаемые). В данном стандарте люди рассматриваются

как пользователи и как элементы системы. В первом случае человек-пользователь является лицом, извлекающим выгоду из эксплуатации системы. Во втором случае человек является оператором, выполняющим определенные системные функции. Человек может, одновременно или последовательно быть и пользователем, и элементом системы. Люди вносят свой вклад в рабочие и другие характеристики системы по многим причинам, например, в связи с их специальными навыками, потребностью в приспособляемости, по юридическим причинам и др. Являются ли они пользователями или операторами, люди очень сложны, а их поведение часто трудно предсказать, и они нуждаются в защите от вреда. Это требует, чтобы процессы жизненного цикла системы отражали влияние человеческого фактора в следующих областях: устойчивость функционирования системы, системная безопасность, оценка факторов риска, квалификация персонала.

Процессы жизненного цикла системы в данном международном стандарте описываются по отношению к системе, которая составлена из наборов взаимодействующих элементов, каждый из которых может быть реализован для того, чтобы выполнить соответствующие установленные требования. Поэтому ответственность за реализацию любого элемента системы может быть передана другой стороне посредством соглашения. Взаимосвязь между системой и полным набором ее элементов обычно может быть разрешена только для наиболее простой системы. Для более сложных исследуемых систем может потребоваться, чтобы сам предполагаемый элемент системы рассматривался как система (которая, в свою очередь, состоит из элементов) прежде, чем с уверенностью можно будет определить полный набор элементов системы. Таким способом, процессы жизненного цикла системы рекурсивно применяются к исследуемой системе, чтобы разложить ее структуру на составляющие элементы, когда понятные и управляемые элементы системы могут быть или реализованы, или повторно использованы, или приобретены у другой стороны.

ИСО (ISO) и МЭК (IEC) образуют специализированную систему всемирной стандартизации. Государственные органы, являющиеся членами ИСО или МЭК, участвуют в разработке международных стандартов посредством технических комитетов, учрежденных соответствующей организацией для того, чтобы обсуждать определенные области технической деятельности. Технические комитеты ИСО и МЭК сотрудничают в областях взаимного интереса. Другие международные организации, правительственные и неправительственные, контактирующие с ИСО и МЭК, также принимают участие в работе. В области информационных технологий ИСО и МЭК учредили Совместный технический комитет СТК ИСО/МЭК JTC 1. Проекты международных стандартов составляются в соответствии с правилами, определенными директивами ИСО/МЭК.

Основная задача совместного технического комитета состоит в том, чтобы подготавливать международные стандарты. Проекты международных стандартов, принятые совместным техническим комитетом, рассылаются государственным органам на голосование. Для опубликования документа в качестве международного стандарта необходимо как минимум 75% голосов членов-организаций, принимающих участие в голосовании.

Методика ORACLE CDM (Custon Development Method) ORACLE Casl, Designer/2000

Методика выделяет следующие этапы цикла:

1) Анализ – формулирование детальных требований к системе;
2) Проектирование – преобразование требований в детальные спецификации системы;
3) Реализация – написание и тестирование приложений;
4) Внедрение – установка новой системы, подготовка к началу эксплуатации;
5) Эксплуатация – поддержка и слежение за приложением, планирование будущих расширений.

Методика выделяет следующие процессы:

- Определение производственных требований;
- Исследование существующих систем;
- Определение архитектуры;
- Проектирование и построение базы данных;
- Проектирование и реализация модулей;

– Конвертирование данных;
– Документирование;
– Тестирование;
– Обучение;
– Переход к новой системе;
– Поддержка и сопровождение.

Особенности методики ORACLE CDM:

– степень адаптивности CDM ограничивается тремя моделями жизненного цикла: 1) "классическая" (предусмотрены все работы/задачи и этапы); 2) "быстрая" еще более сильно ориентированная на использование инструментов моделирования и программирования ORACLE; 3) "облегченный подход", рекомендуемый в случае малых проектов и возможности быстро прототипировать приложения;
– модели жизненного цикла АС являются по сути каскадными;
– степень обязательности: методика необязательна, но может считаться фирменным стандартом;
– направленность на создание ИС с базами данных.

Организация разработки ИС

Каноническое проектирование ИС

Организация канонического проектирования ИС ориентирована на использование главным образом каскадной модели жизненного цикла ИС. Стадии и этапы работы описаны в отечественном стандарте **ГОСТ 34.601- 90**.

В зависимости от сложности объекта автоматизации и набора задач, требующих решения при создании конкретной ИС, стадии и этапы работ могут иметь различную трудоемкость. Допускается объединять последовательные этапы и даже исключать некоторые из них на любой стадии проекта. Допускается также начинать выполнение работ следующей стадии до окончания предыдущей.

Стадии и этапы создания ИС, выполняемые участниками проекта, устанавливаются в договорах и техническом задании на основе настоящего стандарта.

Стадии и составляющие их этапы создания автоматизированной системы (АС), указанные в стандарте, приведены ниже.

Стадия 1. Формирование требований к АС:
- обследование объекта и обоснование необходимости создания АС;
- формирование требований пользователей к АС;
- оформление отчета о выполненной работе и тактико - технического задания на разработку.

Стадия 2. Разработка концепции АС:
- изучение объекта автоматизации;
- проведение необходимых научно-исследовательских работ;
- разработка вариантов концепции АС, удовлетворяющих требованиям пользователей;
- оформление отчета и утверждение концепции.

Стадия 3. Техническое задание:
- разработка и утверждение технического задания на создание АС.

Стадия 4. Эскизный проект:
- разработка предварительных проектных решений по системе и ее частям;
- разработка эскизной документации на АС и ее части.

Стадия 5. Технический проект:
- разработка проектных решений по системе и ее частям;
- разработка документации на АС и ее части;
- разработка и оформление документации на поставку комплектующих изделий;
- разработка заданий на проектирование в смежных частях проекта.

Стадия 6. Рабочая документация:

- разработка рабочей документации на АС и ее части;
- разработка и адаптация программ.

Стадия 7. Ввод в действие:

- подготовка объекта автоматизации;
- подготовка персонала;
- комплектация АС поставляемыми изделиями (программными и техническими средствами, программно - техническими комплексами, информационными изделиями);
- строительно-монтажные работы;
- пусконаладочные работы;
- проведение предварительных испытаний;
- проведение опытной эксплуатации;
- проведение приемочных испытаний.

Стадия 8. Сопровождение АС:

- выполнение работ в соответствии с гарантийными обязательствами;
- послегарантийное обслуживание.

Ниже приводится краткая характеристика отдельных стадий и этапов работ.

Формирование требований к АС, разработка концепции АС (предпроектное обследование).

На начальной стадии проектирования ИС выполняется предпроектное обследование объекта автоматизации. Обследование объекта - это изучение и диагностический анализ организационной структуры предприятия, его деятельности и существующей системы обработки информации. Материалы, полученные в результате обследования, используются для следующих работ:

- обоснования разработки и поэтапного внедрения систем;
- составления технического задания на разработку систем;
- разработки технического и рабочего проектов систем.

На этапе обследования объекта целесообразно выделить две составляющие: *определение стратегии внедрения ИС и детальный анализ деятельности организации.* Основная задача первого этапа обследования - оценка реального объема проекта, его целей и задач на основе выявленных функций и информационных элементов автоматизируемого объекта высокого уровня. Эти задачи могут быть реализованы или заказчиком ИС самостоятельно, или с привлечением консалтинговых организаций. Этап предполагает тесное взаимодействие с основными потенциальными пользователями системы и бизнес - экспертами. Основная задача взаимодействия - получить полное и однозначное понимание требований заказчика. Как правило, нужная информация может быть получена в результате интервью, бесед или семинаров с руководством, экспертами и пользователями. По завершении этого этапа появляется возможность определить вероятные технические подходы к созданию системы и оценить затраты на ее реализацию (затраты на аппаратное обеспечение, закупаемое программное обеспечение и разработку нового программного обеспечения). Результатом этапа определения стратегии является *документ (технико-экономическое обоснование проекта)*, где четко сформулировано, что получит заказчик, если согласится финансировать проект, когда он получит готовый продукт (график выполнения работ) и сколько это будет стоить (для крупных проектов должен быть составлен график финансирования на разных этапах работ). В документе желательно отразить не только затраты, но и выгоду проекта, например время окупаемости проекта, ожидаемый экономический эффект (если его удается оценить).

Ориентировочное содержание документа - технико-экономическое обоснование проекта:

- ограничения, риски, критические факторы, которые могут повлиять на успешность проекта;
- совокупность условий, при которых предполагается эксплуатировать будущую систему: архитектура системы, аппаратные и программные ресурсы, условия функционирования, обслуживающий персонал и пользователи системы;
- сроки завершения отдельных этапов, форма приемки/сдачи работ, привлекаемые ресурсы, меры по защите информации;
- описание выполняемых системой функций;

- возможности развития системы;
- информационные объекты системы;
- интерфейсы и распределение функций между человеком и системой;
- требования к программным и информационным компонентам ПО, требования к СУБД;
- что не будет реализовано в рамках проекта.

На этапе *детального анализа деятельности организации* изучаются задачи, обеспечивающие реализацию функций управления, организационная структура, штаты и содержание работ по управлению предприятием, а также характер подчиненности вышестоящим органам управления. На этом этапе должны быть выявлены:

- инструктивно-методические и директивные материалы, на основании которых определяются состав подсистем и перечень задач;
- возможности применения новых методов решения задач.

Аналитики собирают и фиксируют информацию в двух взаимосвязанных формах:

- функции - информация о событиях и процессах, которые происходят в бизнесе;
- сущности - информация о вещах, имеющих значение для организации и о которых что-то известно.

При изучении каждой функциональной задачи управления определяются:

- наименование задачи; сроки и периодичность ее решения;
- степень формализуемости задачи;
- источники информации, необходимые для решения задачи;
- показатели и их количественные характеристики;
- порядок корректировки информации;
- действующие алгоритмы расчета показателей и возможные методы контроля;
- действующие средства сбора, передачи и обработки информации;
- действующие средства связи;
- принятая точность решения задачи;
- трудоемкость решения задачи;
- действующие формы представления исходных данных и результатов их обработки в виде документов;
- потребители результатной информации по задаче.

Одной из наиболее трудоемких, хотя и хорошо формализуемых задач этого этапа является описание документооборота организации. При обследовании документооборота составляется схема маршрута движения документов, которая должна отразить:

- количество документов;
- место формирования показателей документа;
- взаимосвязь документов при их формировании;
- маршрут и длительность движения документа;
- место использования и хранения данного документа;
- внутренние и внешние информационные связи;
- объем документа в знаках.

По результатам обследования объекта устанавливается перечень задач управления, решение которых целесообразно автоматизировать, и очередность их разработки. На этапе обследования следует классифицировать планируемые функции системы по степени важности. Один из форматов представления такой классификации – MuSCoW: Must have - необходимые функции; Should have - желательные функции; Could have - возможные функции; Won’t have - отсутствующие функции. Функции первой категории обеспечивают критичные для успешной работы системы возможности. Реализация функций второй и третьей категорий ограничивается временными и финансовыми рамками. Последняя категория функций особенно важна, поскольку необходимо четко представлять границы проекта и набор функций, которые будут отсутствовать в системе.

Модели деятельности организации (информационных процессов) создаются в двух видах:

- модель "как есть" ("as-is") - отражает существующие информационные процессы;
- модель "как должно быть" ("as-to-be") - отражает необходимые изменения информационных

процессов с учетом внедрения ИС.

Результаты обследования объекта позволяют разработать концепцию ИС и представляют объективную основу для формирования технического задания на создание информационной системы.

Техническое задание определяет цели, требования и основные исходные данные, необходимые для разработки ИС

ТЗ на АС является основным документом, определяющим требования и порядок создания (развития или модернизации - далее создания) автоматизированной системы, в соответствии с которым проводится разработка АС и ее приемка при вводе в действие (ГОСТ 34.602-89).

При разработке технического задания необходимо решить следующие задачи:

- установить общую цель создания ИС, определить состав подсистем и функциональных задач;
- разработать и обосновать требования, предъявляемые к подсистемам;
- разработать и обосновать требования, предъявляемые к информационной базе, математическому и программному обеспечению, комплексу технических средств (включая средства связи и передачи данных);
- установить общие требования к проектируемой системе;
- определить перечень задач создания системы и исполнителей;
- определить этапы создания системы и сроки их выполнения;
- провести предварительный расчет затрат на создание системы и определить уровень экономической эффективности ее внедрения.

Разработка ТЗ ведётся в соответствии со стандартами:

ГОСТ 34.601-90 Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадии создания;

ГОСТ 34.602-89 Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы.

ГОСТ 34.201-89 Информационная технология. Комплекс стандартов на автоматизированные системы. Виды, комплектность и обозначения документов при создании автоматизированных систем.

Комплекс стандартов 34 устанавливает состав, содержание, правила оформления документов. Структура технического задания по ГОСТ 34.602-89 представлена ниже:

УКАЗАНИЯ ГОСТ:
Настоящий стандарт распространяется на автоматизированные системы (АС) для автоматизации различных видов деятельности (управление, проектирование, исследование и т. п.), включая их сочетания, и устанавливает состав, содержание, правила оформления документа "Техническое задание на создание (развитие или модернизацию) системы" (далее - ТЗ на АС). Структура документа (разделы 4-8): 4 ТРЕБОВАНИЯ К СИСТЕМЕ 4.1 Требования к системе в целом 4.1.1 Требования к структуре и функционированию системы 4.1.1.1 Перечень подсистем, их назначение и основные характеристики 4.1.1.2 Требования к способам и средствам связи для информационного обмена между компонентами системы 4.1.1.3 Требования к характеристикам взаимосвязей создаваемой системы со смежными системами 4.1.1.4 Требования к режимам функционирования системы 4.1.1.5 Требования по диагностированию системы

Ниже приводится содержание раздела 4.3.2. из ГОСТ 34:

Требования к информационному обеспечению системы

УКАЗАНИЯ ГОСТ:
Для информационного обеспечения системы приводят требования:
1) к составу, структуре и способам организации данных в системе;
2) к информационному обмену между компонентами системы;
3) к информационной совместимости со смежными системами;
4) по использованию общесоюзных и зарегистрированных республиканских, отраслевых классификаторов, унифицированных документов и классификаторов, действующих на данном предприятии;
5) по применению систем управления базами данных;
6) к структуре процесса сбора, обработки, передачи данных в системе и представлению данных;
7) к защите данных от разрушений при авариях и сбоях в электропитании системы;
8) к контролю, хранению, обновлению и восстановлению данных;
9) к процедуре придания юридической силы документам, продуцируемым техническими средствами АС (в соответствии с ГОСТ 6.10.4).

ФОРМАЛЬНОЕ СОДЕРЖАНИЕ:
Состав, структура и способы организации данных в системе должны быть опередедены на этапе технического проектирования.
Уровень хранения данных в системе должен быть построен на основе современных реляционных или объектно-реляционных СУБД. Для обеспечения целостности данных должны использоваться встроенные механизмы СУБД.
Средства СУБД, а также средства используемых операционных систем должны обеспечивать документирование и протоколирование обрабатываемой в системе информации.
Структура базы данных должна поддерживать кодирование хранимой и обрабатываемой информации в соответствии с общероссийскими классификаторами (там, где они применимы).
Доступ к данным должен быть предоставлен только авторизованным пользователям с учетом их служебных полномочий, а также с учетом категории запрашиваемой информации.
Структура базы данных должна быть организована рациональным способом, исключающим единовременную полную выгрузку информации, содержащейся в базе данных системы.
Технические средства, обеспечивающие хранение информации, должны использовать современные технологии, позволяющие обеспечить повышенную надежность хранения данных и оперативную замену оборудования (распределенная избыточная запись/считывание данных; зеркалирование; независимые дисковые массивы; кластеризация).
В состав системы должна входить специализированная подсистема резервного копирования и восстановления данных.
При проектировании и развертывании системы необходимо рассмотреть возможность использования накопленной информации из уже функционирующих информационных систем. Перечень функционирующих информационных систем приведен в разделе 3 настоящего документа.

В состав ТЗ на создание ИС включают приложения, содержащие расчёт ожидаемой эффективности системы; оценку научно-технического уровня системы; использованные при разработке ТЗ методические и наиболее важные информационные материалы из состава документов, указанных в разделе "Источники разработки".

По результатам проделанной работы оформляется, согласовывается и утверждается документация в объеме, необходимом для описания полной совокупности принятых проектных решений и достаточном для дальнейшего выполнения работ по созданию системы.

Технический проект

На основе технического задания (и эскизного проекта) разрабатывается технический проект ИС. Технический проект системы - это техническая документация, содержащая общесистемные проектные решения, алгоритмы решения задач, а также оценку экономической эффективности автоматизированной системы управления и перечень мероприятий по подготовке объекта к внедрению.

На этом этапе осуществляется комплекс научно-исследовательских и экспериментальных работ для выбора основных проектных решений и расчет экономической эффективности системы.

Примерный состав и содержание *технического проекта* приведены в таблице 3.2.

Таблица 1.3. Содержание технического проекта (ГОСТ 34.602-89)

№ п\п	Раздел	Содержание
1	Пояснительная записка	• основания для разработки системы • перечень организаций разработчиков • краткая характеристика объекта с указанием основных *технико-экономических показателей* его функционирования и связей с другими объектами • краткие сведения об основных проектных решениях по функциональной и обеспечивающим частям системы
2	Функциональная и организационная структура системы	• обоснование выделяемых подсистем, их перечень и назначение • перечень задач, решаемых в каждой подсистеме, с краткой характеристикой их содержания • схема информационных связей между подсистемами и между задачами в рамках каждой подсистемы
3	Постановка задач и алгоритмы решения	• организационно-экономическая сущность задачи (наименование, цель решения, краткое содержание, метод, периодичность и время решения задачи, способы сбора и передачи данных, связь задачи с другими задачами, характер использования результатов решения, в которых они используются) • экономико-математическая модель задачи (структурная и развернутая форма представления) • входная оперативная информация (характеристика показателей, диапазон изменения, формы представления) • нормативно-справочная информация (НСИ) (содержание и формы представления) • информация, хранимая для связи с другими задачами • информация, накапливаемая для последующих решений данной задачи • информация по внесению изменений (система внесения изменений и перечень информации, подвергающейся изменениям) • алгоритм решения задачи (последовательность этапов расчета, схема, расчетные формулы) • контрольный пример (набор заполненных данными форм входных документов, условные документы с накапливаемой и хранимой информацией, формы выходных документов, заполненные по результатам решения экономико-технической задачи и в соответствии с разработанным алгоритмом расчета)

№ п\п	Раздел	Содержание
4	Организация информационной базы	• источники поступления информации и способы ее передачи • совокупность показателей, используемых в системе • состав документов, сроки и периодичность их поступления • основные проектные решения по организации фонда НСИ • состав НСИ, включая перечень реквизитов, их определение, диапазон изменения и перечень документов НСИ • перечень массивов НСИ, их объем, порядок и частота корректировки информации • структура фонда НСИ с описанием связи между его элементами; требования к технологии создания и ведения фонда • методы хранения, поиска, внесения изменений и контроля • определение объемов и потоков информации НСИ • контрольный пример по внесению изменений в НСИ • предложения по унификации документации
5	Альбом форм документов	
6	Система математического обеспечения	• обоснование структуры математического обеспечения • обоснование выбора системы программирования • перечень стандартных программ
7	Принцип построения комплекса технических средств	• описание и обоснование схемы технологического *процесса обработки данных* • обоснование и выбор структуры комплекса технических средств и его *функциональных групп* • обоснование требований к разработке нестандартного оборудования • комплекс мероприятий по обеспечению надежности функционирования технических средств
8	Расчет экономической эффективности системы	• сводная *смета* затрат, связанных с эксплуатацией систем • расчет годовой экономической эффективности, источниками которой являются оптимизация производственной структуры хозяйства (объединения), снижение себестоимости продукции за счет рационального использования производственных ресурсов и уменьшения потерь, улучшения принимаемых управленческих решений
9	Мероприятия по подготовке объекта к внедрению системы	• перечень организационных мероприятий по совершенствованию бизнес-процессов • перечень работ по внедрению системы, которые необходимо выполнить на стадии рабочего проектирования, с указанием сроков и ответственных лиц
10	Ведомость документов	

В завершение стадии технического проектирования производится разработка документации на поставку необходимых типовых программных и аппаратных компонентов для комплектования ИС, а также определяются технические требования и составляются ТЗ на разработку нетиповых изделий.

Рабочая документация

На стадии "Рабочая документация" осуществляется создание программного продукта и разработка всей сопровождающей документации. Документация должна содержать все необходимые и достаточные сведения для обеспечения выполнения работ по вводу ИС в действие и ее эксплуатации, а также для поддержания уровня эксплуатационных характеристик (качества) системы. Разработанная документация должна быть соответствующим образом оформлена, согласована и утверждена.

Для ИС, которые являются разновидностью автоматизированных систем, устанавливают следующие основные виды испытаний: предварительные, опытная эксплуатация и приемочные. При необходимости допускается дополнительно проведение других видов испытаний системы и ее частей.

В зависимости от взаимосвязей частей ИС и объекта автоматизации испытания могут быть автономные или комплексные. Автономные испытания охватывают части системы. Их проводят по мере готовности частей системы к сдаче в опытную эксплуатацию. Комплексные испытания проводят для групп взаимосвязанных частей или для системы в целом.

Для планирования проведения всех видов испытаний разрабатывается документ "*Программа и методика испытаний*". Разработчик документа устанавливается в договоре или ТЗ. В качестве приложения в документ могут включаться тесты или контрольные примеры.

Предварительные испытания проводят для определения работоспособности системы и решения вопроса о возможности ее приемки в опытную эксплуатацию. Предварительные испытания следует выполнять после проведения разработчиком отладки и тестирования поставляемых программных и технических средств системы и представления им соответствующих документов об их готовности к испытаниям, а также после ознакомления персонала ИС с эксплуатационной документацией.

Опытную эксплуатацию системы проводят с целью определения фактических значений количественных и качественных характеристик системы и готовности персонала к работе в условиях ее функционирования, а также определения фактической эффективности и корректировки, при необходимости, документации.

Приемочные испытания проводят для определения соответствия системы техническому заданию, оценки качества опытной эксплуатации и решения вопроса о возможности приемки системы в постоянную эксплуатацию.

Типовое проектирование ИС

Типовое проектирование ИС предполагает создание системы из готовых типовых элементов. Основополагающим требованием для применения методов типового проектирования является возможность декомпозиции проектируемой ИС на множество составляющих компонентов (подсистем, комплексов задач, программных модулей и т.д.). Для реализации выделенных компонентов выбираются имеющиеся на рынке типовые проектные решения, которые настраиваются на особенности конкретного предприятия.

Типовое проектное решение (ТПР) - это тиражируемое (пригодное к многократному использованию) проектное решение.

Принятая классификация ТПР основана на уровне декомпозиции системы. Выделяются следующие классы ТПР:

- элементные ТПР - типовые решения по задаче или по отдельному виду обеспечения задачи (информационному, программному, техническому, математическому, организационному);
- подсистемные ТПР - в качестве элементов типизации выступают отдельные подсистемы, разработанные с учетом функциональной полноты и минимизации внешних информационных связей;
- объектные ТПР - типовые отраслевые проекты, которые включают полный набор

функциональных и обеспечивающих подсистем ИС.

Каждое типовое решение предполагает наличие, кроме собственно функциональных элементов (программных или аппаратных), документации с детальным описанием ТПР и процедур настройки в соответствии с требованиями разрабатываемой системы.

Основные особенности различных классов *ТПР* приведены в таблице 1.4

Достоинства и недостатки *ТПР*		
Класс *ТПР* **Реализация** *ТПР*	**Достоинства**	**Недостатки**
Элементные *ТПР* Библиотеки методо-ориентированных программ	• обеспечивается применение модульного подхода к проектированию и документированию ИС	• большие затраты времени на сопряжение разнородных элементов вследствие информационной, программной и технической несовместимости • большие затраты времени на доработку *ТПР* отдельных элементов
Подсистемные *ТПР* Пакеты прикладных программ	• достигается высокая степень интеграции элементов ИС • позволяют осуществлять: модульное проектирование; параметрическую настройку программных компонентов на различные объекты управления • обеспечивают: сокращение затрат на проектирование и программирование взаимосвязанных компонентов; хорошее документирование отображаемых процессов обработки информации	• адаптивность *ТПР* недостаточна с позиции непрерывного инжиниринга деловых процессов • возникают проблемы в комплексировании разных функциональных подсистем, особенно в случае использования решений нескольких производителей программного обеспечения
Объектные *ТПР* Отраслевые проекты ИС	• комплексирование всех компонентов ИС за счет методологического единства и информационной, программной и технической совместимости • открытость архитектуры — позволяет устанавливать *ТПР* на разных программно-технических платформах • масштабируемость — допускает конфигурацию ИС для переменного числа рабочих мест • конфигурируемость — позволяет выбирать необходимое подмножество компонентов	• проблемы привязки типового проекта к конкретному объекту управления, что вызывает в некоторых случаях даже необходимость изменения организационно-экономической структуры объекта автоматизации

Для реализации типового проектирования используются два подхода: *параметрически - ориентированное и модельно-ориентированное проектирование.*

Параметрически-ориентированное проектирование

Параметрически-ориентированное проектирование включает следующие этапы: определение критериев оценки пригодности пакетов прикладных программ (ППП) для решения поставленных задач, анализ и оценка доступных ППП по сформулированным критериям, выбор и закупка наиболее под- ходящего пакета, настройка параметров (доработка) закупленного ППП.

Критерии оценки ППП делятся на следующие группы:

- назначение и возможности пакета;
- отличительные признаки и свойства пакета;
- требования к техническим и программным средствам;
- документация пакета;
- факторы финансового порядка;
- особенности установки пакета;
- особенности эксплуатации пакета;
- помощь поставщика по внедрению и поддержанию пакета;
- оценка качества пакета и опыт его использования;
- перспективы развития пакета.

Внутри каждой группы критериев выделяется некоторое подмножество частных показателей, детализирующих каждый из десяти выделенных аспектов анализа выбираемых ППП. Числовые значения показателей для конкретных ППП устанавливаются экспертами по выбранной шкале оценок (например, 10-балльной). На их основе формируются групповые оценки и комплексная оценка пакета (путем вычисления средневзвешенных значений). Нормированные взвешивающие коэффициенты также получаются экспертным путем.

Модельно-ориентированное проектирование

Модельно-ориентированное проектирование заключается в адаптации состава и характеристик типовой ИС в соответствии с моделью объекта автоматизации.

Технология проектирования в этом случае должна обеспечивать единые средства для работы как с моделью типовой ИС, так и с моделью конкретного предприятия. Типовая ИС в специальной базе метаинформации - репозитории - содержит модель объекта автоматизации, на основе которой осуществляется конфигурирование программного обеспечения. Таким образом, модельно-ориентированное проектирование ИС предполагает, прежде всего, построение модели объекта автоматизации с использованием специального программного инструментария (например, SAP Business Engineering Workbench (BEW), BAAN Enterprise Modeler). Возможно также создание системы на базе типовой модели ИС из репозитория, который поставляется вместе с программным продуктом и расширяется по мере накопления опыта проектирования информационных систем для различных отраслей и типов производства.

Репозиторий содержит базовую (ссылочную) модель ИС, типовые (референтные) модели определенных классов ИС, модели конкретных ИС предприятий. Базовая модель ИС в репозитории содержит описание бизнес-функций, бизнес-процессов, бизнес- объектов, бизнес-правил, организационной структуры, которые поддерживаются программными модулями типовой ИС. Типовые модели описывают конфигурации информационной системы для определенных отраслей или типов производства. Модель конкретного предприятия строится либо путем выбора фрагментов основной или типовой модели в соответствии со специфическими особенностями предприятия (BAAN Enterprise Modeler), либо путем автоматизированной адаптации этих моделей в результате экспертного опроса (SAP Business Engineering Workbench).

Построенная модель предприятия в виде метаописания хранится в репозитории и при необходимости может быть откорректирована. На основе этой модели автоматически осуществляется конфигурирование и настройка информационной системы.

Бизнес-правила определяют условия корректности совместного применения различных компонентов ИС и используются для поддержания целостности создаваемой системы.

Модель бизнес-функций представляет собой иерархическую декомпозицию функциональной деятельности предприятия.

Модель бизнес-процессов отражает выполнение работ для функций самого нижнего уровня модели бизнес-функций. Для отображения процессов используется модель управления событиями (EPC - Event- driven Process Chain). Именно модель бизнес-процессов позволяет выполнить настройку программных модулей - приложений информационной системы в соответствии с характерными особенностями конкретного предприятия.

Модели бизнес-объектов используются для интеграции приложений, поддерживающих исполнение различных бизнес-процессов (могут быть разработаны с применением языка UML).

Модель организационной структуры предприятия представляет собой традиционную иерархическую структуру подчинения подразделений и персонала.

Внедрение типовой ИС начинается с анализа требований к конкретной ИС, которые выявляются на основе результатов предпроектного обследования объекта автоматизации. Для оценки соответствия этим требованиям программных продуктов может использоваться описанная выше методика оценки ППП. После выбора программного продукта на базе имеющихся в нем референтных моделей строится предварительная модель ИС, в которой отражаются все особенности реализации ИС для конкретного предприятия. Предварительная модель является основой для выбора типовой модели системы и определения перечня компонентов, которые будут реализованы с использованием других программных средств или потребуют разработки с помощью имеющихся в составе типовой ИС инструментальных средств (например, ABAP в SAP, Tools в BAAN).

Реализация типового проекта предусматривает выполнение следующих операций:

- установку глобальных параметров системы;
- задание структуры объекта автоматизации;
- определение структуры основных данных;
- задание перечня реализуемых функций и процессов;
- описание интерфейсов;
- описание отчетов;
- настройку авторизации доступа;
- настройку системы архивирования.

Структурный анализ предметной области

Информационное обеспечение ИС

Структурная модель предметной области

В основе *проектирования ИС* лежит моделирование предметной области. Для того чтобы получить адекватный предметной области проект ИС в виде системы правильно работающих программ, необходимо иметь целостное, системное представление модели, которое отражает все аспекты функционирования будущей информационной системы. При этом под ***моделью предметной области*** понимается некоторая система, имитирующая структуру или функционирование исследуемой предметной области и отвечающая основному требованию – быть адекватной этой области.

Предварительное моделирование предметной области позволяет сократить время и сроки проведения проектировочных работ и получить более эффективный и качественный проект. Без проведения моделирования предметной области велика вероятность допущения большого количества ошибок в решении стратегических вопросов, приводящих к экономическим потерям и высоким затратам на последующее перепроектирование системы. Вследствие этого все современные технологии *проектирования ИС* основываются на использовании методологии моделирования предметной области.

К *моделям предметных областей* предъявляются следующие требования:

- формализация, обеспечивающая однозначное описание структуры предметной области;
- понятность для заказчиков и разработчиков на основе применения графических средств отображения модели;
- реализуемость, подразумевающая наличие средств физической реализации *модели предметной области* в ИС;
- обеспечение оценки эффективности реализации *модели предметной области* на основе определенных методов и вычисляемых показателей.

Для реализации перечисленных требований, как правило, строится ***система моделей***, которая отражает структурный и оценочный аспекты функционирования предметной области.

Структурный аспект предполагает построение:

- объектной структуры, отражающей состав взаимодействующих в процессах материальных и информационных объектов предметной области;
- функциональной структуры, отражающей взаимосвязь *функций* (действий) по преобразованию объектов в процессах;
- структуры управления, отражающей события и бизнес-правила, которые воздействуют на выполнение процессов;
- организационной структуры, отражающей взаимодействие *организационных единиц* предприятия и персонала в процессах;
- технической структуры, описывающей топологию расположения и способы коммуникации комплекса технических средств.

Для отображения структурного аспекта *моделей предметных областей* в основном используются графические методы, которые должны гарантировать представление информации о компонентах системы. Главное требование к графическим методам документирования — простота. Графические методы должны обеспечивать возможность структурной декомпозиции спецификаций системы с *максимальной степенью* детализации и согласований описаний на смежных уровнях декомпозиции.

С моделированием непосредственно связана проблема **выбора языка** представления проектных решений, позволяющего как можно больше привлекать будущих пользователей системы к ее разработке. ***Язык моделирования*** – это *нотация*, в основном графическая, которая используется для описания проектов. ***Нотация*** представляет собой совокупность графических объектов, используемых в модели. *Нотация* является синтаксисом *языка моделирования*. *Язык моделирования*, с одной стороны, должен делать решения проектировщиков понятными пользователю, с другой стороны, предоставлять проектировщикам средства достаточно формализованного и однозначного определения проектных решений, подлежащих реализации в виде программных комплексов, образующих целостную систему программного обеспечения.

Графическое изображение нередко оказывается наиболее емкой формой представления информации. При этом проектировщики должны учитывать, что графические методы документирования не могут полностью обеспечить декомпозицию проектных решений от постановки задачи проектирования до реализации программ ЭВМ. Трудности возникают при переходе от этапа анализа системы к этапу проектирования и в особенности к программированию.

Главный критерий адекватности структурной модели *предметной области* заключается в *функциональной полноте* разрабатываемой ИС.

Оценочные аспекты моделирования предметной области связаны с показателями эффективности автоматизируемых процессов, к которым относятся:

- время решения задач;
- стоимостные затраты на обработку данных;
- надежность процессов;
- косвенные показатели эффективности, такие, как объемы производства, производительность труда, оборачиваемость капитала, рентабельность и т.д.

Для расчета показателей эффективности, как правило, используются статические методы функционально-*стоимостного анализа* (АВС) и *динамические методы имитационного моделирования*.

В основе различных методологий моделирования предметной области ИС лежат принципы последовательной детализации абстрактных категорий. Обычно модели строятся на трех уровнях:

– на внешнем уровне (определении требований),
– на концептуальном уровне (спецификации требований)
– внутреннем уровне (реализации требований).

Так, на внешнем уровне модель отвечает на вопрос, что должна делать система, то есть определяется состав основных компонентов системы: объектов, *функций*, событий, *организационных единиц*, технических средств. На концептуальном уровне модель отвечает на вопрос, как должна функционировать система? Иначе говоря, определяется характер взаимодействия компонентов системы одного и разных типов. На внутреннем уровне модель отвечает на вопрос: с помощью каких программно-технических средств реализуются требования к системе? С позиции жизненного цикла ИС описанные уровни моделей соответственно строятся на этапах *анализа требований*, логического (технического) и физического (рабочего) проектирования. Рассмотрим особенности построения *моделей предметной области* на трех уровнях детализации.

Объектная структура

Объект — это сущность, которая используется при выполнении некоторой *функции* или *операции* (преобразования, обработки, формирования и т.д.). Объекты могут иметь динамическую или статическую природу: *динамические объекты* используются в одном цикле воспроизводства, например заказы на продукцию, счета на оплату, платежи; статические объекты используются во многих циклах воспроизводства, например, оборудование, персонал, запасы материалов.

На внешнем уровне детализации модели выделяются основные виды материальных объектов (например, сырье и материалы, полуфабрикаты, готовые изделия, услуги) и основные виды информационных объектов или документов (например, заказы, накладные, счета и т.д.).

На концептуальном уровне построения *модели предметной области* уточняется состав классов объектов, определяются их атрибуты и взаимосвязи. Таким образом, строится обобщенное представление структуры предметной области-*концептуальная модель.*

На внутреннем уровне *концептуальная модель* представляется в виде файлов базы данных, входных и выходных документов ИС. Причем *динамические объекты* представляются единицами переменной информации или документами, а *статические объекты* — единицами условно-постоянной информации в виде списков, номенклатур, ценников, справочников, *классификаторов*. Модель базы данных как постоянно поддерживаемого информационного ресурса отображает хранение условно-постоянной и накапливаемой переменной информации, используемой в повторяющихся информационных процессах.

Функциональная структура

Функция (*операция*) представляет собой некоторый преобразователь входных объектов в выходные. Последовательность взаимосвязанных по входам и выходам *функций* составляет *бизнес-процесс*. *Функция бизнес-процесса* может порождать объекты любой природы (материальные, денежные, информационные). Причем *бизнес-процессы* и информационные процессы, как правило, неразрывны, то есть *функции* материального процесса не могут осуществляться без информационной поддержки. Например, отгрузка готовой продукции осуществляется на основе документа "Заказ", который, в свою очередь, порождает документ "Накладная", сопровождающий партию отгруженного товара.

Функция может быть представлена одним действием или некоторой совокупностью действий. В последнем случае каждой *функции* может соответствовать некоторый процесс, в котором могут существовать свои *подпроцессы*, и т.д., пока каждая из подфункций не будет представлять некоторую недекомпозируемую последовательность действий.

На внешнем уровне моделирования определяется список основных бизнес-функций или видов *бизнес-процессов*. Обычно таких *функций* насчитывается 15–20.

На концептуальном уровне выделенные *функции* декомпозируются и строятся иерархии взаимосвязанных *функций*.

На внутреннем уровне отображается структура информационного процесса в компьютере: определяются иерархические структуры программных модулей, реализующих автоматизируемые *функции*.

Структура управления

В совокупности *функций бизнес-процесса* возможны альтернативные или циклические последовательности в зависимости от различных условий протекания процесса. Эти условия связаны с происходящими событиями во внешней среде или в самих процессах и с образованием определенных состояний объектов (например, заказ принят, отвергнут, отправлен на корректировку). **События** вызывают выполнение *функций*, которые, в свою очередь, изменяют состояния объектов и формируют новые события, и т.д., пока не будет завершен некоторый *бизнес-процесс*. Тогда последовательность событий составляет конкретную реализацию *бизнес-процесса*.

Каждое событие описывается с двух точек зрения: информационной и процедурной. Информационно событие отражается в виде некоторого сообщения, фиксирующего факт выполнения некоторой *функции* изменения состояния или появления нового. Процедурно событие вызывает выполнение новой *функции*, и поэтому для каждого состояния объекта должны быть заданы описания этих вызовов. Таким образом, события выступают в связующей роли для выполнения *функций бизнес-процессов*.

На внешнем уровне определяются список внешних событий, вызываемых взаимодействием предприятия с внешней средой (платежи налогов, процентов по кредитам, поставки по контрактам и т.д.), и список целевых установок, которым должны соответствовать *бизнес-процессы* (регламент выполнения процессов, поддержка уровня материальных запасов, уровень качества продукции и т.д.).

На концептуальном уровне устанавливаются бизнес-правила, определяющие условия вызова *функций* при возникновении событий и достижении состояний объектов.

На внутреннем уровне выполняется формализация бизнес-правил в виде триггеров или вызовов программных модулей.

Организационная структура

Организационная структура представляет собой совокупность *организационных единиц*, как правило, связанных иерархическими и процессными отношениями. Организационная единица — это подразделение, представляющее собой объединение людей (персонала) для выполнения совокупности общих *функций* или *бизнес-процессов*. В функционально-ориентированной организационной структуре организационная единица выполняет набор *функций*, относящихся к одной *функции* управления и входящих в различные процессы. В процессно-ориентированной структуре организационная единица выполняет набор *функций*, входящих в один тип процесса и относящихся к разным *функциям* управления.

На внешнем уровне строится *структурная модель* предприятия в виде иерархии подчинения *организационных единиц* или списков взаимодействующих подразделений.

На концептуальном уровне для каждого подразделения задается организационно-штатная структура должностей (ролей персонала).

На внутреннем уровне определяются требования к правам доступа персонала к автоматизируемым *функциям* информационной системы.

Техническая структура

Топология определяет территориальное размещение технических средств по структурным подразделениям предприятия, а коммуникация — технический способ реализации взаимодействия структурных подразделений.

На внешнем уровне модели определяются типы технических средств обработки данных и их размещение по структурным подразделениям.

На концептуальном уровне определяются способы коммуникаций между техническими комплексами структурных подразделений: физическое перемещение документов, машинных носителей, обмен информацией по каналам связи и т.д.

На внутреннем уровне строится модель "клиент-серверной" архитектуры вычислительной сети.

Описанные *модели предметной области* нацелены на проектирование отдельных компонентов ИС: данных, функциональных программных модулей, управляющих программных модулей, программных модулей интерфейсов пользователей, структуры технического комплекса. Для более качественного проектирования указанных компонентов требуется построение моделей, увязывающих различные компоненты ИС между собой. В простейшем случае в качестве таких моделей взаимодействия могут использоваться матрицы перекрестных ссылок: "объекты-функции", "функции-события", "*организационные единицы — функции* ", "*организационные единицы* — объекты", "*организационные единицы* — технические средства" и т д. Такие матрицы не наглядны и не отражают особенности реализации взаимодействий.

Для правильного отображения взаимодействий компонентов ИС важно осуществлять совместное моделирование таких компонентов, особенно с содержательной точки зрения объектов и *функций*. Методология структурного *системного анализа* существенно помогает в решении таких задач.

Структурным анализом принято называть метод исследования предметной области, который начинается с ее общего обзора, а затем детализируется, приобретая иерархическую структуру с все большим числом уровней. Для таких методов характерно: разбиение на уровни абстракции с

ограниченным числом элементов (от 3 до 7); ограниченный контекст, включающий только существенные детали каждого уровня; использование строгих формальных правил записи; последовательное приближение к результату. *Структурный анализ* основан на двух базовых принципах – "разделяй и властвуй" и принципе иерархической упорядоченности. Решение трудных проблем путем их разбиения на множество меньших независимых задач (так называемых "черных ящиков") и организация этих задач в древовидные иерархические структуры значительно повышают понимание сложных систем. Определим ключевые понятия *структурного анализа*.

Операция – элементарное (неделимое в рамках решаемой задачи) действие,.

Функция – совокупность *операций*, сгруппированных по определенному признаку.

Бизнес-процесс — связанная совокупность *функций*, в ходе выполнения которой потребляются определенные ресурсы и создается продукт (предмет, услуга, научное открытие, идея), представляющая ценность для потребителя.

Подпроцесс – это *бизнес-процесс*, являющийся структурным элементом некоторого *бизнес-процесса* и представляющий ценность для потребителя.

Бизнес-модель – структурированное описание сети процессов и *операций*, связанных с данными, документами, организационными единицами и прочими объектами, отражающими существующую или предполагаемую деятельность предприятия.

Существуют различные методологии структурного моделирования предметной области, среди которых следует выделить *функционально-ориентированные и объектно-ориентированные методологии.*

Функционально-ориентированные методологии описания предметной области

Процесс моделирования исследуемой предметной области может быть реализован в рамках различных подходов (соответствующих методологий и методик), отличающихся представлением исследуемого объекта. Существующие подходы к моделированию можно разделить на функциональные (структурные), объектные, синтезированные.

Функциональный подход (наиболее известной является методика *IDEF0)*, в котором исследуемый объект рассматривается как набор *функций*, преобразующих поступающий поток информации в выходной поток. Процесс преобразования информации потребляет определенные ресурсы. Основной особенностью *функционального подхода* заключается в четком отделении *функций* (методов обработки данных) от самих данных.

Объектный подход, в котором исследуемый объект (предмет, процесс, явление) рассматривается как совокупность взаимосвязанных объектов, имеющих четко определяемое поведение.

Синтетический подход заключается в последовательном применении функционального и объектного

подхода с учетом возможности реинжиниринга существующей ситуации.

Функциональная методика IDEF0

Методику *IDEF0* можно считать следующим этапом развития хорошо известного графического языка описания функциональных систем *SADT* (*Structured Analysis* and *Design Technique*). Исторически *IDEF0* как стандарт был разработан в 1981 году в рамках обширной программы автоматизации промышленных

предприятий, которая носила обозначение *ICAM* (Integrated Computer Aided *Manufacturing*). Семейство стандартов *IDEF* унаследовало свое обозначение от названия этой программы (*IDEF=Icam* DEFinition), и последняя его редакция была выпущена в декабре 1993 года Национальным Институтом по Стандартам и Технологиям США (*NIST*).

Целью методики является построение *функциональной схемы* исследуемой системы, описывающей все необходимые процессы с точностью, достаточной для однозначного моделирования деятельности системы.

В основе методики используются четыре основных понятия: функциональный блок, интерфейсная дуга, декомпозиция, глоссарий.

Функциональный блок (Activity Box) представляет собой некоторую конкретную *функцию* в рамках рассматриваемой системы. По требованиям стандарта название каждого функционального блока должно быть сформулировано в глагольном наклонении (например, "производить услуги"). На диаграмме функциональный блок изображается прямоугольником (рис. 3.1). Каждая из четырех сторон функционального блока имеет свое определенное значение (роль), при этом:

- верхняя сторона имеет значение "Управление" (Control);
- левая сторона имеет значение "Вход" (Input);
- правая сторона имеет значение "Выход" (Output);
- нижняя сторона имеет значение "Механизм" (*Mechanism*).

Рис. 3.1. Функциональный блок

Интерфейсная дуга (Arrow) отображает элемент системы, который обрабатывается функциональным блоком или оказывает иное влияние на *функцию*, представленную данным функциональным блоком. Интерфейсные дуги часто называют потоками или стрелками.

С помощью интерфейсных дуг отображают различные объекты, в той или иной степени определяющие процессы, происходящие в системе. Такими объектами могут быть элементы реального мира (детали, вагоны, сотрудники и т.д.) или потоки данных и информации (документы, данные, инструкции и т.д.).

В зависимости от того, к какой из сторон функционального блока подходит данная интерфейсная дуга, она носит название "входящей", "исходящей" или "управляющей".

Необходимо отметить, что любой функциональный блок по требованиям стандарта должен иметь, по крайней мере, одну управляющую интерфейсную дугу и одну исходящую. Это и понятно – каждый процесс должен происходить по каким-то правилам (отображаемым управляющей дугой) и должен выдавать некоторый результат (выходящая дуга), иначе его рассмотрение не имеет никакого смысла.

Обязательное наличие управляющих интерфейсных дуг является одним из главных отличий стандарта *IDEF0* от других методологий классов *DFD* (*Data Flow Diagram*) и WFD (Work Flow Diagram).

Декомпозиция (*Decomposition*) является основным понятием стандарта *IDEF0*. Принцип декомпозиции применяется при разбиении сложного процесса на составляющие его *функции*. При этом *уровень детализации* процесса определяется непосредственно разработчиком модели.

Декомпозиция позволяет постепенно и структурировано представлять модель системы в виде иерархической структуры отдельных диаграмм, что делает ее менее перегруженной и легко усваиваемой.

Последним из понятий *IDEF0* является глоссарий (Glossary). Для каждого из элементов *IDEF0* — диаграмм, функциональных блоков, интерфейсных дуг — существующий стандарт подразумевает создание и поддержание набора соответствующих определений, ключевых слов, повествовательных изложений и т.д., которые характеризуют объект, отображенный данным элементом. Этот набор называется глоссарием и является описанием сущности данного элемента. Глоссарий гармонично дополняет наглядный графический язык, снабжая диаграммы необходимой дополнительной информацией.

Модель *IDEF0* всегда начинается с представления системы как единого целого – одного функционального блока с интерфейсными дугами, простирающимися за пределы рассматриваемой области. Такая диаграмма с одним функциональным блоком называется контекстной диаграммой.

В пояснительном тексте к *контекстной диаграмме* должна быть указана цель (Purpose) построения диаграммы в виде краткого описания и зафиксирована точка зрения (Viewpoint).

Определение и формализация цели разработки *IDEF0*-модели является крайне важным моментом. Фактически цель определяет соответствующие области в исследуемой системе, на которых необходимо фокусироваться в первую очередь.

Точка зрения определяет основное направление развития модели и уровень необходимой детализации. Четкое фиксирование точки зрения позволяет разгрузить модель, отказавшись от детализации и исследования отдельных элементов, не являющихся необходимыми, исходя из выбранной точки зрения на систему. Правильный выбор точки зрения существенно сокращает временные затраты на построение конечной модели.

Выделение *подпроцессов*. В процессе декомпозиции функциональный блок, который в *контекстной диаграмме* отображает систему как единое целое, подвергается детализации на другой диаграмме. Получившаяся диаграмма второго уровня содержит функциональные блоки, отображающие главные подфункции функционального блока *контекстной диаграммы*, и называется дочерней (Child Diagram) по отношению к нему (каждый из функциональных блоков, принадлежащих дочерней диаграмме, соответственно называется дочерним блоком – Child Box). В свою очередь, функциональный блок — предок называется родительским блоком по отношению к дочерней диаграмме (Parent Box), а диаграмма, к которой он принадлежит – родительской диаграммой (Parent Diagram). Каждая из подфункций дочерней диаграммы может быть далее детализирована путем аналогичной декомпозиции соответствующего ей функционального блока. В каждом случае декомпозиции функционального блока все интерфейсные дуги, входящие в данный блок или исходящие из него, фиксируются на дочерней диаграмме. Этим достигается структурная целостность *IDEF0*–модели.

Иногда отдельные интерфейсные дуги высшего уровня не имеет смысла продолжать рассматривать на диаграммах нижнего уровня, или наоборот — отдельные дуги нижнего отражать на диаграммах более высоких уровней – это будет только перегружать диаграммы и делать их сложными для восприятия. Для решения подобных задач в стандарте *IDEF0* предусмотрено понятие *туннелирования*. Обозначение "*туннеля*" (Arrow *Tunnel*) в виде двух круглых скобок вокруг начала интерфейсной дуги обозначает, что эта дуга не была унаследована от функционального родительского блока и появилась (из "*туннеля*") только на этой диаграмме. В свою очередь, такое же обозначение вокруг конца (стрелки) интерфейсной

дуги в непосредственной близи от блока–приемника означает тот факт, что в дочерней по отношению к этому блоку диаграмме эта дуга отображаться и рассматриваться не будет. Чаще всего бывает, что отдельные объекты и соответствующие им интерфейсные дуги не рассматриваются на некоторых промежуточных уровнях иерархии, – в таком случае они сначала "погружаются в *туннель*", а затем при необходимости "возвращаются из *туннеля*".

Обычно *IDEF0*-модели несут в себе сложную и концентрированную информацию, и для того, чтобы ограничить их перегруженность и сделать *удобочитаемыми*, в стандарте приняты соответствующие ограничения сложности.

Рекомендуется представлять на диаграмме от трех до шести функциональных блоков, при этом количество подходящих к одному функциональному блоку (выходящих из одного функционального блока) интерфейсных дуг предполагается не более четырех.

Стандарт *IDEF0* содержит набор процедур, позволяющих разрабатывать и согласовывать модель большой группой людей, принадлежащих к разным областям деятельности моделируемой системы. Обычно процесс разработки является итеративным и состоит из условных 3-х этапов:

1) Создание модели группой специалистов, относящихся к различным сферам деятельности предприятия. Эта группа в терминах *IDEF0* называется авторами (Authors). Построение первоначальной модели является динамическим процессом, в течение которого авторы опрашивают компетентных лиц о структуре различных процессов, создавая модели деятельности подразделений. При этом их интересуют ответы на следующие вопросы:

Что поступает в подразделение "на входе"?

Какие *функции* и в какой последовательности выполняются в рамках подразделения?
Кто является ответственным за выполнение каждой из *функций*?
Чем руководствуется исполнитель при выполнении каждой из *функций*?
Что является результатом работы подразделения (на выходе)?

На основе имеющихся положений, документов и результатов опросов создается черновик (Model Draft) модели.

2) Распространение черновика для рассмотрения, согласований и комментариев. На этой стадии происходит обсуждение черновика модели с широким кругом компетентных лиц (в терминах *IDEF0* — читателей) на предприятии. При этом каждая из диаграмм черновой модели письменно критикуется и комментируется, а затем передается автору. Автор, в свою очередь, также письменно соглашается с критикой или отвергает ее с изложением логики принятия решения и вновь возвращает откорректированный черновик для дальнейшего рассмотрения. Этот цикл продолжается до тех пор, пока авторы и читатели не придут к единому мнению.

3) Официальное утверждение модели. Утверждение согласованной модели происходит руководителем рабочей группы в том случае, если у авторов модели и читателей отсутствуют разногласия по поводу ее адекватности. Окончательная модель представляет собой согласованное представление о предприятии (системе) с заданной точки зрения и для заданной цели.

Наглядность графического языка *IDEF0* делает модель вполне читаемой и для лиц, которые не принимали участия в проекте ее создания, а также эффективной для проведения показов и презентаций. В дальнейшем на базе построенной модели могут быть организованы новые проекты, нацеленные на производство изменений в модели.

Ниже приводятся примеры функциональных моделей в нотации IDEF0 (рис. 2.1, 2.2) для информационной системы, обеспечивающей обработку заявок студента на получение учебных изданий в библиотеке университета.

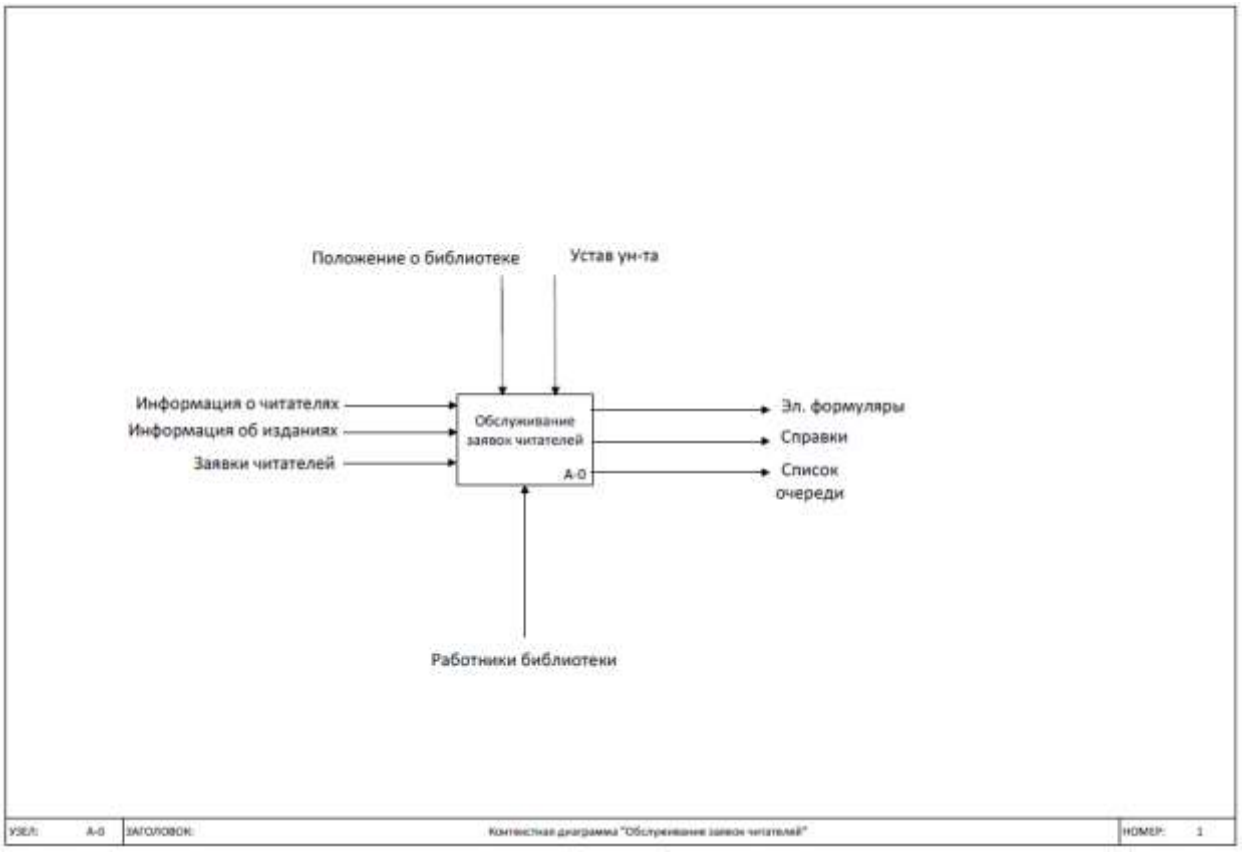


Рис. 2.1

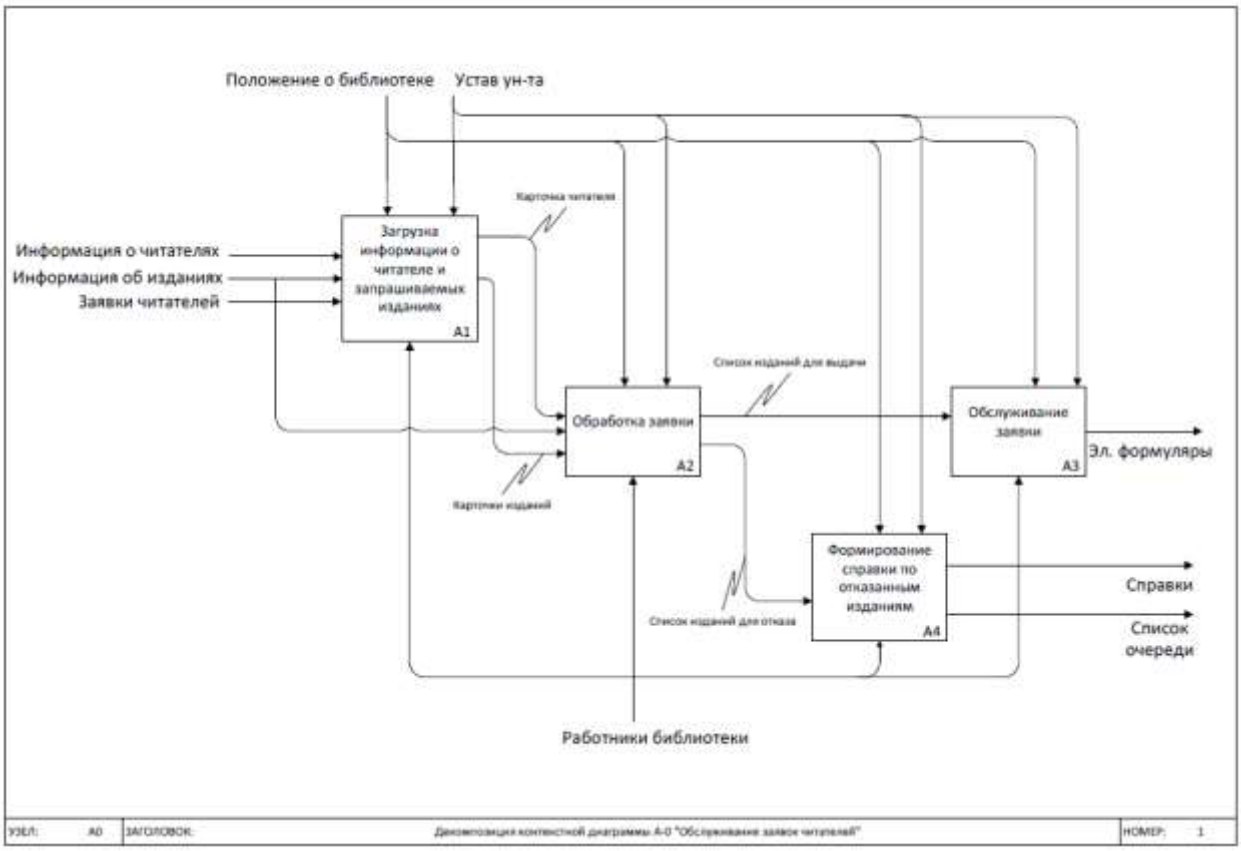


Рис. 2.2.

Моделирование деловых процессов, как правило, выполняется с помощью case-средств. К таким средствам относятся BPwin (PLATINUM technology), Silverrun (Silverrun technology), Oracle Designer (Oracle), Rational Rose (Rational Software) и др. Средства моделирования предметной области также называются инструментальными средствами структурного моделирования.

Функциональная методика потоков данных

Целью методики является построение модели рассматриваемой системы в виде *диаграммы потоков данных* (*Data Flow Diagram — DFD*), обеспечивающей правильное описание выходов (отклика системы в виде данных) при заданном воздействии на вход системы (подаче сигналов через внешние интерфейсы). *Диаграммы потоков данных* являются основным средством моделирования *функциональных требований* к проектируемой системе.

При создании *диаграммы потоков данных* используются четыре основных понятия: потоки данных, процессы (работы) преобразования входных потоков данных в выходные, внешние сущности, накопители данных (хранилища).

Потоки данных являются абстракциями, использующимися для моделирования передачи информации (или физических компонент) из одной части системы в другую. Потоки на диаграммах изображаются именованными стрелками, ориентация которых указывает направление движения информации.

Назначение процесса (работы) состоит в продуцировании выходных потоков из входных в соответствии с действием, задаваемым именем процесса. Имя процесса должно содержать глагол в неопределенной форме с последующим дополнением (например, "получить документы по отгрузке продукции"). Каждый процесс имеет уникальный номер для ссылок на него внутри диаграммы, который может использоваться совместно с номером диаграммы для получения *уникального индекса* процесса во всей модели.

Хранилище (накопитель) данных позволяет на указанных участках определять данные, которые будут сохраняться в памяти между процессами. Фактически хранилище представляет "срезы" потоков данных во времени. Информация, которую оно содержит, может использоваться в любое время после ее получения, при этом данные могут выбираться в любом порядке. Имя хранилища должно определять его содержимое и быть существительным.

Внешняя сущность представляет собой материальный объект вне контекста системы, являющейся источником или приемником системных данных. Ее имя должно содержать существительное, например, "склад товаров". Предполагается, что объекты, представленные как *внешние сущности*, не должны участвовать ни в какой обработке.

Кроме основных элементов, в состав *DFD* входят словари данных и миниспецификации.

Словари данных являются каталогами всех элементов данных, присутствующих в *DFD*, включая групповые и индивидуальные потоки данных, хранилища и процессы, а также все их атрибуты.

Миниспецификации обработки — описывают *DFD*-процессы нижнего уровня, представляют собой алгоритмы описания задач, выполняемых процессами.

Информационная система принимает извне потоки данных. Для обозначения элементов среды функционирования системы используется понятие внешней сущности. Внутри системы существуют процессы преобразования информации, порождающие новые потоки данных. Потоки данных могут поступать на вход к другим процессам, помещаться (и извлекаться) в накопители данных, передаваться к внешним сущностям.

Модель DFD, как и большинство других структурных моделей — иерархическая модель. Каждый процесс может быть подвергнут декомпозиции, то есть разбиению на структурные составляющие, отношения между которыми в той же нотации могут быть показаны на отдельной диаграмме. Когда достигнута требуемая глубина декомпозиции — процесс нижнего уровня сопровождается мини-спецификацией (текстовым описанием).

Кроме того, нотация DFD поддерживает понятие подсистемы — структурного компонента разрабатываемой системы.

Нотация DFD — удобное средство для формирования контекстной диаграммы, то есть диаграммы, показывающей разрабатываемую АИС в коммуникации с внешней средой. Это — диаграмма верхнего уровня в иерархии диаграмм DFD. Ее назначение — ограничить рамки системы, определить, где заканчивается разрабатываемая система и начинается среда.

Для всех *внешних сущностей* строится *таблица событий*, с помощью которой отражается взаимосвязь сущностей и событий с основным процессом. *Таблица событий* включает в себя наименование внешней сущности, событие, с которым сущность связана, тип события (типичный для системы или исключительный, реализующийся при определенных условиях) и реакцию системы.

На следующем шаге происходит декомпозиция основного процесса на набор взаимосвязанных процессов, обменивающихся потоками данных. Сами потоки не конкретизируются, определяется лишь характер взаимодействия. Декомпозиция завершается, когда процесс становится простым, т.е.:

- процесс имеет два-три входных и выходных потока;
- процесс может быть описан в виде преобразования входных данных в выходные;
- процесс может быть описан в виде *последовательного алгоритма*.

Для простых процессов строится миниспецификация – формальное описание алгоритма преобразования входных данных в выходные.

После декомпозиции основного процесса для каждого *подпроцесса* строится аналогичная таблица *внутренних событий*.

Следующим шагом после определения полной *таблицы событий* выделяются **потоки данных,** которыми обмениваются процессы и *внешние сущности*. Простейший способ их выделения заключается в анализе таблиц событий. События преобразуются в потоки данных от инициатора события к запрашиваемому процессу, а реакции – в обратный поток событий. После построения входных и выходных потоков аналогичным образом строятся внутренние потоки. Для их выделения для каждого из внутренних процессов выделяются поставщики и потребители информации. Если поставщик или потребитель информации представляет процесс сохранения или запроса информации, то вводится хранилище данных, для которого данный процесс является интерфейсом.

После построения потоков данных диаграмма должна быть проверена на полноту и непротиворечивость. Полнота диаграммы обеспечивается, если в системе нет "повисших" процессов, не используемых в процессе преобразования входных потоков в выходные. Непротиворечивость системы обеспечивается: на диаграмме не может быть потока, связывающего две *внешние сущности* – это взаимодействие удаляется из рассмотрения; ни одна сущность не может непосредственно получать или отдавать информацию в хранилище данных – хранилище данных является пассивным элементом, управляемым с помощью интерфейсного процесса; два хранилища данных не могут непосредственно обмениваться информацией – эти хранилища должны быть объединены.

DFD – диаграмма — один из основных инструментов структурного анализа и проектирования информационных систем, существовавших до широкого распространения UML. Несмотря на имеющее место в современных условиях смещение акцентов от структурного к объектно-ориентированному подходу к анализу и проектированию систем, "старинные" структурные нотации по-прежнему широко и эффективно используются как в бизнес-анализе, так и в анализе информационных систем.

К преимуществам методики *DFD* относятся:

- возможность однозначно определить *внешние сущности*, анализируя потоки информации внутри и вне системы;
- возможность проектирования сверху вниз, что облегчает построение модели "как должно быть";
- наличие спецификаций процессов нижнего уровня, что позволяет преодолеть логическую незавершенность *функциональной модели* и построить полную *функциональную спецификацию* разрабатываемой системы.

Исторически сложилось так, что для описания диаграмм DFD используются две нотации — Йордана (Yourdon) и Гейна-Сарсона (Gane-Sarson), отличающиеся синтаксисом.

Построение диаграммы потоков данных (DFD)

В основе функциональной методики потоков данных лежит построение диаграммы *DFD* для анализируемой ИС - проектируемой или реально существующей. В соответствии с методикой модель системы определяется как иерархия диаграмм потоков данных, описывающих асинхронный процесс преобразования информации от ее ввода в систему до выдачи пользователю. Диаграммы верхних уровней иерархии (контекстные диаграммы) определяют основные процессы или подсистемы ИС с внешними входами и выходами. Они детализируются при помощи диаграмм нижнего уровня. Такая декомпозиция продолжается, создавая многоуровневую иерархию диаграмм, до тех пор, пока не будет

достигнут такой уровень декомпозиции, на котором процессы становятся элементарными и детализировать их далее невозможно или не имеет смысла.

Источники информации (внешние сущности) порождают информационные потоки (потоки данных), переносящие информацию к подсистемам или процессам. Те в свою очередь преобразуют информацию и порождают новые потоки, которые переносят информацию к другим процессам или подсистемам, накопителям данных или внешним сущностям - потребителям информации.

Внешние сущности

Внешняя сущность представляет собой материальный предмет или физическое лицо, представляющее собой источник или приемник информации, например, заказчики, персонал, поставщики, клиенты, склад. Определение некоторого объекта или системы в качестве внешней сущности указывает на то, что она находится за пределами границ анализируемой ИС. В процессе анализа некоторые внешние сущности могут быть перенесены внутрь диаграммы анализируемой ИС, если это необходимо, или, наоборот, часть процессов ИС может быть вынесена за пределы диаграммы и представлена как внешняя сущность.

Внешняя сущность обозначается квадратом (рисунок 4.1) с затемненными 2-мя сторонами для того, чтобы можно было выделить этот объект среди других обозначений:

Рис. 2.4. Внешняя сущность

Системы и подсистемы

При построении модели сложной ИС она может быть представлена в самом общем виде на так называемой контекстной диаграмме в виде одной системы как единого целого, либо может быть декомпозирована на ряд подсистем. Подсистема (или система) на контекстной диаграмме изображается следующим образом (рисунок 4.2).

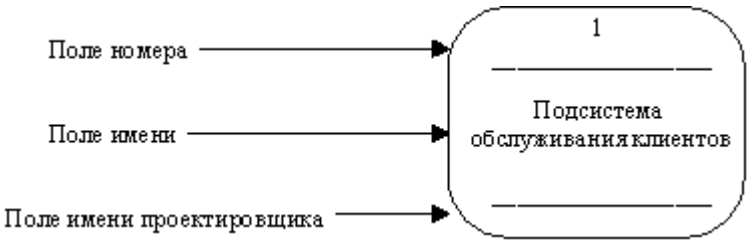


Рис. 2.5. Подсистема

Номер подсистемы служит для ее идентификации. В поле имени вводится наименование подсистемы в виде предложения с подлежащим и соответствующими определениями и дополнениями.

Процессы

Процесс представляет собой преобразование входных потоков данных в выходные в соответствии с определенным алгоритмом. Физически процесс может быть реализован различными способами: это может быть подразделение организации (отдел), выполняющее обработку входных документов и выпуск

отчетов, программа, аппаратно реализованное логическое устройство и т.д. Процесс на диаграмме потоков данных изображается, как показано на рисунке 4.3.

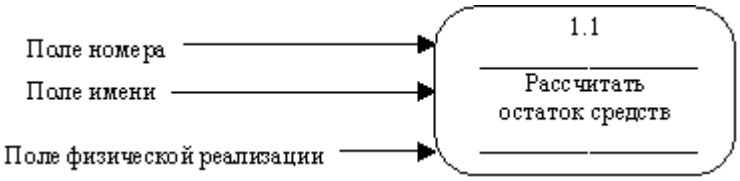


Рис. 2.6. Процесс

Номер процесса служит для его идентификации. В поле имени вводится наименование процесса в виде предложения с активным недвусмысленным глаголом в неопределенной форме (вычислить, рассчитать, проверить, определить, создать, получить), за которым следуют существительные в винительном падеже, например:

- "Ввести сведения о клиентах";
- "Выдать информацию о текущих расходах";
- "Проверить кредитоспособность клиента".

Использование таких глаголов, как "обработать", "модернизировать" или "отредактировать" означает, как правило, недостаточно глубокое понимание данного процесса и требует дальнейшего анализа.

Информация в поле физической реализации показывает, какое подразделение организации, программа или аппаратное устройство выполняет данный процесс.

Накопители данных

Накопитель данных представляет собой 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 ящика в картотеке, таблицы в оперативной памяти, файла на магнитном носителе и т.д. Накопитель данных на диаграмме потоков данных изображается, как показано на рисунке 4.4.

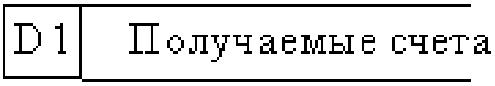


Рис. 2.7. Накопитель данных

Накопитель данных идентифицируется буквой "D" и произвольным числом. Имя накопителя выбирается из соображения наибольшей информативности для проектировщика.

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

Потоки данных

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 и т.д.

Поток данных на диаграмме изображается линией, оканчивающейся стрелкой, которая показывает направление потока (рисунок 4.5). Каждый поток данных имеет имя, отражающее его содержание.

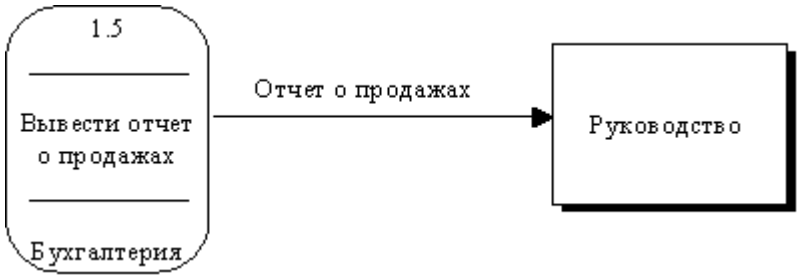


Рис. 2.8. Поток данных

Построение иерархии диаграмм потоков данных

Первым шагом при построении иерархии *DFD* является построение контекстных диаграмм. Обычно при проектировании относительно простых ИС строится единственная контекстная диаграмма со звездообразной топологией, в центре которой находится так называемый главный процесс, соединенный с приемниками и источниками информации, посредством которых с системой взаимодействуют пользователи и другие внешние системы.

Если же для сложной системы ограничиться единственной контекстной диаграммой, то она будет содержать слишком большое количество источников и приемников информации, которые трудно расположить на листе бумаги нормального формата, и кроме того, единственный главный процесс не раскрывает структуры распределенной системы. Признаками сложности (в смысле контекста) могут быть:

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

Для сложных ИС строится иерархия контекстных диаграмм. При этом контекстная диаграмма верхнего уровня содержит не единственный главный процесс, а набор подсистем, соединенных потоками данных. Контекстные диаграммы следующего уровня детализируют контекст и структуру подсистем.

Иерархия контекстных диаграмм определяет взаимодействие основных функциональных подсистем проектируемой ИС как между собой, так и с внешними входными и выходными потоками данных и внешними объектами (источниками и приемниками информации), с которыми взаимодействует ИС.

Разработка контекстных диаграмм решает проблему строгого определения функциональной структуры ИС на самой ранней стадии ее проектирования, что особенно важно для сложных многофункциональных систем, в разработке которых участвуют разные организации и коллективы разработчиков.

После построения контекстных диаграмм полученную модель следует проверить на полноту исходных данных об объектах системы и изолированность объектов (отсутствие информационных связей с другими объектами).

Для каждой подсистемы, присутствующей на контекстных диаграммах, выполняется ее детализация при помощи *DFD*. Каждый процесс на *DFD*, в свою очередь, может быть детализирован при помощи *DFD* или миниспецификации. 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 (описание логики процесса) должна формулировать его основные функции таким образом, чтобы в дальнейшем специалист, выполняющий реализацию проекта, смог выполнить их или разработать соответствующую программу.

Миниспецификация является конечной вершиной иерархии *DFD*. Решение о завершении детализации процесса и использовании миниспецификации принимается аналитиком исходя из следующих критериев:

- наличия у процесса относительно небольшого количества входных и выходных потоков данных (2-3 потока);
- 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).

При построении иерархии *DFD* переходить к детализации процессов следует только после определения содержания всех потоков и накопителей данных, которое описывается при помощи структур данных. Структуры данных конструируются из элементов данных и могут содержать альтернативы, условные вхождения и итерации. Условное вхождение означает, что данный компонент может отсутствовать в структуре. Альтернатива означает, что в структуру может входить один из перечисленных элементов. Итерация означает вхождение любого числа элементов в указанном диапазоне. Для каждого элемента данных может указываться его тип (непрерывные или дискретные данные). Для непрерывных данных может указываться единица измерения (кг, см и т.п.), диапазон значений, точность представления и форма физического кодирования. Для дискретных данных может указываться таблица допустимых значений.

Пример DFD – диаграммы бизнес-процесса А2-Обработка заявки читателя библиотеки показан на рисунке 2.2.

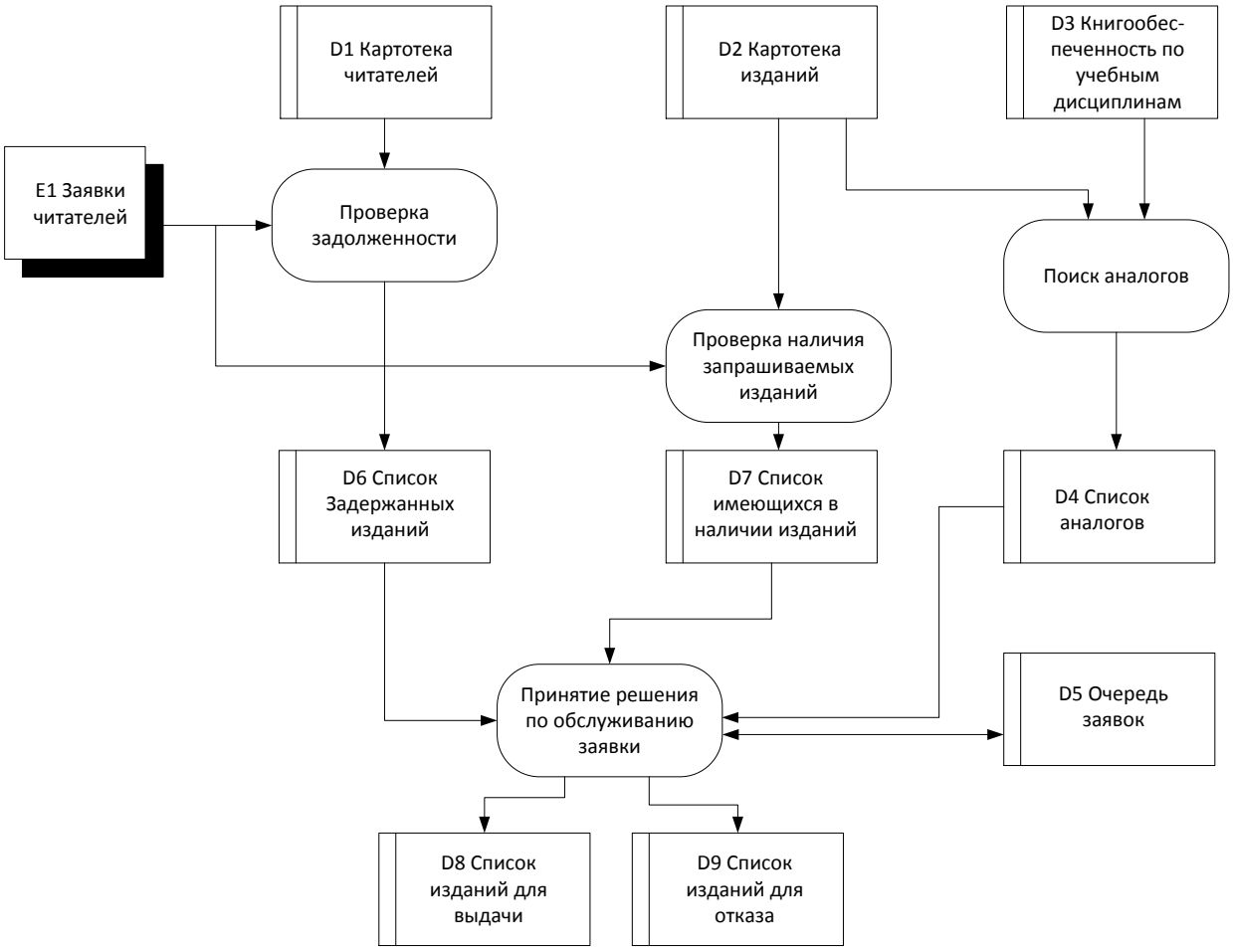


Рис. 2.9. DFD – диаграмма бизнес-процесса "Обработка заявки читателя библиотеки" (диаграмма A2)

После построения законченной модели системы ее необходимо верифицировать (проверить на полноту и согласованность). В полной модели все ее объекты (подсистемы, процессы, потоки данных) должны быть подробно описаны и детализированы. Выявленные не детализированные объекты следует детализировать, вернувшись на предыдущие шаги разработки. 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 записаны.

Метод описания бизнес-процессов IDEF3

Наличие в диаграммах *DFD* элементов для обозначения *источников*, приемников и *хранилищ данных* позволяет более эффективно и наглядно описать *процесс* документооборота. Однако для описания логики взаимодействия информационных потоков более подходит метод IDEF3, называемый также Workflow diagramming. Метод IDEF3 использует графическое описание информационных потоков, взаимоотношений между *процессами* обработки информации и объектов, являющихся частью этих *процессов*. Диаграммы Workflow могут быть использованы в моделировании бизнес-процессов для анализа завершенности процедур обработки информации. С их помощью можно описывать сценарии действий сотрудников организации, например последовательность обработки заказа или события, которые необходимо обработать за конечное время. Каждый сценарий сопровождается описанием *процесса* и может быть использован для документирования каждой функции.

IDEF3 — это метод, имеющий основной целью дать возможность аналитикам описать ситуацию, когда *процессы* выполняются в определенной последовательности, а также описать объекты, участвующие совместно в одном *процессе.*

Техника описания набора данных IDEF3 является частью *структурного анализа*. В отличие от некоторых методик описаний *процессов* IDEF3 не ограничивает аналитика чрезмерно жесткими рамками синтаксиса, что может привести к созданию неполных или противоречивых моделей.

IDEF3 может быть также использован как метод создания *процессов*. IDEF3 дополняет *IDEF0* и содержит все необходимое для построения моделей, которые в дальнейшем могут быть использованы для имитационного анализа.

Каждая работа в IDEF3 описывает какой-либо сценарий бизнес-процесса и может являться составляющей другой работы. Поскольку сценарий описывает цель и рамки модели, важно, чтобы работы именовались отглагольным существительным, обозначающим *процесс* действия, или фразой, содержащей такое существительное.

Точка зрения на модель должна быть документирована. Обычно это точка зрения человека, ответственного за работу в целом. Также необходимо документировать цель модели — те вопросы, на которые призвана ответить модель.

Диаграмма является основной единицей описания в IDEF3. Важно правильно построить диаграммы, поскольку они предназначены для чтения другими людьми (а не только автором).

Единицы работы — Unit of Work (UOW) — также называемые работами (activity), являются центральными компонентами модели. В IDEF3 работы изображаются прямоугольниками с прямыми углами и имеют имя, выраженное отглагольным существительным, обозначающим *процесс* действия, одиночным или в составе фразы, и номер (идентификатор); другое имя существительное в составе той же фразы обычно отображает основной выход (результат) работы (например, "Изготовление изделия"). Часто имя существительное в имени работы меняется в *процессе* моделирования, поскольку модель может уточняться и редактироваться. Идентификатор работы присваивается при создании и не меняется никогда. Даже если работа будет удалена, ее идентификатор не будет вновь использоваться для других работ. Обычно номер работы состоит из номера родительской работы и порядкового номера на текущей диаграмме.

Связи показывают взаимоотношения работ. Все *связи* в IDEF3 однонаправлены и могут быть направлены куда угодно, но обычно диаграммы IDEF3 стараются построить так, чтобы *связи* были направлены слева направо. В IDEF3 различают три типа стрелок, изображающих *связи*, стиль которых устанавливается через меню Edit/Arrow Style:

Старшая (Precedence)

сплошная линия, связывающая единицы работ (UOW). Рисуется слева направо или сверху вниз. Показывает, что работа-источник должна закончиться прежде, чем работа-цель начнется.

Отношения (Relational Link)

пунктирная линия, использующаяся для изображения *связей* между единицами работ (UOW) а также между единицами работ и объектами ссылок.

Потоки объектов (Object Flow)

стрелка с двумя наконечниками, применяется для описания того факта, что объект используется в двух или более единицах работы, например, когда объект порождается в одной работе и используется в другой.

Старшая *связь* показывает, что работа-источник заканчивается ранее, чем начинается работа-цель. Часто результатом работы-источника становится объект, необходимый для запуска работы-цели. В этом случае стрелку, обозначающую объект, изображают с двойным наконечником. Имя стрелки должно ясно идентифицировать отображаемый объект. Поток объектов имеет ту же семантику, что и старшая стрелка.

Отношение показывает, что стрелка является альтернативой старшей стрелке или *потоку объектов* в смысле задания последовательности выполнения работ — работа-источник не обязательно должна закончиться, прежде чем работа-цель начнется. Более того, работа-цель может закончиться прежде, чем закончится работа-источник.

Окончание одной работы может служить сигналом к началу нескольких работ, или же одна работа для своего запуска может ожидать окончания нескольких работ. Для отображения логики взаимодействия стрелок при слиянии и разветвлении или для отображения множества событий, которые могут или должны быть завершены перед началом следующей работы, используются *перекрестки* (Junction). Различают *перекрестки* для слияния (*Fan*-in *Junction*) и разветвления стрелок (*Fan*-out *Junction*). Перекресток не может использоваться одновременно для слияния и для разветвления.

Смысл каждого типа приведен ниже в таблице 2.1.

Таблица 2.1 - Типы перекрестков

Обозначение	Наименование	Смысл в случае слияния стрелок (*Fan*-in *Junction*)	Смысл в случае разветвления стрелок (*Fan*-out *Junction*)
	*Asynchronous* AND	Все предшествующие *процессы* должны быть завершены	Все следующие *процессы* должны быть запущены
	Synchronous AND	Все предшествующие *процессы* завершены одновременно	Все следующие *процессы* запускаются одновременно
	*Asynchronous* OR	Один или несколько предшествующих *процессов* должны быть завершены	Один или несколько следующих *процессов* должны быть запущены
	Synchronous OR	Один или несколько предшествующих *процессов* завершены одновременно	Один или несколько следующих *процессов* запускаются одновременно
	XOR (Exclusive OR)	Только один предшествующий *процесс* завершен	Только один следующий *процесс* запускается

При внесении объектов ссылок помимо имени следует указывать тип объекта ссылки. Типы объектов ссылок приведены в таблице 2.2.

Таблица 2.2 - Типы объектов ссылок

Тип объекта ссылки	**Цель описания**
OBJECT	Описывает участие важного объекта в работе
GOTO	Инструмент циклического перехода (в повторяющейся последовательности работ), возможно на текущей диаграмме, но не обязательно. Если все работы цикла присутствуют на текущей диаграмме, цикл может также изображаться стрелкой, возвращающейся на стартовую работу. GOTO может ссылаться на перекресток
UOB (Unit of behaviour)	Применяется, когда необходимо подчеркнуть множественное использование какой-либо работы, но без цикла. Например, работа "Контроль качества" может быть использована в *процессе* "Изготовление изделия" несколько раз, после каждой единичной операции. Обычно этот тип ссылки не используется для моделирования автоматически запускающихся работ
NOTE	Используется для документирования важной информации, относящейся к каким-либо графическим объектам на диаграмме. NOTE является альтернативой внесению текстового объекта в диаграмму
ELAB (Elaboration)	Используется для усовершенствования графиков или их более детального описания. Обычно употребляется для детального описания разветвления и слияния стрелок на *перекрестках*

В IDEF3 **декомпозиция** используется для детализации работ. Методология IDEF3 позволяет декомпозировать работу многократно, т. е. работа может иметь множество дочерних работ. Это позволяет в одной модели описать альтернативные потоки. Возможность множественной декомпозиции предъявляет дополнительные требования к нумерации работ. Так, номер работы состоит из номера родительской работы, версии декомпозиции и собственного номера работы на текущей диаграмме.

Рассмотрим *процесс* декомпозиции диаграмм IDEF3, включающий взаимодействие автора (аналитика) и одного или нескольких экспертов предметной области.

Перед проведением сеанса экспертизы у экспертов предметной области должны быть документированные сценарии и рамки модели, для того чтобы понять цели декомпозиции. Обычно эксперт предметной области передает аналитику текстовое описание сценария. В дополнение к этому может существовать документация, описывающая интересующие *процессы*. Из этой информации аналитик должен составить предварительный список работ (отглагольные существительные, обозначающие *процесс*) и объектов (существительные, обозначающие результат выполнения работы), которые необходимы для перечисленных работ. В некоторых случаях целесообразно создать графическую модель для представления ее эксперту предметной области.

На рисунке 2.10 представлено описание *процесса* "Сборка настольных компьютеров" в методологии IDEF3.

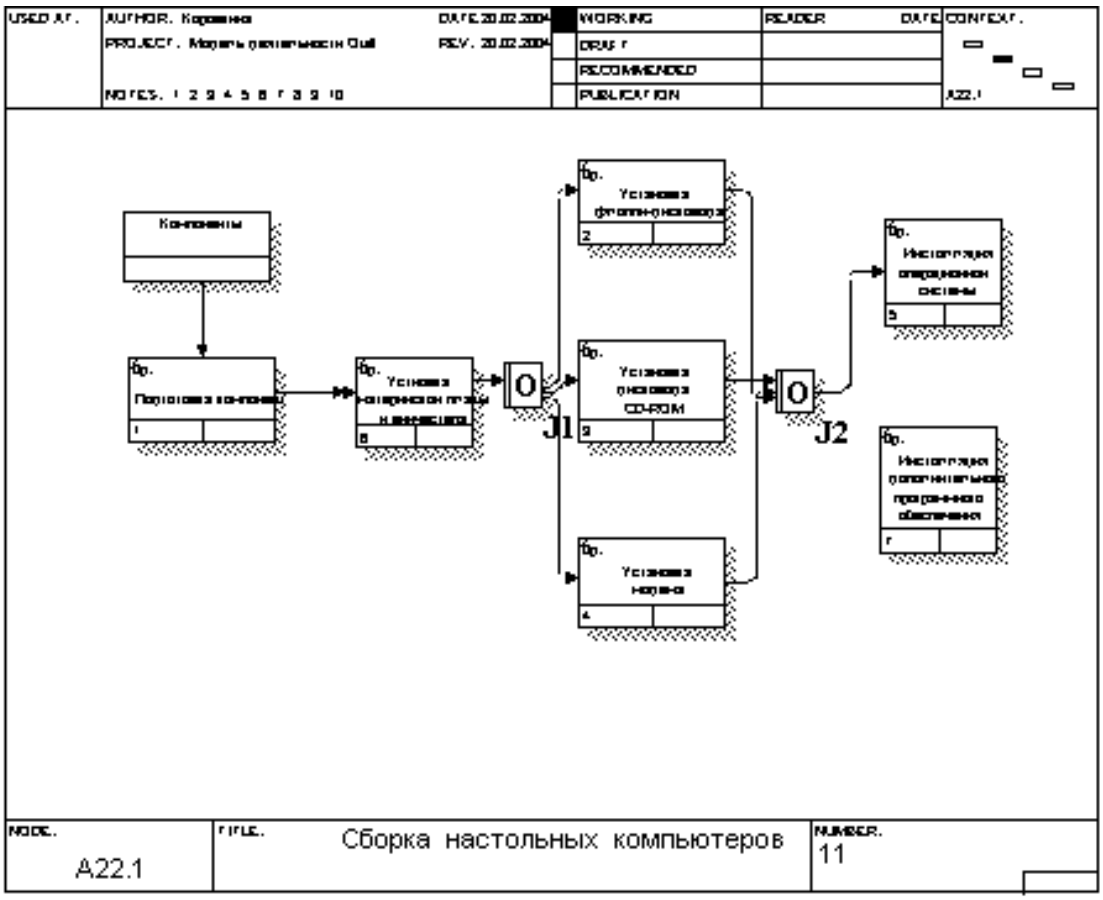


Рис. 2.10. Пример описания бизнес-процесса по методу IDEF3

Информационное обеспечение ИС

Информационное обеспечение ИС является средством для решения следующих задач:

– однозначного и экономичного представления информации в системе (на основе кодирования объектов);

– организации процедур анализа и обработки информации с учетом характера связей между объектами (на основе *классификации* объектов);

– организации взаимодействия пользователей с системой (на основе *экранных форм* ввода-вывода данных);

– обеспечения эффективного использования информации в контуре управления деятельностью объекта автоматизации (на основе *системы документации*).

Информационное обеспечение ИС включает два комплекса:

- внемашинное информационное обеспечение (*классификаторы* технико-экономической информации, документы, методические инструктивные материалы);

- внутримашинное информационное обеспечение (макеты/экранные формы для ввода первичных данных в ЭВМ или вывода результатной информации, структуры *информационной базы*: входных, выходных файлов, базы данных).

К информационному обеспечению предъявляются следующие общие требования:

– информационное обеспечение должно быть достаточным для поддержания всех автоматизируемых функций объекта;
– для кодирования информации должны использоваться принятые у заказчика *классификаторы* ;
– для кодирования входной и *выходной информации*, которая используется на высшем уровне управления, должны быть использованы *классификаторы* этого уровня;
– должна быть обеспечена совместимость с информационным обеспечением систем, взаимодействующих с разрабатываемой системой;
– формы документов должны отвечать требованиям корпоративных стандартов заказчика (или унифицированной *системы документации*);
– структура документов и *экранных форм* должна соответствовать характеристиками терминалов на рабочих местах конечных пользователей;
– графики формирования и содержание *информационных сообщений*, а также используемые аббревиатуры должны быть общеприняты в этой предметной области и согласованы с заказчиком;
– в ИС должны быть предусмотрены средства контроля входной и результатной информации, обновления данных в информационных массивах, *контроля целостности информационной базы*, защиты от несанкционированного доступа.

Информационное обеспечение ИС ***можно определить как совокупность единой*** *системы классификации****, унифицированной*** *системы документации* **и** *информационной базы***.**

Внемашинное информационное обеспечение

Основные понятия классификации технико-экономической информации.

Для того чтобы обеспечить эффективный поиск, обработку на ЭВМ и передачу по каналам связи технико-экономической информации, ее необходимо представить в цифровом виде. С этой целью ее нужно сначала упорядочить (классифицировать), а затем формализовать (закодировать) с использованием *классификатора*.

Классификация – это *разделение множества* объектов на подмножества по их сходству или различию в соответствии с принятыми методами. *Классификация* фиксирует закономерные связи между классами объектов. Под объектом понимается любой предмет, процесс, явление материального или нематериального свойства. *Система классификации* позволяет сгруппировать объекты и выделить определенные классы, которые будут характеризоваться рядом общих свойств. Таким образом, совокупность правил распределения объектов множества на подмножества называется *системой классификации* .

Свойство или характеристика объекта *классификации*, которое позволяет установить его сходство или различие с другими объектами *классификации*, называется признаком *классификации*. Например, признак "роль предприятия-партнера в отношении деятельности объекта автоматизации" позволяет разделить все предприятия на две группы (на два подмножества): "поставщики" и "потребители". Множество или подмножество, объединяющее часть объектов *классификации* по одному или нескольким признакам, носит название классификационной группировки.

Классификатор — это документ, с помощью которого осуществляется формализованное описание информации в ИС, содержащей наименования объектов, наименования классификационных группировок и их кодовые обозначения [21].

По сфере действия выделяют следующие виды *классификаторов*: международные, общегосударственные (общесистемные), отраслевые и локальные *классификаторы*.

Международные *классификаторы* входят в состав Системы международных экономических стандартов (СМЭС) и обязательны для передачи информации между организациями разных стран мирового сообщества.

Общегосударственные (общесистемные) *классификаторы*, обязательны для организации процессов передачи и обработки информации между экономическими системами государственного уровня внутри страны.

Отраслевые *классификаторы* используют для выполнения процедур обработки информации и передачи ее между организациями внутри отрасли.

Локальные *классификаторы* используют в пределах отдельных предприятий.

Каждая *система классификации* характеризуется следующими свойствами:

- гибкостью системы;
- емкостью системы;
- степенью заполненности системы.

Гибкость системы — это способность допускать включение новых признаков, объектов без разрушения структуры *классификатора*. Необходимая гибкость определяется временем жизни системы.

Емкость системы — это наибольшее количество классификационных группировок, допускаемое в данной *системе классификации*.

Степень заполненности системы определяется как частное от деления фактического количества группировок на величину емкости системы.

В настоящее время чаще всего применяются два типа *систем классификации*: иерархическая и многоаспектная.

При использовании иерархического метода *классификации* происходит "последовательное *разделение множества* объектов на подчиненные, зависимые классификационные группировки". Получаемая на основе этого процесса классификационная схема имеет иерархическую структуру. В ней первоначальный объем классифицируемых объектов разбивается на подмножества по какому-либо признаку и детализируется на каждой следующей ступени *классификации*. Обобщенное изображение иерархической *классификационной схемы* представлено на рис. 9.1.

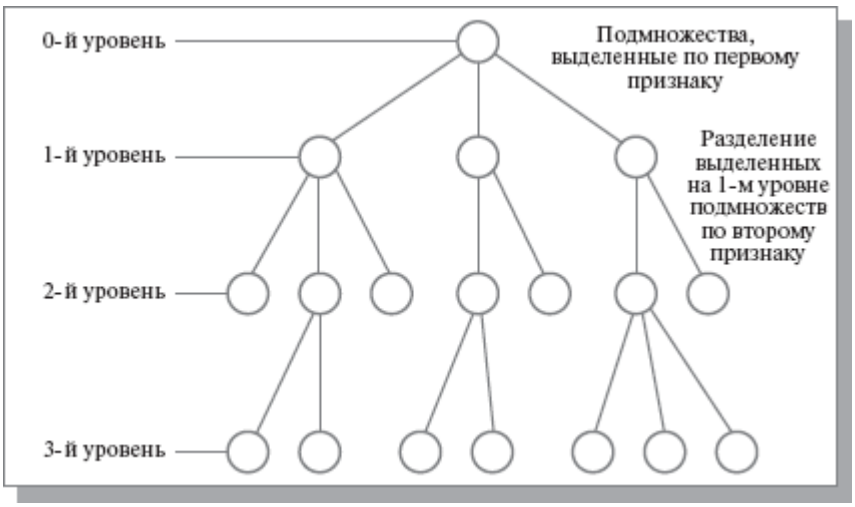


Рис. 2.11. Иерархическая классификационная схема

Характерными особенностями иерархической системы являются:

- возможность использования неограниченного количества признаков *классификации* ;
- соподчиненность признаков *классификации*, что выражается разбиением каждой классификационной группировки, образованной по одному признаку, на множество классификационных группировок по нижестоящему (подчиненному) признаку.

Таким образом, классификационные схемы, построенные на основе иерархического принципа, имеют неограниченную емкость, величина которой зависит от глубины *классификации* (числа ступеней деления) и количества объектов *классификации*, которое можно расположить на каждой ступени. Количество же объектов на каждой ступени *классификации* определяется основанием кода, то есть числом знаков в выбранном алфавите кода. (Например, если алфавит – двузначные десятичные числа, то можно на одном уровне разместить 100 объектов). Выбор необходимой глубины *классификации* и структуры кода зависит от характера объектов *классификации* и характера задач, для решения которых предназначен *классификатор*.

При построении иерархической *системы классификации* сначала выделяется некоторое множество объектов, подлежащее классифицированию, для которого определяются полное множество признаков *классификации* и их соподчиненность друг другу, затем производится разбиение исходного множества объектов на классификационные группировки на каждой ступени *классификации*.

К положительным сторонам данной системы следует отнести логичность, простоту ее построения и удобство логической и арифметической обработки.

Серьезным недостатком иерархического метода *классификации* является жесткость *классификационной схемы*. Она обусловлена заранее установленным выбором признаков *классификации* и порядком их использования по ступеням *классификации*. Это ведет к тому, что при изменении состава объектов *классификации*, их характеристик или характера решаемых при помощи *классификатора* задач требуется коренная переработка *классификационной схемы*. Гибкость этой системы обеспечивается только за счет ввода большой избыточности в ветвях, что приводит к слабой заполненности структуры *классификатора*. Поэтому при разработке *классификаторов* следует учитывать, что иерархический метод классификации более предпочтителен для объектов с относительно стабильными признаками и для решения стабильного комплекса задач.

Примеры применения иерархической *классификации* объектов в корпоративной ИС приведены на рис 2.12 и 2.13. Использование приведенных моделей позволяет выполнить *кодирование информации* о

соответствующих объектах, а также использовать процедуры обобщения при обработке данных (при анализе затрат на заработную плату — по принадлежности работника к определенной службе, при анализе затрат на производство — по группам материалов: по металлу, по покупным комплектующим и пр.).

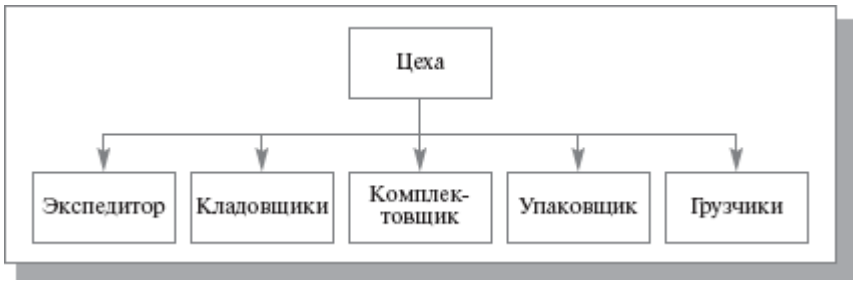


Рис. 2.12. Организационная структура подразделения предприятия-цеха отгрузки

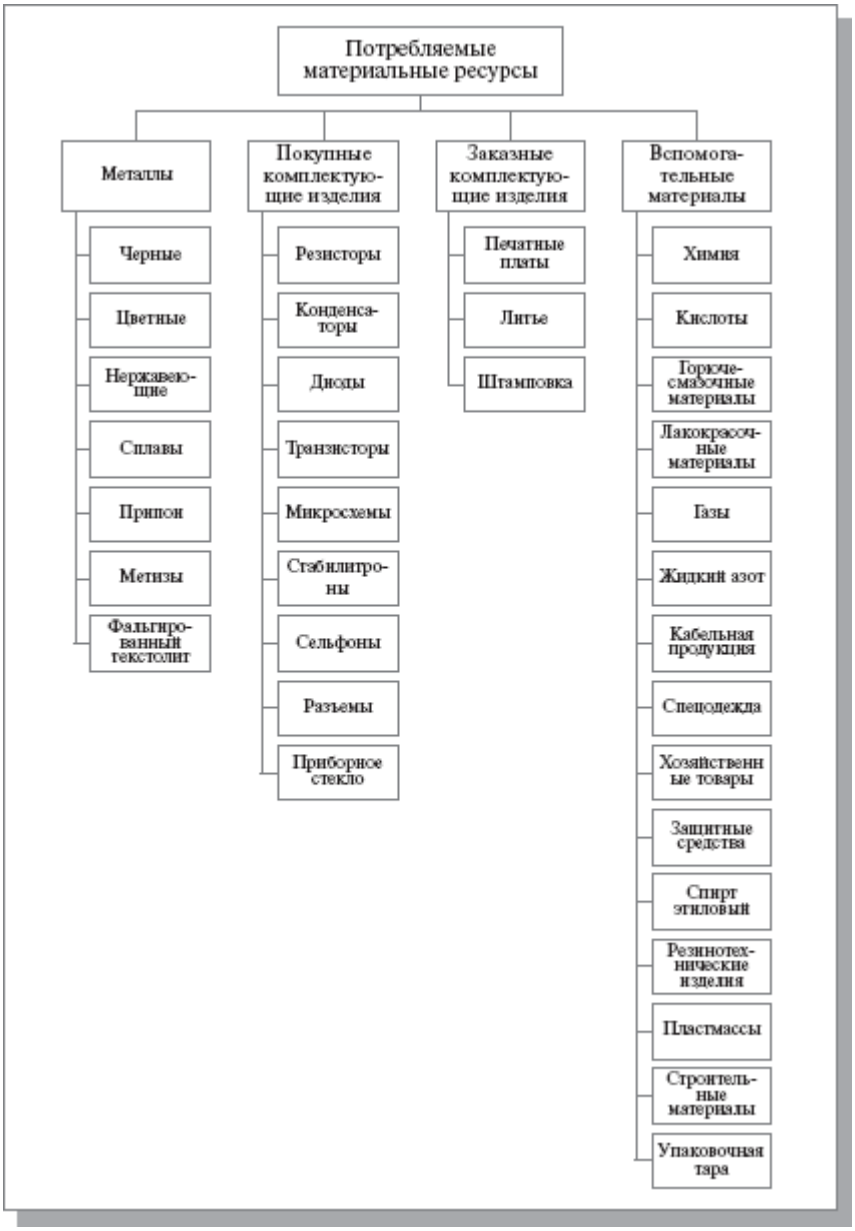


Рис. 2.13. Классификатор материальных ресурсов для обеспечения производства

Недостатки, отмеченные в иерархической системе, отсутствуют в других системах, которые относятся к классу *многоаспектных систем классификации*.

Аспект — точка зрения на объект *классификации*, который характеризуется одним или несколькими признаками.

Многоаспектная система — это *система классификации*, которая использует параллельно несколько независимых признаков (аспектов) в качестве основания *классификации*. Существуют два типа многоаспектных систем:

- фасетная,

- дескрипторная,

- *комбинированная схема.*

Фасет — это аспект *классификации*, который используется для образования независимых классификационных группировок. **Дескриптор** — ключевое слово, определяющее некоторое понятие, которое формирует описание объекта и дает принадлежность этого объекта к классу, группе и т.д.

Под фасетным методом *классификации* понимается "параллельное *разделение множества* объектов на независимые классификационные группировки". При этом методе *классификации* заранее жесткой *классификационной схемы* и конечных группировок не создается. Разрабатывается лишь система таблиц признаков объектов *классификации*, называемых фасетами. При необходимости создания классификационной группировки для решения конкретной задачи осуществляется выборка необходимых признаков из фасетов и их объединение в определенной последовательности. Общий вид фасетной *классификационной схемы* представлен на рис. 2.14.

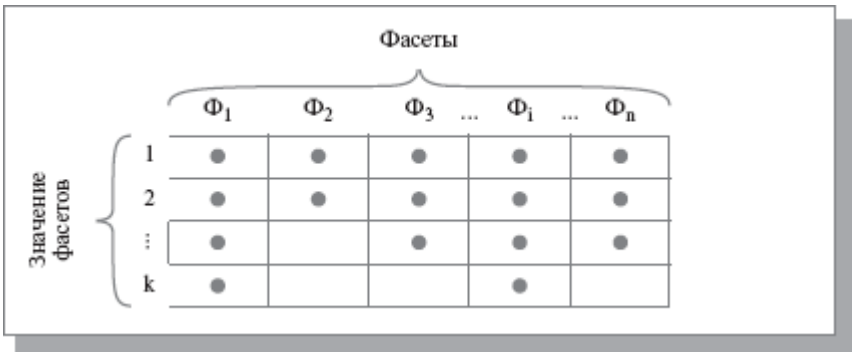


Рис. 2.14. Схема признаков фасетной классификации

Внутри фасета значения признаков могут просто перечисляться по некоторому порядку или образовывать сложную иерархическую структуру, если существует соподчиненность выделенных признаков.

К преимуществам данной системы следует отнести большую емкость системы и высокую степень гибкости, поскольку при необходимости можно вводить дополнительные фасеты и изменять их место в формуле. При изменении характера задач или характеристик объектов *классификации* разрабатываются новые фасеты или дополняются новыми признаками уже существующие фасеты без коренной перестройки структуры всего *классификатора*.

К недостаткам, характерным для данной системы, можно отнести сложность структуры и низкую степень заполненности системы.

В современных *классификационных схемах* часто одновременно используются оба метода *классификации – комбинированная схема*. Это снижает влияние недостатков методов *классификации* и расширяет возможность использования *классификаторов* в информационном обеспечении управления.

В качестве примера использования комбинированных схем *классификации* в корпоративных ИС можно привести следующую модель описания продукции предприятия.

Пример.
Принята классификация выпускаемой продукции по следующему ряду уровней (иерархическая классификация):

- семество продуктов;
- группа продуктов;
- серия продуктов.

Однако эта *система классификации* не обеспечивает идентификацию любого выпускаемого изделия. Для каждой единицы продукта должны указываться следующие атрибуты (Фасеты):

- код серии продукта;
- *конфигурационные параметры;*
- свойства.

Код серии продукта – алфавитно-цифровой код, однозначно идентифицирующий отдельный продукт. *Конфигурационные параметры* – свойства, значения которых могут быть различными в зависимости от *потребностей пользователей*. Свойства – предопределенные характеристики отдельных продуктов, которые не могут меняться для одного и того же продукта.

Признаки фасета "*Конфигурационные параметры*" для одного семейства продуктов приведены в таблице 2.3.

Рассмотренные выше *системы классификации* хорошо приспособлены для организации поиска с целью последующей логической и арифметической обработки информации на ЭВМ, но лишь частично решают проблему содержательного поиска информации при принятии управленческих решений.

Таблица 2.3 - Признаки фасета "*Конфигурационные параметры*" для одного семейства продуктов

Продукты и модификации	**Характеристики**	
	Общие для семейства	**Специальные для отдельных моделей**
Датчики разности давлений	– Искробезопасное исполнение – Взрывозащищенное исполнение – Исполнение по материалам – Климатическое исполнение – Предел допускаемой основной погрешности – Верхний предел измерений – Код выходного сигнала	Предельно допустимое рабочее избыточное давление
Датчики абсолютного давления, избыточного давления, разрежения, давления-разрежения		Измеряемый параметр

	– Состав комплекта монтажных частей	

Для поиска показателей и документов по набору содержательных признаков используется информационный язык дескрипторного типа, который характеризуется совокупностью терминов (дескрипторов) и набором отношений между терминами.

Содержание документов или показателей можно достаточно полно и точно отразить с помощью списка ключевых слов — дескрипторов. **Дескриптор** — это термин естественного языка (слово или словосочетание), используемый при описании документов или показателей, который имеет самостоятельный смысл и неделим без изменения своего значения.

Для того чтобы обеспечить точность и однозначность поиска с помощью дескрипторного языка, необходимо предварительно определить все постоянные отношения между терминами: родовидовые, отношения синонимии, омонимии и полисемии, а также *ассоциативные отношения*.

Все выделенные отношения явно описываются в систематическом словаре понятий — **тезаурусе**, который разрабатывается с целью проведения индексирования документов, показателей и информационных запросов.

Кодирование технико-экономической информации

Для полной формализации информации недостаточно простой *классификации*, поэтому проводят следующую процедуру — кодирование. **Кодирование** — это процесс присвоения условных обозначений объектам и классификационным группам по соответствующей системе кодирования. Кодирование реализует перевод информации, выраженной одной системой знаков, в другую систему, то есть перевод записи на естественном языке в запись с помощью кодов. **Система кодирования** — это совокупность правил обозначения объектов и группировок с использованием кодов. **Код** — это условное обозначение объектов или группировок в виде знака или группы знаков в соответствии с принятой системой. Код базируется на определенном алфавите (некоторое множество знаков). Число знаков этого множества называется основанием кода. Различают следующие типы алфавитов: цифровой, буквенный и смешанный.

Код характеризуется следующими параметрами:

- длиной;
- основанием кодирования;
- структурой кода, под которой понимают распределение знаков по признакам и объектам *классификации* ;
- степенью информативности, рассчитываемой как частное от деления общего количества признаков на длину кода;
- коэффициентом избыточности, который определяется как отношение максимального количества объектов к фактическому количеству объектов.

К методам кодирования предъявляются определенные требования:

- код должен осуществлять идентификацию объекта в пределах заданного множества объектов *классификации* ;
- желательно предусматривать использование в качестве алфавита кода десятичных цифр и букв;
- необходимо обеспечивать по возможности минимальную длину кода и достаточный резерв незанятых позиций для кодирования новых объектов без нарушения структуры *классификатора*.

Методы кодирования могут носить самостоятельный характер – регистрационные методы кодирования, или быть основанными на предварительной *классификации* объектов – классификационные методы кодирования.

Регистрационные методы кодирования бывают двух видов: порядковый и серийно-порядковый. В первом случае кодами служат числа натурального ряда. Каждый из объектов классифицируемого множества кодируется путем присвоения ему текущего порядкового номера. Данный метод кодирования обеспечивает довольно большую долговечность *классификатора* при незначительной избыточности кода. Этот метод обладает наибольшей простотой, использует наиболее короткие коды и лучше обеспечивает однозначность каждого объекта *классификации*. Кроме того, он обеспечивает наиболее простое присвоение кодов новым объектам, появляющимся в процессе ведения *классификатора*. Существенным недостатком порядкового метода кодирования является отсутствие в коде какой-либо конкретной информации о свойствах объекта, а также сложность машинной обработки информации при получении итогов по группе объектов *классификации* с одинаковыми признаками.

В серийно-порядковом методе кодирования кодами служат числа натурального ряда с закреплением отдельных серий этих чисел (интервалов натурального ряда) за объектами *классификации* с одинаковыми признаками. В каждой серии, кроме кодов имеющихся объектов *классификации*, предусматривается определенное количество кодов для резерва.

Классификационные коды используют для отражения классификационных взаимосвязей объектов и группировок и применяются в основном для сложной логической обработки экономической информации. Группу классификационных систем кодирования можно разделить на две *подгруппы* в зависимости от того, какую *систему классификации* используют для упорядочения объектов: системы последовательного кодирования и параллельного кодирования.

Последовательные системы кодирования характеризуются тем, что они базируются на предварительной *классификации* по иерархической системе. Код объекта *классификации* образуется с использованием кодов последовательно расположенных подчиненных группировок, полученных при иерархическом методе кодирования. В этом случае код нижестоящей группировки образуется путем добавления соответствующего количества разрядов к коду вышестоящей группировки.

Параллельные системы кодирования характеризуются тем, что они строятся на основе использования фасетной *системы классификации* и коды группировок по фасетам формируются независимо друг от друга.

В параллельной системе кодирования возможны два варианта записи кодов объекта:

1. Каждый фасет и признак внутри фасета имеют свои коды, которые включаются в состав кода объекта. Такой способ записи удобно применять тогда, когда объекты характеризуются неодинаковым набором признаков. При формировании кода какого-либо объекта берутся только необходимые признаки.
2. Для определения групп объектов выделяется фиксированный набор признаков и устанавливается стабильный порядок их следования, то есть устанавливается фасетная формула. В этом случае не надо каждый раз указывать, значение какого из признаков приведено в определенных разрядах кода объекта.

Параллельный метод кодирования имеет ряд преимуществ. К достоинствам рассматриваемого метода следует отнести гибкость структуры кода, обусловленную независимостью признаков, из кодов которых строится код объекта *классификации*. Метод позволяет использовать при решении конкретных технико-экономических и социальных задач коды только тех признаков объектов, которые необходимы, что дает возможность работать в каждом отдельном случае с кодами небольшой длины. При этом методе

кодирования можно осуществлять группировку объектов по любому сочетанию признаков. Параллельный метод кодирования хорошо приспособлен для машинной обработки информации. По конкретной кодовой комбинации легко узнать, набором каких характеристик обладает рассматриваемый объект. При этом из небольшого числа признаков можно образовать большое число кодовых комбинаций. Набор признаков при необходимости может легко пополняться присоединением кода нового признака. Это свойство параллельного метода кодирования особенно важно при решении технико-экономических задач, состав которых часто меняется.

Наиболее сложными вопросами, которые приходится решать при разработке *классификатора*, являются выбор методов *классификации* и кодирования и выбор системы признаков *классификации*. Основой *классификатора* должны быть наиболее существенные признаки *классификации*, соответствующие характеру решаемых с помощью *классификатора* задач. При этом данные признаки могут быть или соподчиненными, или несоподчиненными. При соподчиненных признаках *классификации* и стабильном комплексе задач, для решения которых предназначен *классификатор*, целесообразно использовать иерархический метод *классификации*, который представляет собой последовательное *разделение множества* объектов на подчиненные классификационные группировки. При несоподчиненных признаках *классификации* и при большой динамичности решаемых задач целесообразно использовать фасетный метод *классификации*.

Важным вопросом является также правильный выбор последовательности использования признаков *классификации* по ступеням *классификации* при иерархическом методе *классификации*. Критерием при этом является статистика запросов к *классификатору*. В соответствии с этим критерием на верхних ступенях *классификации* в *классификаторе* должны использоваться признаки, к которым будут наиболее частые запросы. По этой же причине на верхних ступенях *классификации* выбирают наименьшее основание кода.

Понятие унифицированной системы документации

Основной компонентой внемашинного *информационного обеспечения ИС* является *система документации*, применяемая в процессе управления экономическим объектом. Под документом понимается определенная совокупность сведений, используемая при решении технико-экономических задач, расположенная на материальном носителе в соответствии с установленной формой.

Система документации — это совокупность взаимосвязанных форм документов, регулярно используемых в процессе управления экономическим объектом. Отличительной особенностью системы экономической документации является большое разнообразие видов документов. Существующие *системы документации*, характерные для неавтоматизированных ИС, отличаются большим количеством разных типов форм документов, большим объемом потоков документов и их запутанностью, дублированием информации в документах и работ по их обработке и, как следствие, низкой достоверностью получаемых результатов.

Для того чтобы упростить *систему документации*, используют следующие два подхода:

- проведение унификации и стандартизации документов;
- введение безбумажной технологии, основанной на использовании *электронных документов* и новых информационных технологий их обработки.

Унификация документов выполняется путем введения единых форм документов. Таким образом, вводится единообразие в наименования показателей, единиц измерения и терминов, в результате чего получается унифицированная *система документации*.

Унифицированная *система документации* (УСД) — это рационально организованный комплекс взаимосвязанных документов, который отвечает единым правилам и требованиям и содержит информацию, необходимую для управления некоторым экономическим объектом. По уровням управления, они делятся на межотраслевые *системы документации*, отраслевые и *системы документации* локального уровня, т. е. обязательные для использования в рамках предприятий или организаций.

Любой тип УСД должен удовлетворять следующим **требованиям**:

- документы, входящие в состав УСД, должны разрабатываться с учетом их использования в системе взаимосвязанных ЭИС;
- УСД должна содержать полную информацию, необходимую для оптимального управления тем объектом, для которого разрабатывается эта система;
- УСД должна быть ориентирована на использование средств вычислительной техники для сбора, обработки и передачи информации;
- УСД должна обеспечить информационную совместимость ЭИС различных уровней;
- все документы, входящие в состав разрабатываемой УСД, и все реквизиты-признаки в них должны быть закодированы с использованием международных, общесистемных или локальных *классификаторов*.

Внутримашинное информационное обеспечение

Внутримашинное информационное обеспечение включает макеты (экранные формы) для ввода первичных данных в ЭВМ или вывода результатной информации, и структуры *информационной базы*: входных, выходных файлов, базы данных.

Проектирование экранных форм электронных документов

Под *электронными формами документов* понимается не изображение бумажного документа, а изначально электронная (безбумажная) технология работы.

Электронная форма документа (ЭД) — это страница с пустыми полями, оставленными для заполнения пользователем. Формы могут допускать различный тип *входной информации* и содержать *командные кнопки*, переключатели, выпадающие меню или списки для выбора.

Создание форм *электронных документов* требует использования специального программного обеспечения. На рис. 9.6 приведены основные типовые элементы *электронного документа*, использование которых предусмотрено в большинстве программ автоматизации проектирования *электронных документов*.

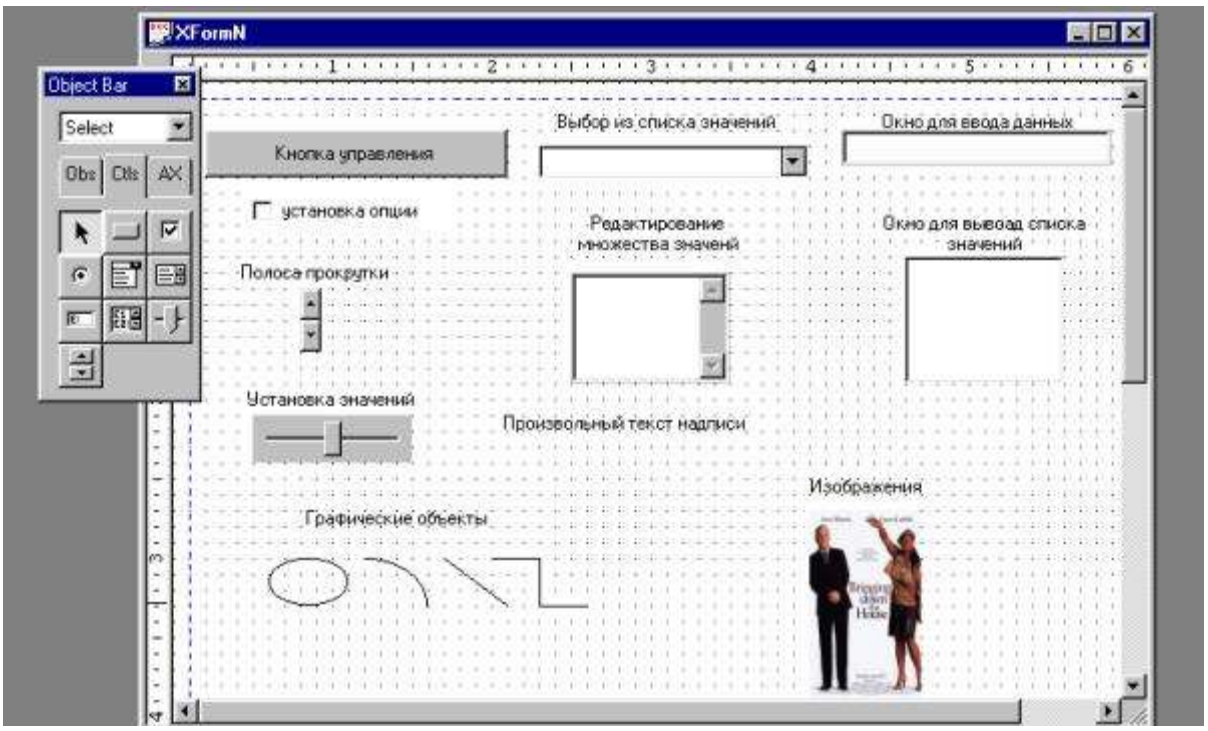


Рис. 9.6. Элементы электронного документа

К недостаткам *электронных документов* можно отнести неполную юридическую проработку процесса их утверждения или подписания.

Технология обработки *электронных документов* требует использования специализированного программного обеспечения — программ управления документооборотом, которые зачастую встраиваются в корпоративные ИС.

Проектирование форм электронных документов, т.е. создание шаблона формы с помощью программного обеспечения проектирования форм, обычно включает в себя выполнение следующих шагов:

- создание структуры ЭД — подготовка внешнего вида с помощью графических средств проектирования;
- определение содержания формы ЭД, т.е. выбор способов, которыми будут заполняться поля. Поля могут быть заполнены вручную или посредством выбора значений из какого-либо списка, меню, базы данных;
- определение перечня макетов экранных форм — по каждой задаче проектировщик анализирует "постановку" задачи, в которой приводятся перечни используемых входных документов с оперативной и постоянной информацией и документов с результатной информацией;
- определение содержания макетов — выполняется на основе анализа состава реквизитов первичных документов с постоянной и оперативной информацией и результатных документов.

Работа заканчивается программированием разработанных макетов *экранных форм* и их апробацией.

Информационная база и способы ее организации

Основной частью внутримашинного информационного обеспечения является *информационная база*. *Информационная база* (ИБ) — это совокупность данных, организованная определенным способом и хранимая в памяти вычислительной системы в виде файлов, с помощью которых удовлетворяются информационные потребности управленческих процессов и решаемых задач.

Все файлы ИБ можно классифицировать по следующим признакам:

- по этапам обработки (входные, базовые, результатные);
- по типу носителя (на промежуточных носителях — гибких магнитных дисках и магнитных лентах и на основных носителях — жестких магнитных дисках, магнитооптических дисках и др.);
- по составу информации (файлы с оперативной информацией и файлы с постоянной информацией);
- по назначению (по типу функциональных подсистем);
- по типу логической организации (файлы с линейной и иерархической структурой записи, реляционные, табличные);
- по способу физической организации (файлы с последовательным, индексным и прямым способом доступа).

Входные файлы создаются с первичных документов для ввода данных или обновления базовых файлов.

Файлы с результатной информацией предназначаются для вывода ее на печать или передачи по каналам связи и не подлежат долговременному хранению.

К числу базовых файлов, хранящихся в *информационной базе*, относят основные, рабочие, промежуточные, служебные и архивные файлы.

Основные файлы должны иметь однородную структуру записей и могут содержать записи с оперативной и условно-постоянной информацией. Оперативные файлы могут создаваться на базе одного или нескольких входных файлов и отражать информацию одного или нескольких первичных документов. Файлы с условно-постоянной информацией могут содержать справочную, расценочную, табличную и другие виды информации, изменяющейся в течение года не более чем на 40%, а следовательно, имеющие коэффициент стабильности не менее 0,6.

Файлы со справочной информацией должны отражать все характеристики элементов материального производства (материалы, сырье, основные фонды, трудовые ресурсы и т.п.). Как правило, справочники содержат информацию *классификаторов* и дополнительные сведения об элементах Материальной сферы, например о ценах. Нормативно-расценочные файлы должны содержать данные о нормах расхода и расценках на выполнение операций и услуг. Табличные файлы содержат сведения об экономических показателях, считающихся постоянными в течение длительного времени (например, процент удержания, отчисления и пр.). Плановые файлы содержат плановые показатели, хранящиеся весь плановый период.

Рабочие файлы создаются для решения конкретных задач на базе основных файлов путем выборки части информации из нескольких основных файлов с целью сокращения времени обработки данных.

Промежуточные файлы отличаются от рабочих файлов тем, что они образуются в результате решения экономических задач, подвергаются хранению с целью дальнейшего использования для решения других задач. Эти файлы, так же как и рабочие файлы, при высокой частоте обращений могут быть также переведены в категорию основных файлов.

Служебные файлы предназначаются для ускорения поиска информации в основных файлах и включают в себя справочники, индексные файлы и каталоги.

Архивные файлы содержат ретроспективные данные из основных файлов, которые используются для решения аналитических, например прогнозных, задач. Архивные данные могут также использоваться для восстановления *информационной базы* при разрушениях.

Организация хранения файлов в *информационной базе* должна отвечать следующим требованиям:

- полнота хранимой информации для выполнения всех функций управления и решения экономических задач;
- целостность хранимой информации, т. е. обеспечение непротиворечивости данных при вводе информации в ИБ;
- своевременность и одновременность обновления данных во всех копиях данных;
- гибкость системы, т.е. адаптируемость ИБ к изменяющимся информационным потребностям;
- реализуемость системы, обеспечивающая требуемую степень сложности структуры ИБ;
- релевантность ИБ, под которой подразумевается способность системы осуществлять поиск и выдавать информацию, точно соответствующую запросам пользователей;
- удобство языкового интерфейса, позволяющее быстро формулировать запрос к ИБ;
- разграничение прав доступа, т.е. определение для каждого пользователя доступных типов записей, полей, файлов и видов операций над ними.

Существуют следующие способы организации ИБ: совокупность локальных файлов, поддерживаемых функциональными пакетами прикладных программ, и интегрированная база данных, основывающаяся на использовании универсальных программных средств загрузки, хранения, поиска и ведения данных, т.е. системы управления базами данных (СУБД).

Локальные файлы вследствие специализации структуры данных под задачи обеспечивают, как правило, более быстрое время обработки данных. Однако недостатки организации локальных файлов, связанные с большим дублированием данных в информационной системе и, как следствие, несогласованностью данных в разных приложениях, а также негибкостью доступа к информации, перекрывают указанные преимущества. Поэтому организация локальных файлов может применяться только в специализированных приложениях, требующих очень высокой скорости реакции при импорте необходимых данных.

Интегрированная ИБ, т.е. база данных (БД) — это совокупность взаимосвязанных, хранящихся вместе данных при такой минимальной избыточности, которая допускает их использование оптимальным образом для множества приложений.

Централизация управления данными с помощью СУБД обеспечивает совместимость этих данных, уменьшение синтаксической и семантической избыточности, соответствие данных реальному состоянию объекта, разделение хранения данных между пользователями и возможность подключения новых пользователей. Но централизация управления и интеграция данных приводят к проблемам другого характера: необходимости усиления контроля вводимых данных, необходимости обеспечения соглашения между пользователями по поводу состава и структуры данных, разграничения доступа и секретности данных.

Основными способами организации БД являются создание централизованных и распределенных БД. Основным критерием выбора способа организации ИБ является достижение минимальных трудовых и стоимостных затрат на проектирование структуры ИБ, программного обеспечения системы ведения файлов, а также на перепроектирование ИБ при возникновении новых задач.

Моделирование информационного обеспечения ИС

ER-модель данных

Модель сущность-связь (ER-модель) (англ. *entity-relationship model*, ERM) — модель данных, позволяющая описывать концептуальные схемы предметной области. Впервые была анонсирована в марте 1976 г. когда ученый Питер Пин - Шань Чен (Peter Pin-Shan Chen) опубликовал работу "The entity-relationship model. Toward a Unified View of Data" в научном журнале "Transactions on Database Systems

(TODS)". Он предложил простой и эффективный способ представления сущностей, атрибутов и связей между ними в виде наглядных диаграмм.

ER- модель принадлежит к разряду концептуальных (понятийных), т.е. отражает логическую природу представления данных. ER – это объектно-ориентированная модель.

Инфологическая модель применяется на втором этапе проектирования базы данных (БД) системы, то есть после словесного описания предметной области. Зачем нужна инфологическая модель и какую пользу она дает проектировщикам? Необходимо напомнить, что процесс проектирования длительный, он требует обсуждений с заказчиком, со специалистами в предметной области. Наконец, при разработке серьезных корпоративных информационных систем проект базы данных является тем фундаментом, на котором строится вся система в целом, и вопрос о возможном кредитовании часто решается экспертами банка на основании именно грамотно сделанного инфологического проекта БД. Следовательно, инфологическая модель должна включать такое формализованное описание предметной области, которое легко будет "читаться" не только специалистами по базам данных. И это описание должно быть настолько емким, чтобы можно было оценить глубину и корректность проработки проекта БД, и конечно, оно не должно быть привязано к конкретной СУБД. Выбор СУБД — это отдельная задача, для корректного ее решения необходимо иметь проект, который не привязан ни к какой конкретной СУБД.

Инфологическое проектирование прежде всего связано с попыткой представления семантики предметной области в модели БД. Реляционная модель данных в силу своей простоты и лаконичности не позволяет отобразить семантику, то есть смысл предметной области. Ранние теоретико-графовые модели в большей степени отображали семантику предметной области. Они в явном виде определяли иерархические связи между объектами предметной области.

Проблема представления семантики давно интересовала разработчиков, и в семидесятых годах было предложено несколько моделей данных, названных семантическими моделями. К ним можно отнести семантическую модель данных, предложенную *Хаммером* (*Hammer*) и *Мак-Леоном* (*McLeon*) в 1981 году, функциональную модель данных *Шипмана* (*Shipman*), также созданную в 1981 году, модель "сущность—связь", предложенную *Ченом* (*Chen*) в 1976 году, и ряд других моделей. У всех моделей были свои положительные и отрицательные стороны, но испытание временем выдержала только последняя. И в настоящий момент именно модель Чена "сущность—связь", или "Entity Relationship", стала фактическим стандартом при инфологическом моделировании баз данных. Общепринятым стало сокращенное название ER-модель, большинство современных CASE-средств содержат инструментальные средства для описания данных в формализме этой модели. Кроме того, разработаны методы автоматического преобразования проекта БД из ER-модели в реляционную, при этом преобразование выполняется в даталогическую модель, соответствующую конкретной СУБД. Все CASE-системы имеют развитые средства документирования процесса разработки БД, автоматические генераторы отчетов позволяют подготовить отчет о текущем состоянии проекта БД с подробным описанием объектов БД и их отношений как в графическом виде, так и в виде готовых стандартных печатных отчетов, что существенно облегчает ведение проекта.

В настоящий момент не существует единой общепринятой системы обозначений для ER-модели и разные CASE-системы используют разные графические нотации, но разобравшись в одной, можно легко понять и другие нотации.

ER-модель используется при высокоуровневом (концептуальном) проектировании баз данных. С её помощью можно выделить ключевые сущности и обозначить связи, которые могут устанавливаться между этими сущностями.

ER-модель представляет собой формальную конструкцию, которая сама по себе не предписывает никаких графических средств её визуализации. В качестве стандартной графической нотации, с помощью которой можно визуализировать ER-модель, была предложена диаграмма сущность-связь (ER-диаграмма) (англ. *entity-relationship diagram*, ERD).

Понятия *ER-модель* и *ER-диаграмма* часто ошибочно не различают.

Для визуализации ER-моделей существуют различные графические нотации:

- Питера Чена
- Bachman notation
- EXPRESS
- IDEF1x
- Martin notation
- (min, max)-Notation
- UML

Инструменты для создания ER-моделей представлены ниже в таблице 2.3.
Таблица 2.3 - Инструменты для создания ER-моделей

Название	Платформа	Лицензия [*]
ARIS		Проприетарная
Avolution (*англ.*)		Проприетарная (EULA)
dbForge Studio for MySQL (*англ.*)	Microsoft Windows	Проприетарная / бесплатное ПО (для неком. использования)
Devgems Data Modeler (*англ.*)	Microsoft Windows: 2000, XP, Vista	Проприетарная
DeZign for Databases (*англ.*)	Microsoft Windows: NT, 2000, XP, Vista, Windows 7 [software 1]	Проприетарная
Dia	Кроссплатформенное ПО	Свободная (GNU GPL)
ER/Studio (*англ.*)	Microsoft Windows	Проприетарная
ERwin	Microsoft Windows: 2000, XP, Server 2003 [software 3]	Проприетарная
Fujaba	Кроссплатформенное ПО (на основе Java)	Свободная (GNU LGPL)
Innovator (*нем.*)	Microsoft Windows 2000, SuSE Linux 10.3, Solaris 8, Red Hat	Проприетарная
MEGA International	Microsoft Windows, Web Platform [software 4]	Проприетарная
Microsoft Visio	Microsoft Windows	Проприетарная
MySQL Workbench	Кроссплатформенное ПО	Свободная (GNU GPL) / проприетарная (EULA)
OmniGraffle (англ.)	Mac OS X v10.5+ [software 5]	Проприетарная
Oracle Designer (*англ.*)	Microsoft Windows	Проприетарная
PowerDesigner (*англ.*)	Microsoft Windows	Проприетарная
Rational Rose (*англ.*)	Кроссплатформенное ПО	Проприетарная
RISE Editor (*англ.*)	Microsoft Windows	Проприетарная / бесплатное ПО
SiSy (*нем.*)	Microsoft Windows	Проприетарная
Sparx Enterprise Architect (*англ.*)	Microsoft Windows, Linux, Mac OS X (с использованием CrossOver)	Проприетарная
SQL Maestro		Проприетарная
SQLyog	Microsoft Windows: 4.10+, NT	Бесплатное ПО / проприетарная (EULA)

Название	Платформа	Лицензия *
StarUML (*англ.*)	Microsoft Windows	Свободная (модифицированный вариант GNU GPL)
System Architect (*англ.*)	Microsoft Windows	Проприетарная
Toad Data Modeler (*англ.*)		Проприетарная
Visual Paradigm (*англ.*)	Кроссплатформенное ПО	Проприетарная / бесплатное ПО (для некоммерческого использования)

В столбце "Лицензия" указан тип программного обеспечения - проприетарное (собственническое) или свободная лицензия, под которой распространяется данное ПО.

Аналогом *диаграммы классов* (UML) может быть *ER-модель*, которая используется при проектировании баз-данных (реляционной модели).

ER-модель позволяет описывать концептуальные схемы предметной области, используется при высокоуровневом (концептуальном) проектировании баз данных.

Любой фрагмент предметной области может быть представлен как множество сущностей, между которыми существует некоторое множество связей. С помощью ER-модели можно выделить ключевые сущности и обозначить связи, которые могут устанавливаться между этими сущностями.

Основные понятия ER-модели.

- *Сущность*, с помощью которой моделируется класс однотипных объектов. Сущность имеет имя, уникальное в пределах моделируемой системы. Так как сущность соответствует некоторому классу однотипных объектов, то предполагается, что в системе существует множество экземпляров данной сущности. Объект, которому соответствует понятие сущности, имеет свой набор *атрибутов* — характеристик, определяющих свойства данного представителя класса. При этом набор атрибутов должен быть таким, чтобы можно было различать конкретные экземпляры сущности. Например, у сущности Сотрудник может быть следующий набор атрибутов: Табельный номер, Фамилия, Имя, Отчество, Дата рождения, Количество детей, Наличие родственников за границей. Набор атрибутов, однозначно идентифицирующий конкретный экземпляр сущности, называют *ключевым*.Для сущности Сотрудник ключевым будет атрибут Табельный номер, поскольку для всех сотрудников данного предприятия табельные номера будут различны. Экземпляром сущности Сотрудник будет описание конкретного сотрудника предприятия. Одно из общепринятых графических обозначений сущности — прямоугольник, в верхней части которого записано имя сущности, а ниже перечисляются атрибуты, причем ключевые атрибуты помечаются, например, подчеркиванием или специальным шрифтом (рис. 2.15):

Рис. 2.15

- *Связи* — бинарные ассоциации, показывающие, каким образом сущности соотносятся или взаимодействуют между собой. Связь может существовать между двумя разными сущностями или между сущностью и ей же самой *(рекурсивная связь)*. Она показывает, как связаны экземпляры сущностей между собой. Если связь устанавливается между двумя сущностями, то она определяет взаимосвязь между экземплярами одной и другой сущности. Например, если у нас есть связь между сущностью

"Студент" и сущностью "Преподаватель" и эта связь — руководство дипломными проектами, то каждый студент имеет только одного руководителя, но один и тот же преподаватель может руководить множеством студентов-дипломников. Поэтому это будет связь "один-ко-многим" (1:M), один со стороны "Преподаватель" и многие со стороны "Студент" (см. рис. 2.16).

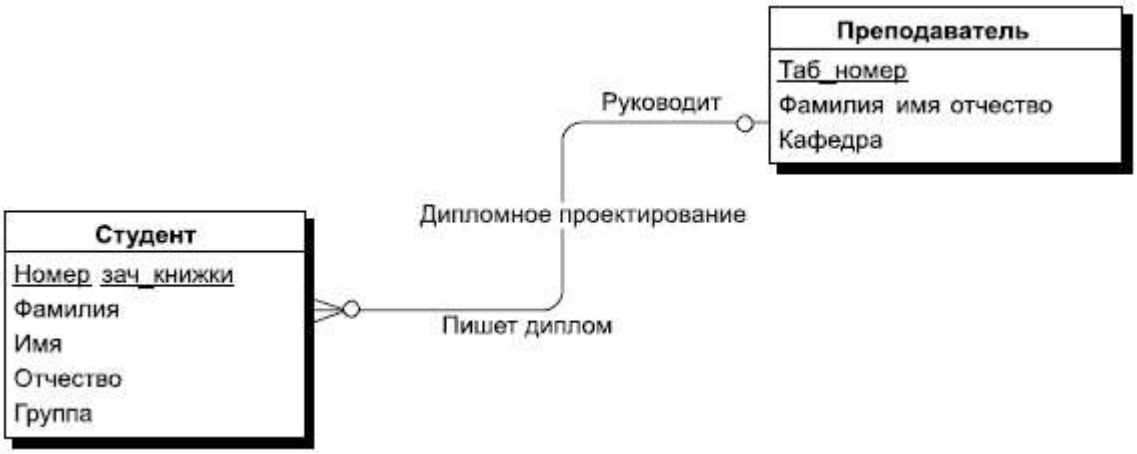


Рис. 2.16

В разных нотациях мощность связи изображается по-разному. В нашем примере мы используем нотацию CASE системы POWER DESIGNER, здесь множественность изображается путем разделения линии связи на 3. Связь имеет общее имя "Дипломное проектирование" и имеет имена ролей со стороны обеих сущностей. Со стороны студента эта роль называется "Пишет диплом под руководством", со стороны преподавателя эта связь называется "Руководит". Графическая интерпретация связи позволяет сразу прочитать смысл взаимосвязи между сущностями, она наглядна и легко интерпретируема.

Связи делятся на три типа по множественности: *один-к-одному* (1:1), *один-ко-многим* (1:M), *многие-ко-многим* (M:M).

Связь один-к-одному означает, что экземпляр одной сущности связан только с одним экземпляром другой сущности. Связь 1: M означает, что один экземпляр сущности, расположенный слева по связи, может быть связан с несколькими экземплярами сущности, расположенными справа по связи. Связь "один-к-одному" (1:1) означает, что один экземпляр одной сущности связан только с одним экземпляром другой сущности, а связь "многие-ко-многим" (M:M) означает, что один экземпляр первой сущности может быть связан с несколькими экземплярами второй сущности, и наоборот, один экземпляр второй сущности может быть связан с несколькими экземплярами первой сущности. Например, если мы рассмотрим связь типа "Изучает" между сущностями "Студент" и "Дисциплина", то это связь типа "многие-ко-многим" (M:M), потому что каждый студент может изучать несколько дисциплин, но и каждая дисциплина изучается множеством студентов.

Между двумя сущностями может быть задано сколько угодно связей с разными смысловыми нагрузками. Например, между двумя сущностями "Студент" и "Преподаватель" можно установить две смысловые связи, одна — рассмотренная уже ранее "Дипломное проектирование", а вторая может быть условно названа "Лекции", и она определяет, лекции каких преподавателей слушает данный студент и каким студентам данный преподаватель читает лекции. Ясно, что это связь типа *многие-ко-многим*.

Пример этих связей приведен на рис. 2.17.

Рис. 2.17. Пример моделирования связи "многие-ко-многим"

- Связь любого из этих типов может быть *обязательной*,если в данной связи должен участвовать каждый экземпляр сущности, *необязательной* — если не каждый экземпляр сущности должен участвовать в данной связи. При этом связь может быть *обязательной с одной стороны* и *необязательной с другой стороны*.Обязательность связи тоже по-разному обозначается в разных нотациях. Мы снова используем нотацию POWER DESIGNER. Здесь необязательность связи обозначается пустым кружочком на конце связи, а обязательность перпендикулярной линией, перечеркивающей связь. И эта нотация имеет простую интерпретацию. Кружочек означает, что ни один экземпляр не может участвовать в этой связи. А перпендикуляр интерпретируется как то, что по крайней мере один экземпляр сущности участвует в этой связи. Рассмотрим для этого ранее приведенный пример связи "Дипломное проектирование". На нашем рисунке эта связь интерпретируется как необязательная с двух сторон. Но ведь на самом деле каждый студент, который пишет диплом, должен иметь своего руководителя дипломного проектирования, но, с другой стороны, не каждый преподаватель должен вести дипломное проектирование. Поэтому в данной смысловой постановке изображение этой связи изменится и будет выглядеть таким, как представлено на рис. 2.18.

Кроме того, в ER-модели допускается принцип категоризации сущностей. Это значит, что, как и в объектно-ориентированных языках программирования, вводится понятие подтипа сущности, то есть сущность может быть представлена в виде двух или более своих подтипов — *сущностей*, каждая из которых может иметь общие атрибуты и отношения и/или атрибуты и отношения, которые определяются однажды на верхнем уровне и наследуются на нижнем уровне. Все подтипы одной сущности рассматриваются как взаимоисключающие, и при разделении сущности на подтипы она должна быть представлена в виде полного набора взаимоисключающих подтипов. Если на уровне анализа не удается выявить полный перечень подтипов, то вводится специальный подтип, называемый условно ПРОЧИЕ, который в дальнейшем может быть уточнен. В реальных системах бывает достаточно ввести подтипизацию на двух-трех уровнях.

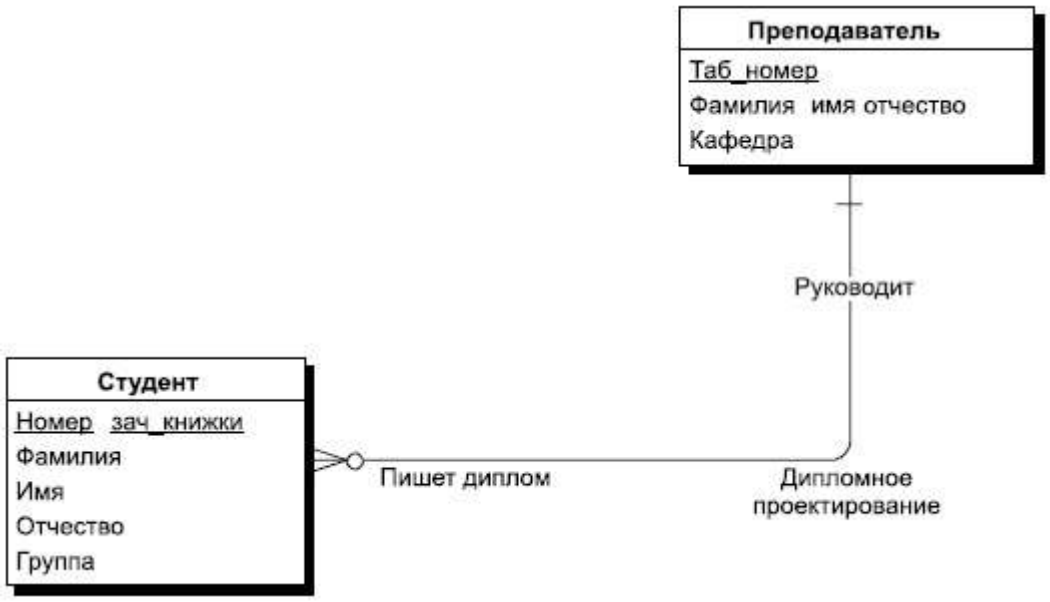


Рис. 2.18. Пример обязательной и необязательной связи между сущностями

Сущность, на основе которой строятся подтипы, называется *супертипом*.Любой экземпляр супертипа должен относиться к конкретному подтипу. Для графического изображения принципа категоризации или типизации сущности вводится специальный графический элемент, называемый *узел-дискриминатор*,в нотации POWER DESIGNER он изображается в виде полукруга, выпуклой стороной обращенного к суперсущности. Эта сторона соединяется направленной стрелкой с суперсущностью, а к диаметру этого круга стрелками подсоединяются подтипы данной сущности (см. рис. 2.19).

Рис. 2.19. Диаграмма подтипов сущности ТЕСТ

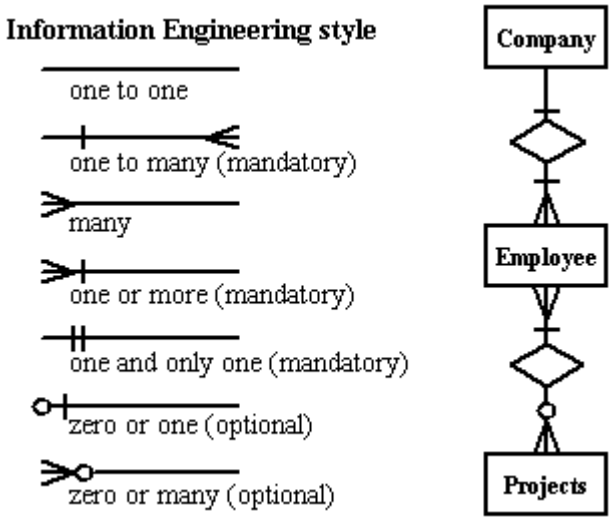
Эту диаграмму можно расшифровать следующим образом. Каждый тест в некоторой системе тестирования является либо тестом проверки знаний языка SQL, либо некоторой аналитической задачей, которая выполняется с использованием заранее написанных Java-апплетов, либо тестом по некоторой области знаний, состоящим из набора вопросов и набора ответов, предлагаемых к каждому вопросу.

- *Домен* (domain) - множество значений (область определения) атрибута (столбец таблицы в реляционной БД).

В результате построения модели предметной области в виде набора сущностей и связей получаем связный граф. В полученном графе необходимо избегать циклических связей — они выявляют некорректность модели.

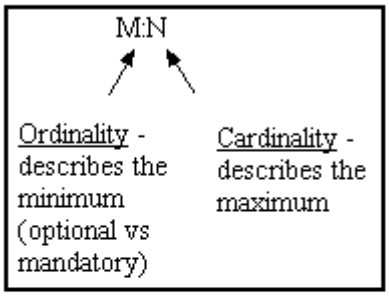
Существует разные способы записи ER-диаграмм, примеры которых приводятся ниже:

Стиль *Information Engineering*



Стиль *Chen*

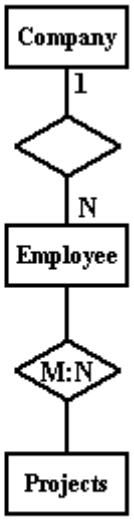
Chen style



1:N (n=0,1,2,3...)
one to zero or more

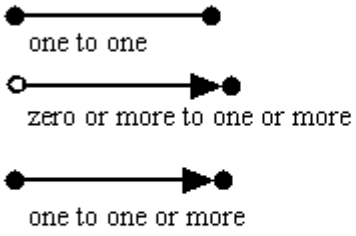
M:N (m and n=0,1,2,3...)
zero or more to zero or more
(many to many)

1:1
one to one



Стиль *Bachman*

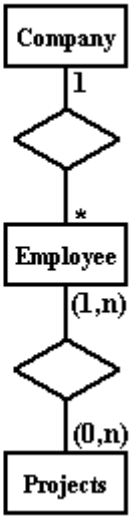
Bachman style



Стиль *Martin*

Martin style

1 - one, and only one (mandatory)
* - many (zero or more - optional)
1...* - one or more (mandatory)
0...1 - zero or one (optional)
(0,1) - zero or one (optional)
(1,n) -one or more (mandatory)
(0,n) - zero or more (optional)
(1,1) - one and only one (mandatory)



Для разработки ER-диаграмм можно воспользоваться программными продуктами:
MySQL Workbench; PostgreSQL Database Modeler; DbSchema; Valentina Studio.

На основе ER-модели разрабатывается реляционная БД, которая представляет собой множество отношений, а ее схема содержит следующие компоненты:

$$S = < A, R, Dom, Rel, V(s) > ,$$

где A – множество имен атрибутов,
R – множество имен отношений (сущностей),
Dom – вхождение атрибутов в домены,
Rel – вхождение атрибутов в отношения (описание структуры отношений),
V(s) – множество ограничений (в том числе функциональных зависимостей), обеспечивающих целостность базы данных.

Отношения (сущности) представляются в виде таблиц. При этом столбцы таблицы соответствуют атрибутам. Каждый атрибут определен на некотором домене. Доменом называют множество атомарных значений объекта автоматизации.
Рассмотрим формирование ER - модели в стиле Чена на конкретном примере.

Пример.

Рассматривается процесс учета расчетов клиентов за интернет –услуги провайдера. Анализ предметной области позволяет выделить, например, следующие сущности:

1. пользователь;
2. тип пользователя;
3. VIP статус пользователя;
4. регион;
5. реквизиты физического лица;
6. реквизиты юридического лица;
7. тип организации;
8. тарифные планы коммутируемого доступа;
9. группы услуг;
10. учетные имена пользователей для доступа в сеть Интернет;
11. ограничения бюджета пользователя по телефонным номерам;
12. ограничения бюджета по времени доступа и дням недели;
13. почтовые ящики пользователя;
14. структура входных файлов;
15. коды услуг входных файлов;
16. импортированные входные данные.

Ниже рассмотрены основные атрибуты каждой сущности.

1. Пользователь

- тип пользователя (юридическое, физическое лицо)
- VIP статус пользователя;
- регион откуда пользователь будет работать;
- лицевой счет пользователя;
- номер договора;
- дата заключения договора;
- имя пользователя (наименование организации или Ф.И.О.);
- почтовый адрес клиента;
- юридический адрес (адрес по прописке);
- ответственный представитель;
- контактный телефон пользователя;
- адрес электронной почты, для связи с пользователем и почтовых рассылок;

- дата расторжения договора.

2. Тип пользователя
- наименование типа пользователя (физическое, юридическое лицо).

3. VIP статус пользователя
- наименование VIP статуса пользователя.

4. Регион
- наименование региона;
- домен региона;
- администратор региона (Ф.И.О.);
- контактный телефон администратора;
- пропускная способность канала.

5. Реквизиты физического лица
- серия паспорта пользователя;
- кем выдан паспорт;
- когда выдан паспорт.

6. Реквизиты юридического лица
- индивидуальный номер налогоплательщика ИНН;
- расчетный счет;
- банк клиента;
- код БИК;
- корреспондентский счет;
- код ОКПО;
- код ОКОНХ;

7. Тип организации
- наименование типа организации.

8. Тарифные планы коммутируемого доступа с учетом времени соединения
- признак группы услуг коммутируемого доступа;
- тип пользователя (юридическое, физическое лицо);
- позиция прейскуранта;
- цена;
- признак будних или выходных дней;
- признак зонового продления;
- код услуги входного файла.

9. Группа услуг
- наименование группы услуг.

10. Учетные имена пользователей для доступа в сеть Интернет
- учетное имя для доступа в сеть Интернет;
- пароль для учетного имени;
- регион откуда будет производиться доступ;
- какому пользователю принадлежит учетное имя;
- дата выделения учетного имени;
- дата отказа от услуги;
- группа услуг тарифного плана.

11. Ограничения бюджета пользователя по телефонным номерам
- учетное имя для доступа в сеть Интернет;
- телефонный номер.

12. Ограничения бюджета по времени доступа и дням недели
- учетное имя для доступа в сеть Интернет;
- разрешить доступ начиная с времени:
- разрешить доступ заканчивая с времени:
- день недели, в который доступ разрешен.

13. Почтовые ящики пользователя

- учетная запись почты;
- пароль для учетной записи;
- адрес электронной почты;
- дата выделения почтового ящика;
- дата отказа от почтового ящика.

14. Группа входных файлов
- имя файла, в котором содержится статистика по видам услуг.

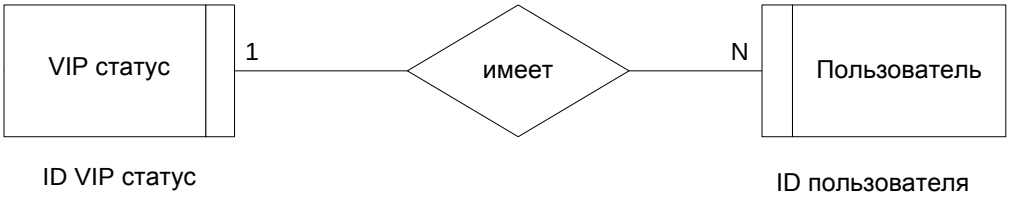
15. Коды услуг входных файлов
- имя файла, в котором содержится статистика по видам услуг;
- номер поля входного файла;
- код услуги входного файла.

16. Импортированные данные входных файлов
- учетное имя пользователя;
- код услуги входного файла;
- количество услуги;
- дата услуги.

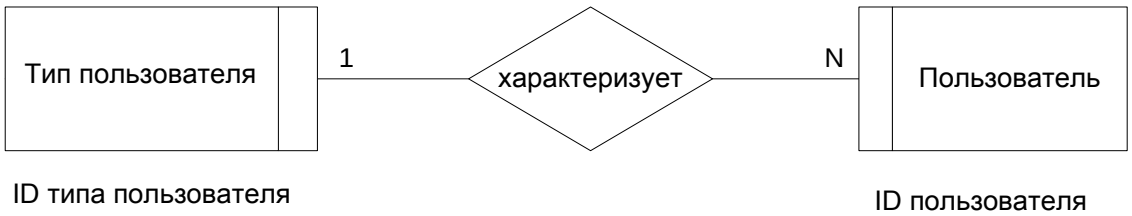
Рассмотрим все связи между всеми сущностями предметной области.

1) Пользователь заключает договор в регионе. Каждый пользователь заключает договор в определенном регионе, но в любом конкретном регионе может работать несколько пользователей. Степень связи между сущностями 1:N, проиллюстрируем связь с помощью диаграммы ER-типов.

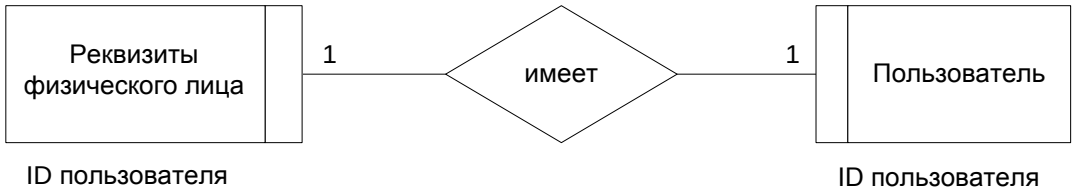
2) Пользователь имеет VIP статус. Каждый пользователь имеет только один VIP статус, в то время как один и тот же VIP статус могут иметь и другие пользователи. Степень связи между сущностями 1:N, проиллюстрируем связь с помощью диаграммы ER-типов.



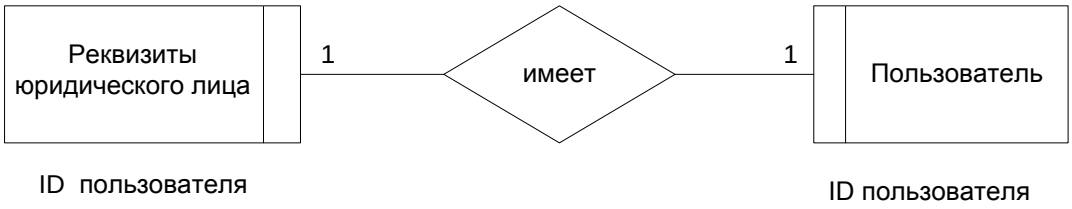
3) Пользователя характеризует тип пользователя. Каждый пользователь имеет только один тип – физическое или юридическое лицо, в то время как один и тот же тип может быть у нескольких пользователей. Степень связи между сущностями 1:N, проиллюстрируем связь с помощью диаграммы ER-типов.



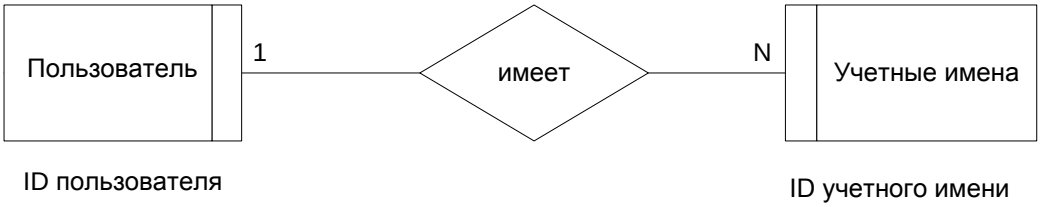
Пользователь физическое лицо имеет реквизиты. Каждый пользователь физическое лицо имеет реквизиты, указывающие его паспортные данные у каждого пользователя они свои. Степень связи между сущностями 1:1, проиллюстрируем связь с помощью диаграммы ER-типов.



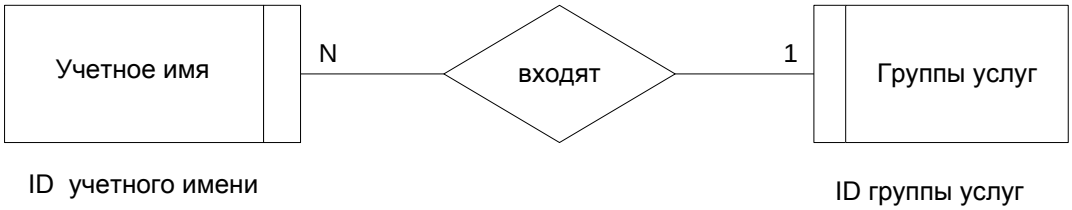
4) Пользователь юридическое лицо имеет реквизиты. Каждый пользователь юридическое лицо имеет данные, указывающие его банковские реквизиты у каждого пользователя юридического лица они свои. Степень связи между сущностями 1:1, проиллюстрируем связь с помощью диаграммы ER-типов.



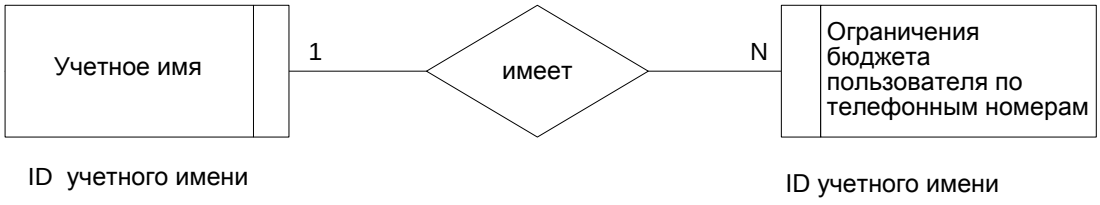
5) Пользователь имеет учетные имена. Каждый пользователь может иметь несколько входных имен для регистрации в сети Интернет. Степень связи между сущностями 1:N, проиллюстрируем связь с помощью диаграммы ER-типов.



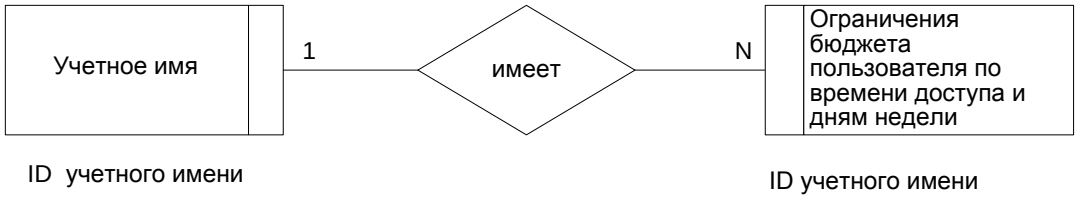
6) Учетные имена входят в группу услуг. Каждое учетное имя входит в какую-то группу услуг: коммутируемый доступ с учетом времени соединения, без учета времени соединения, доступ по линии ISDN, по аналоговой линии. Степень связи между сущностями 1:N, проиллюстрируем связь с помощью диаграммы ER-типов.



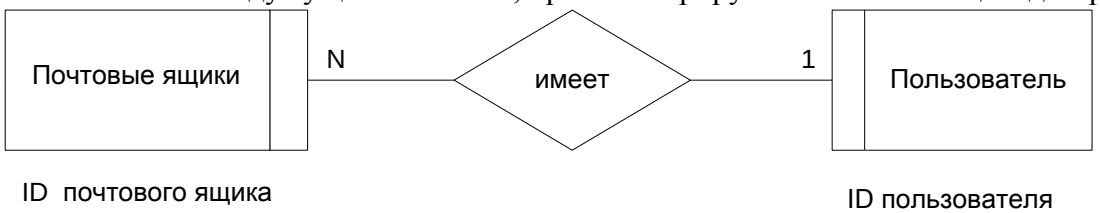
7) Учетное имя имеет ограничения бюджета по телефонным номерам. Учетное имя может иметь несколько ограничений по доступу в сеть по телефонным номерам, степень связи между сущностями 1:N, проиллюстрируем связь с помощью диаграммы ER-типов.



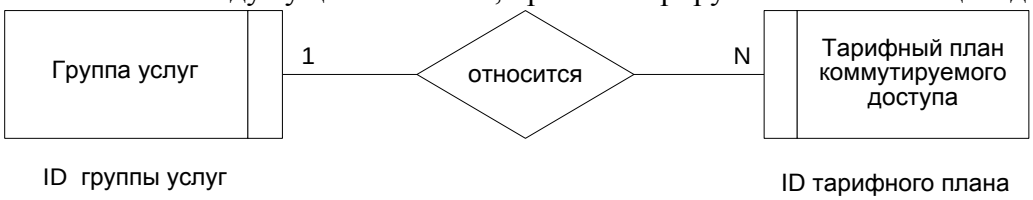
8) Учетное имя имеет ограничения бюджета по времени доступа и дням недели. Учетное имя может иметь несколько ограничений по доступу в сеть по времени доступа и дням недели, степень связи между сущностями 1:N, проиллюстрируем связь с помощью диаграммы ER-типов.



9) Пользователь имеет почтовые ящики. Любой пользователь может иметь несколько почтовых ящиков, степень связи между сущностями 1:N, проиллюстрируем связь с помощью диаграммы ER-типов.

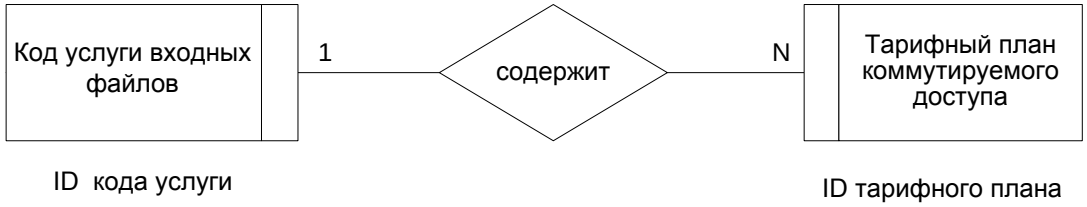


Тарифные планы коммутируемого доступа с учетом времени соединения относятся к группе услуг. Каждый тарифный план для коммутируемого доступа с учетом времени соединения относится к какой-то группе услуг. Степень связи между сущностями 1:N, проиллюстрируем связь с помощью диаграммы ER-типов.

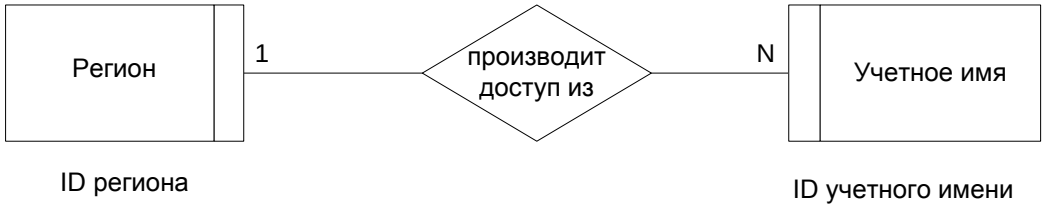


10) Тип пользователя указывает принадлежность тарифа . Каждый тарифный план для коммутируемого доступа с учетом времени соединения относится к какому-то типу пользователя (физическое или юридическое лицо). Степень связи между сущностями 1:N, проиллюстрируем связь с помощью диаграммы ER-типов.

11) Тарифные планы коммутируемого доступа с учетом времени соединения содержат коды услуг входных файлов. Каждый тарифный план для коммутируемого доступа с учетом времени соединения имеет код услуги входных файлов. Степень связи между сущностями 1:N, проиллюстрируем связь с помощью диаграммы ER-типов.



12) Учетное имя производит доступ из региона. Каждое учетное имя может производить доступ с одного региона без учета зонового продления. Степень связи между сущностями 1:N, проиллюстрируем связь с помощью диаграммы ER-типов.



Полная ER - диаграмма показана ниже на рис. 2.20.

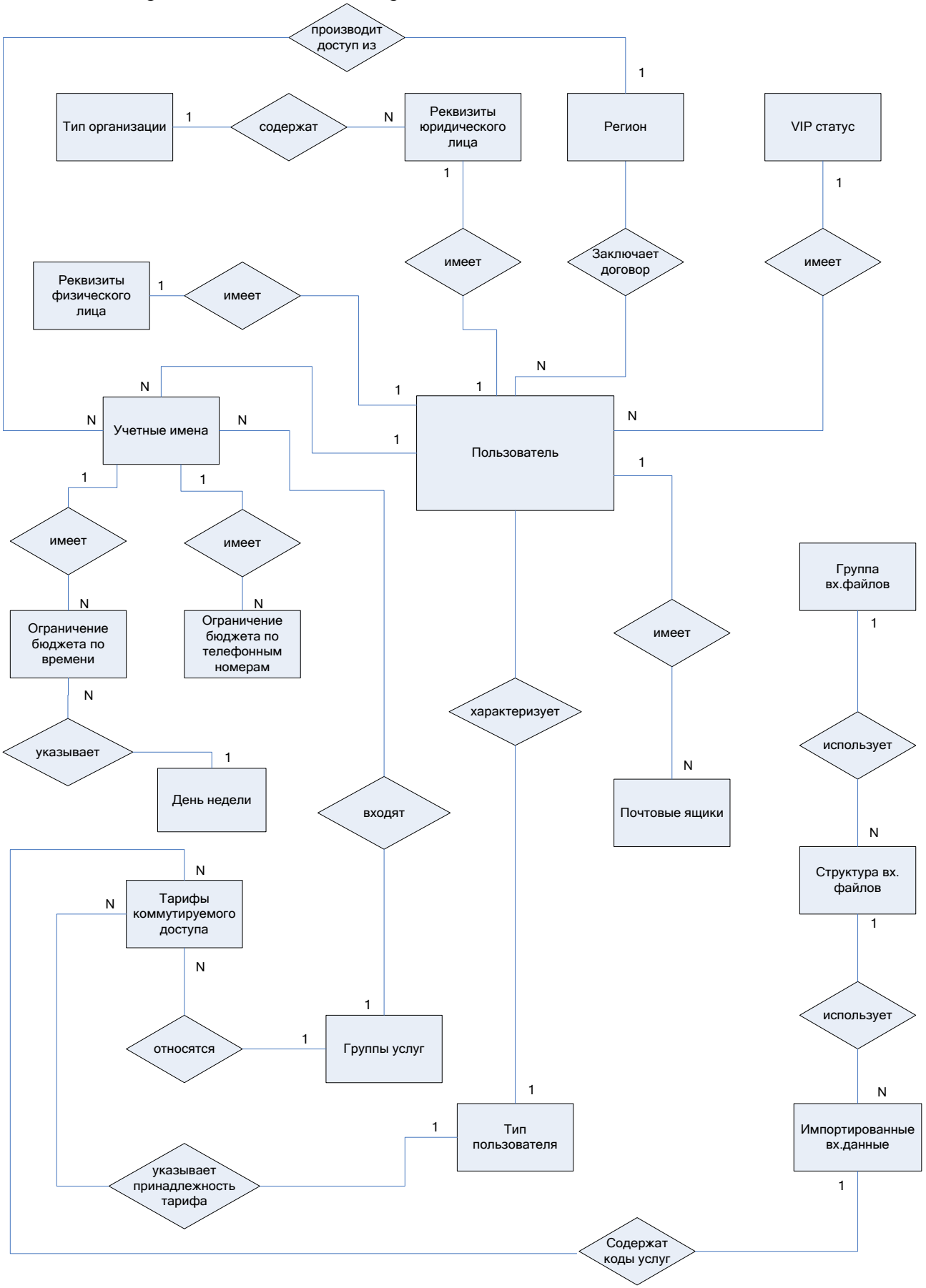


Рис. 2.20. Пример ER - диаграммы

Проектирование ИС с применением UML

Оценка эффективности проекта

Объектно-ориентированные методологии описания предметной области

Принципиальное отличие между функциональным и объектным подходом заключается в способе декомпозиции системы. Объектно-ориентированный подход использует объектную декомпозицию, при этом статическая структура описывается в терминах объектов и связей между ними, а поведение системы описывается в терминах обмена сообщениями между объектами.

Основными понятиями объектно-ориентированного подхода являются объект и класс.

Объект — предмет или явление, имеющее четко определенное поведение и обладающие состоянием, поведением и индивидуальностью. Структура и поведение схожих объектов определяют общий для них класс.

Класс – это множество объектов, связанных общностью структуры и поведения.

Следующую группу важных понятий объектного подхода составляют наследование и полиморфизм.

Понятие *полиморфизм* может быть интерпретировано как способность класса принадлежать более чем одному типу.

Наследование означает построение новых классов на основе существующих с возможностью добавления или переопределения данных и методов.

Важным качеством объектного подхода является согласованность моделей деятельности организации и моделей проектируемой информационной системы от стадии формирования требований до стадии реализации. По объектным моделям может быть прослежено отображение реальных сущностей моделируемой предметной области в объекты и классы информационной системы.

Большинство существующих методов объектно-ориентированного подхода включают язык моделирования и описание процесса моделирования. Процесс – это описание последовательных шагов, которые необходимо выполнить при разработке проекта. В качестве языка моделирования объектного подхода используется унифицированный язык моделирования **UML** (Unified Modeling Language — унифицированный язык моделирования) — язык графического описания для объектного моделирования в области разработки программного обеспечения, моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

UML содержит стандартный набор диаграмм для моделирования.

Диаграмма (Diagram) — это графическое представление множества элементов. Чаще всего она изображается в виде связного графа с вершинами (сущностями) и ребрами (отношениями) и представляет собой некоторую проекцию системы.

Объектно-ориентированный подход обладает следующими преимуществами:

- объектная декомпозиция дает возможность создавать модели меньшего размера путем использования общих механизмов, обеспечивающих необходимую экономию выразительных средств;
- использование объектного подхода существенно повышает уровень унификации разработки и пригодность для повторного использования, что ведет к созданию среды разработки и переходу к сборочному созданию моделей;
- объектная декомпозиция позволяет избежать создания сложных моделей, так как она предполагает эволюционный путь развития модели на базе относительно небольших подсистем;
- объектная модель естественна, поскольку ориентирована на человеческое восприятие мира.

К недостаткам объектно-ориентированного подхода относятся высокие начальные затраты. Этот подход не дает немедленной отдачи. Эффект от его применения сказывается после разработки двух–трех проектов и накопления повторно используемых компонентов. Диаграммы, отражающие специфику объектного подхода, менее наглядны.

Синтетический подход к моделированию

С точки зрения моделирования ИС функциональный и объектный подходы обладают своими преимуществами и недостатками. Объектный подход позволяет построить более устойчивую к изменениям систему, лучше соответствует существующим *структурам организации. Функциональное моделирование* хорошо показывает себя в тех случаях, когда организационная структура находится в процессе изменения или вообще слабо оформлена. Функциональный подход интуитивно лучше понимается исполнителями при получении от них информации об их текущей работе. Каждый из рассмотренных подходов позволяет решить задачу построения формального описания рабочих процедур проектируемой системы, позволяют построить модель "***как есть***" и "***как должно быть***". С другой стороны, каждый из подходов и соответствующие им методики обладают определенными преимуществами и недостатками.

Функциональные методики в целом лучше дают представление о существующих функциях в организации, о методах их реализации, причем, чем выше степень детализации исследуемого процесса, тем лучше они позволяют описать систему. Под лучшим описанием в данном случае понимается наименьшая ошибка при попытке по полученной модели предсказать поведение реальной системы. На уровне отдельных рабочих процедур их описание практически однозначно совпадает с фактической реализацией в потоке работ. Однако на уровне общего описания системы

функциональные методики не обеспечивают представление и выбор интерфейсов системы, ее механизмов и т.д., то есть не позволяют четко определить границы системы.

Хорошо описать систему на этом уровне позволяет объектный подход, основанный на понятии сценария использования. Ключевым является понятие о сценарии использования как о сеансе взаимодействия действующего лица с системой, в результате которого действующее лицо получает нечто, имеющее для него ценность. Использование критерия ценности для пользователя дает возможность отбросить не имеющие значения детали потоков работ и функций и сосредоточиться на тех функциях системы, которые оправдывают ее существование. Однако и в этом случае задача определения границ системы, выделения внешних пользователей является сложной.

Технология потоков данных, исторически возникшая первой, легко решает проблему границ системы, поскольку позволяет за счет анализа информационных потоков выделить внешние сущности и определить основной внутренний процесс. Однако отсутствие выделенных управляющих процессов, потоков и событий не позволяет использовать эту методику в качестве универсальной.

Наилучшим способом преодоления недостатков рассмотренных методик является применение ***синтетической методики***, объединяющей различные этапы отдельных методик. При этом из каждой методики берется та часть, которая наиболее полно и формально изложена и обеспечивает возможность обмена результатами с другими методиками на различных этапах моделирования.

Рассмотрим применение синтетической методики на примере разработки административного регламента ИС.

При построении административных регламентов выделяются следующие стадии:

1. Определение границ системы. На этой стадии при помощи *анализа потоков данных выделяют внешние сущности* и собственно границы моделируемой системы;
2. Выделение сценариев использования системы. На этой стадии при помощи *критерия полезности* строят для каждой внешней сущности *набор сценариев использования системы;*
3. Добавление системных сценариев использования ИС. На этой стадии определяют сценарии*, необходимые для реализации целей системы*, отличных от целей пользователей;
4. Построение диаграммы активностей по сценариям использования ИС. На этой стадии строят *набор действий системы (функций)*, приводящих к реализации сценариев ее использования;
5. Функциональная *декомпозиция диаграмм активностей* (функций) как контекстных диаграмм методики IDEF0;
6. Формальное описание отдельных функциональных активностей в виде административного регламента (с применением различных нотаций).

Сравнение существующих подходов к моделированию

В функциональных моделях (DFD-диаграммах потоков данных, IDEF-диаграммах) главными структурными компонентами являются функции (операции, действия, работы), которые на диаграммах связываются между собой информационными потоками.

Достоинством функциональных моделей является реализация структурного подхода к проектированию ИС по принципу "сверху-вниз", когда каждый функциональный блок может быть декомпозирован на множество подфункций и т.д., выполняя, таким образом, модульное проектирование ИС. Для функциональных моделей характерны процедурная строгость декомпозиции функций и наглядность представления. При функциональном подходе модели данных в виде ER-диаграмм "объект — свойство — связь" разрабатываются отдельно. Для проверки корректности моделирования предметной области между функциональными моделями и моделями данных устанавливаются взаимно однозначные связи. Главный недостаток функциональных моделей заключается в том, что процессы и данные существуют отдельно друг от друга — помимо функциональной декомпозиции существует структура данных, находящаяся на втором плане. Кроме того, не ясны условия выполнения процессов обработки информации, которые динамически могут изменяться.

Перечисленные недостатки функциональных моделей снимаются в объектно-ориентированных моделях, в которых главным структурообразующим компонентом выступает класс объектов с набором функций, которые могут обращаться к атрибутам этого класса.

Для классов объектов характерна иерархия обобщения, позволяющая осуществлять ***наследование*** не только атрибутов (свойств) объектов от вышестоящего класса объектов к нижестоящему классу, но и функций (методов). В случае наследования функций можно абстрагироваться от конкретной реализации процедур (***абстрактные типы данных***), которые отличаются для определенных подклассов ситуаций. Это дает возможность обращаться к подобным программным модулям по общим именам (***полиморфизм***) и осуществлять повторное использование программного кода при модификации программного обеспечения. Таким образом, адаптивность объектно-ориентированных систем к изменению предметной области по сравнению с функциональным подходом значительно выше. При объектно-ориентированном подходе изменяется и принцип проектирования ИС. Сначала выделяются классы объектов, а далее в зависимости от возможных состояний объектов (жизненного цикла объектов) определяются методы обработки (функциональные процедуры), что обеспечивает наилучшую реализацию динамического поведения информационной системы. Для объектно-ориентированного подхода разработаны графические методы моделирования предметной области, обобщенные в языке унифицированного моделирования UML. Однако по наглядности представления модели пользователю-заказчику объектно-ориентированные модели явно уступают функциональным моделям.

При выборе методики моделирования предметной области обычно в качестве критерия выступает степень ее динамичности. Для более регламентированных задач больше подходят функциональные модели, для более адаптивных бизнес-процессов (управления рабочими потоками, реализации динамических запросов к информационным хранилищам) — объектно-ориентированные модели. Однако в рамках одной и той же ИС для различных классов задач могут требоваться различные виды моделей, описывающих одну и ту же проблемную область. В таком случае целесообразно использовать комбинированные модели предметной области.

Унифицированный язык моделирования UML

Существует множество технологий и инструментальных средств, с помощью которых можно реализовать в некотором смысле оптимальный проект ИС, начиная с этапа анализа и заканчивая созданием программного кода системы. В большинстве случаев эти технологии предъявляют весьма жесткие требования к процессу разработки и используемым ресурсам, а попытки трансформировать их под конкретные проекты оказываются безуспешными. Эти технологии в основном представлены двумя группами средств CASE-I (первая технология) и CASE-II (вторая технология).

CASE-I включают:

- средства анализа требований,

- средства проектирования спецификаций и структуры,

- средства разработки и редактирования интерфейсов.

CASE-II включают:

- средства генерации программных кодов;

- средства реализации интегрированного окружения поддержки полного жизненного цикла разработки ПО.

Средства CASE-I являются первой технологией, адресованной непосредственно системным аналитикам и проектировщикам, и включающей средства для поддержки графических моделей, проектирования спецификаций, экранных редакторов и словарей данных. Она не предназначена для поддержки полного ЖЦ разработки ПО и концентрирует внимание на функциональных спецификациях и начальных шагах проекта - предпроектном анализе, определении требований, системном проектировании, логическом проектировании БД и т.п. Средства CASE-II отличаются значительно более развитыми возможностями, улучшенными характеристиками и исчерпывающим подходом к полному ЖЦ. В этой технологии, в первую очередь, используются средства поддержки автоматической кодогенерации, а также обеспечивается полная функциональная поддержка для разработки системных требований и спецификаций проектирования; контроля, анализа и связывания системной информации, а также информации по управлению проектированием; построения прототипов и моделей системы; тестирования, верификации и анализа сгенерированных программ; генерации документов по проекту; контроля на соответствие стандартам по всем этапам ЖЦ.

Средства CASE-I не позволяют оптимизировать деятельность на уровне отдельных элементов проекта, средства CASE-II столкнулись с новой проблемой — проблемой организации взаимодействия между различными командами, реализующими проект. Унифицированный язык моделирования UML явился средством достижения компромисса между этими технологиями. Существует достаточное количество инструментальных средств, поддерживающих с помощью *UML* жизненный цикл информационных систем, и, одновременно, *UML* является достаточно гибким средством для настройки и поддержки специфики деятельности различных команд разработчиков.

Мощный толчок к разработке UML-технологий дало распространение *объектно-ориентированных* языков программирования в конце 1980-х — начале 1990-х годов. Пользователям хотелось получить единый *язык моделирования*, который объединил бы в себе всю мощь *объектно-ориентированного* подхода и давал бы четкую модель системы, отражающую все ее значимые стороны. Все шло к созданию единого языка, который объединял бы сильные стороны известных методов и обеспечивал наилучшую поддержку моделирования. Таким языком оказался *UML*. Создание *UML* началось в октябре 1994 г., когда Джим Рамбо и Гради Буч из Rational Software Corporation стали работать над объединением своих методов *OMT* и Booch. Осенью 1995 г. увидела свет первая черновая версия объединенной методологии, которую они назвали *Unified* Method 0.8. После присоединения в конце 1995 г. к Rational Software Corporation Айвара Якобсона и его фирмы Objectory, усилия трех создателей наиболее распространенных объектно-ориентированных методологий были объединены и направлены на создание *UML*.

В настоящее время консорциум пользователей *UML* Partners включает в себя представителей таких грандов информационных технологий, как Rational Software, Microsoft, IBM, Hewlett-Packard, Oracle, DEC, Unisys, IntelliCorp, Platinum Technology.

UML представляет собой *объектно-ориентированный язык моделирования*, обладающий следующими основными характеристиками:

- является языком *визуального моделирования*, который обеспечивает разработку *репрезентативных* моделей для организации взаимодействия заказчика и разработчика ИС, различных групп разработчиков ИС;
- содержит механизмы расширения и специализации базовых концепций языка.

UML — это стандартная нотация *визуального моделирования* программных систем, принятая консорциумом *Object Managing* Group (*OMG*) осенью 1997 г., и на сегодняшний день она поддерживается многими *объектно-ориентированными* CASE-продуктами. *UML* включает внутренний набор средств моделирования ("ядро"), которые сейчас приняты во многих методах и средствах моделирования. Эти средства необходимы в большинстве прикладных задач, хотя не каждая концепция необходима в каждой части каждого приложения. Пользователям языка предоставлены следующие возможности:

- строить модели на основе средств ядра, без использования механизмов расширения для большинства типовых приложений;
- добавлять при необходимости новые элементы и условные обозначения, если они не входят в ядро, или специализировать компоненты, систему условных обозначений (нотацию) и ограничения для конкретных предметных областей.

Синтаксис и семантика основных объектов UML

Классы

Классы — это базовые элементы любой *объектно-ориентированной* системы. *Классы* представляют собой описание совокупностей однородных объектов с присущими им свойствами — атрибутами, *операциями, отношениями* и семантикой.

В рамках модели каждому *классу* присваивается уникальное имя, отличающее его от других *классов*. Если используется *составное имя* (в начале имени добавляется *имя пакета*, куда входит *класс*), то имя *класса* должно быть уникальным в пакете.

Атрибут — это свойство *класса*, которое может принимать множество значений. Множество допустимых значений атрибута образует домен. Атрибут имеет имя и отражает некоторое свойство моделируемой сущности, общее для всех объектов данного *класса*. *Класс* может иметь произвольное количество атрибутов.

Операция — реализация функции, которую можно запросить у любого объекта *класса*. Операция показывает, что можно сделать с объектом. Исполнение операции часто связано с обработкой и изменением значений *атрибутов объекта*, а также изменением состояния объекта.

На рис. 3.1 приведено графическое изображение *класса* "Заказ" в нотации *UML*.

Рис. 3.1. Изображение класса в UML

Синтаксис *UML* для свойств *классов* (в отдельных программных средствах, например, в IBM *UML* Modeler, порядок записи параметров может быть иным):

<признак видимости> <имя атрибута> : <тип данных>
= <значение по умолчанию>
<признак видимости> <имя операции> <(список аргументов)>

Видимость свойства указывает на возможность его использования другими *классами*. Один *класс* может "видеть" другой, если тот находится в области действия первого и между ними существует явное или неявное отношение. В языке *UML* определены три уровня видимости:

- public (общий) — любой внешний *класс*, который "видит" данный, может пользоваться его общими свойствами. Обозначаются знаком " + " перед именем атрибута или операции;
- protected (защищенный) — только любой потомок данного *класса* может пользоваться его защищенными свойствами. Обозначаются знаком " # ";
- private (закрытый) — только данный *класс* может пользоваться этими свойствами. Обозначаются символом " - " .

Еще одной важной характеристикой атрибутов и операций *классов* является область действия. Область действия свойства указывает, будет ли оно проявлять себя по-разному в каждом экземпляре *класса*, или одно и то же значение свойства будет совместно использоваться всеми экземплярами:

- instance (экземпляр) — у каждого экземпляра *класса* есть собственное значение данного свойства;
- *classifier* (классификатор) — все экземпляры совместно используют общее значение данного свойства (выделяется на диаграммах подчеркиванием).

Возможное количество экземпляров *класса* называется его *кратностью*. В *UML* можно определять следующие разновидности *классов*:

- не содержащие ни одного экземпляра — тогда *класс* становится служебным (Abstract);
- содержащие ровно один экземпляр (Singleton);
- содержащие заданное число экземпляров;
- содержащие произвольное число экземпляров.

Принципиальное назначение *классов* характеризуют ***стереотипы***. Это, фактически, классификация объектов на высоком уровне, позволяющая определить некоторые основные свойства объекта. Стереотипы являются одним из трех типов механизмов расширяемости в унифицированном языке моделирования (UML). Они позволяют проектировщикам расширять словарь UML для создания новых элементов моделирования, получаемых из существующих, но имеющих определенные свойства, которые подходят для конкретной проблемы предметной области или для другого специализированного использования. Например, при моделировании сети могут понадобиться символы для представления маршрутизаторов и концентраторов. С помощью стереотипных узлов можно представлять их в виде примитивных строительных блоков.

Графически стереотип отображается как имя, заключенное в кавычки (" " или, если такие кавычки недопустимы, то “ “) и расположенное над именем другого элемента. В дополнение или в качестве альтернативы он может быть обозначен соответствующей иконкой. Значок может даже заменить весь символ UML. Например, стереотипы диаграммы классов могут быть использованы для описания методов поведения, таких как "конструктор".

Одной из альтернатив стереотипам, предложенной Петром Коудом в своей книге "Применение Java в моделировании цветом с UML: Организация и производство" является использование цветных архетипов. Архетипы, обозначенные UML-блоками разных цветов, могут быть использованы в сочетании со стереотипами. Это добавочное определение назначения показывает роль, которую играет объект UML в рамках более широкой программной системы.

Диаграммы классов

Классы в *UML* изображаются на *диаграммах классов*, которые позволяют описать систему в статическом состоянии — определить типы объектов системы и различного рода статические связи между ними.

Классы отображают типы объектов системы.

Между *классами* возможны различные отношения, представленные на рис. 3.2:

- зависимости, которые описывают существующие между *классами* отношения использования;
- обобщения, связывающие обобщенные *классы* со специализированными;
- ассоциации, отражающие структурные отношения между объектами *классов*.

Рис. 3.2. Отображение связей между классами

Зависимостью называется отношение использования, согласно которому изменение в спецификации одного элемента (например, *класса* " товар ") может повлиять на использующий его элемент (*класс* "строка заказа"). Часто зависимости показывают, что один *класс* использует другой в качестве аргумента.

Обобщение — это отношение между общей сущностью (родителем — *класс* "клиент") и ее конкретным воплощением (потомком — *классы* "корпоративный клиент" или "частный клиент"). Объекты *класса-потомка* могут использоваться всюду, где встречаются объекты *класса -родителя*, но не наоборот. При этом он наследует свойства родителя (его атрибуты и операции). Операция потомка с той же сигнатурой, что и у родителя, замещает операцию родителя; это свойство называют ***полиморфизмом***. *Класс*, у которого нет родителей, но есть потомки, называется корневым. *Класс*, у которого нет потомков, называется листовым.

Ассоциация — это отношение, показывающее, что объекты одного типа неким образом связаны с объектами другого типа ("клиент" может сделать "заказ"). Если между двумя *классами* определена ассоциация, то можно перемещаться от объектов одного *класса* к объектам другого. При необходимости направление навигации может задаваться стрелкой. Допускается задание ассоциаций на одном *классе*. В этом случае оба конца ассоциации относятся к одному и тому же *классу*. Это означает, что с объектом некоторого *класса* можно связать другие объекты из того же *класса*.

Ассоциации может быть присвоено имя, описывающее семантику отношений.

Каждая ассоциация имеет две роли, которые могут быть отражены на диаграмме (рис.3.3). Роль ассоциации обладает ***свойством множественности***, которое показывает, сколько соответствующих объектов может участвовать в данной связи.

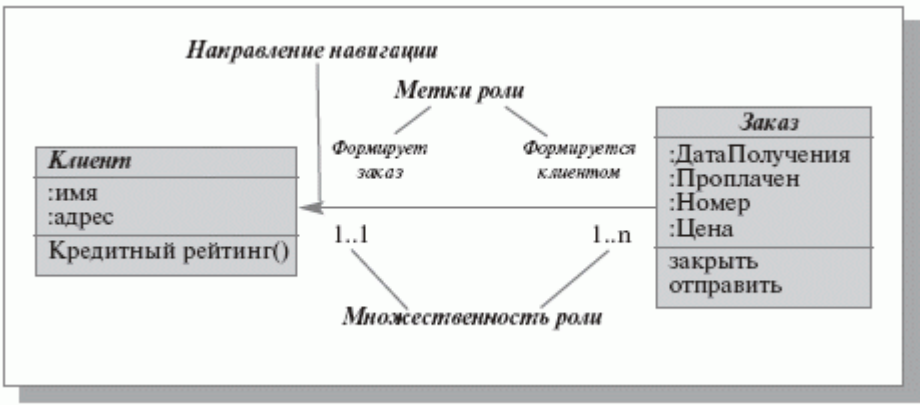


Рис. 3.3. Свойства ассоциации

Рисунок 3.3 иллюстрирует модель формирования заказа. Каждый заказ может быть создан единственным клиентом (множественность роли 1...1). Каждый клиент может создать один и более заказов (множественность роли 1..n). Направление навигации показывает, что каждый заказ должен быть "привязан" к определенному клиенту.

Такого рода ассоциация является простой и отражает отношение между равноправными сущностями, когда оба *класса* находятся на одном концептуальном уровне и ни один не является более важным, чем другой.

Если приходится моделировать отношение типа "часть-целое", то используется ***специальный тип ассоциации — агрегирование***. В такой ассоциации один из *классов*

имеет более высокий ранг (целое — *класс* "заказ", рис.1.2) и состоит из нескольких меньших по рангу *классов* (частей — *класс* "строка заказа").

В *UML* используется и более сильная разновидность *ассоциации — композиция*, в которой объект-часть может принадлежать только единственному целому. В композиции жизненный цикл частей и целого совпадают, любое удаление целого обязательно захватывает и его части.

Для ассоциаций можно задавать атрибуты и операции, создавая по обычным правилам *UML классы* ассоциаций.

Диаграммы использования. Диаграммы прецедентов

Диаграммы использования описывают функциональность ИС, которая будет видна пользователям системы. "Каждая функциональность" изображается в виде "*прецедентов* использования" (use case) или просто *прецедентов*. *Прецедент* — это типичное взаимодействие пользователя с системой, которое при этом:

- описывает видимую пользователем функцию,
- может представлять различные уровни детализации,
- обеспечивает достижение конкретной цели, важной для пользователя.

Прецедент обозначается на диаграмме овалом, связанным с пользователями, которых принято называть действующими лицами (актеры, *actors*). Действующие лица используют систему (или используются системой) в данном прецеденте. Действующее лицо выполняет некоторую роль в данном прецеденте. На диаграмме изображается только одно действующее лицо, однако реальных пользователей, выступающих в данной роли по отношению к ИС, может быть много. Список всех *прецедентов* фактически определяет *функциональные требования* к ИС, которые лежат в основе разработки технического задания на создание системы.

На *диаграммах прецедентов*, кроме связей между действующими лицами и прецедентами, возможно использование еще двух видов связей между прецедентами: "использование" и "расширение" (рис. 3.4). Связь типа "расширение" применяется, когда один *прецедент* подобен другому, но несет несколько большую функциональную нагрузку. Ее следует применять при описании изменений в нормальном поведении системы. Связь типа "использование" позволяет выделить некий фрагмент поведения системы и включать его в различные прецеденты без повторного описания.

Рис. 3.4. Связи на диаграммах прецедентов

На рис. 3.4 показано, что при исполнении прецедента "формирование заказа" возможно использование информации из предыдущего заказа, что позволит не вводить все необходимые данные. А при исполнении *прецедентов* "оценить риск сделки" и "согласовать цену" необходимо выполнить одно и то же действие — рассчитать стоимость заказа.

Прямоугольники на диаграмме представляют различные объекты и роли, которые они имеют в системе, а линии между *классами* отображают отношения (или ассоциации) между ними. Сообщения обозначаются ярлыками возле стрелок, они могут иметь нумерацию и показывать возвращаемые значения.

Динамические аспекты поведения системы отражаются приведенными ниже диаграммами.

В отличие от некоторых подходов объектного моделирования, когда и состояние, и поведение системы отображаются на *диаграммах классов*, *UML* отделяет описание поведения в *диаграммы взаимодействия*. В *UML диаграммы классов* не содержат сообщений, которые усложняют их чтение. Поток сообщений между объектами выносится на *диаграммы взаимодействия*. Как правило, *диаграмма взаимодействия* охватывает поведение объектов в рамках одного варианта использования.

Существуют два вида *диаграмм взаимодействия*:

- *диаграммы последовательностей*,

- *кооперативные диаграммы*.

Диаграммы последовательностей

Этот вид диаграмм используется для точного определения логики сценария выполнения прецедента. *Диаграммы последовательностей* отображают типы объектов, взаимодействующих при исполнении *прецедентов*, сообщения, которые они посылают друг другу, и любые возвращаемые значения, ассоциированные с этими сообщениями. Прямоугольники на вертикальных линиях показывают "время жизни" объекта. Линии со стрелками и надписями названий методов означают вызов метода у объекта. Пример диаграммы последовательности обработки заказа (рис. 3.5):

- вводятся строки заказа;
- по каждой строке проверяется наличие товара;
- если запас достаточен — инициируется поставка;
- если запас недостаточен — инициируется дозаказ (повторный заказ).

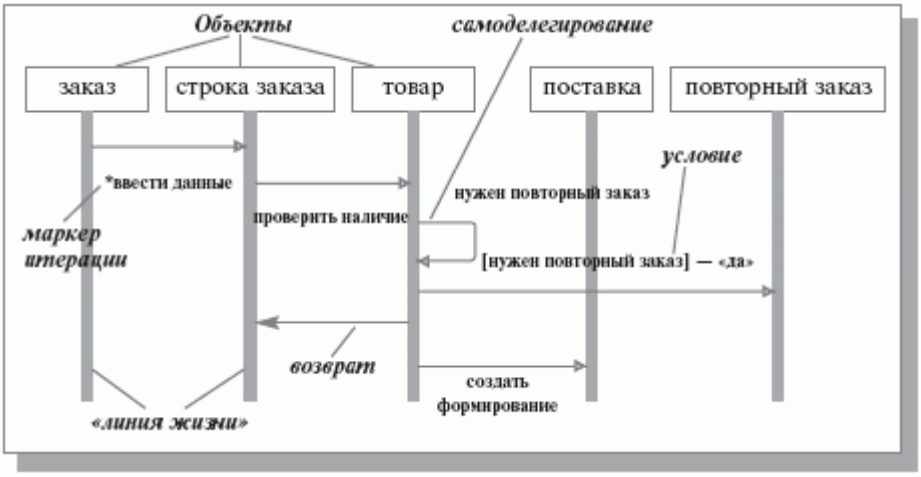


Рис. 3.5. Диаграмма последовательности обработки заказа

Сообщения появляются в той последовательности, как они показаны на диаграмме — сверху вниз. Если предусматривается отправка сообщения объектом самому себе (самоделегирование), то стрелка начинается и заканчивается на одной "линии жизни".

На диаграммы может быть добавлена управляющая информация: описание условий, при которых посылается сообщение; признак многократной отправки сообщения (маркер итерации); признак возврата сообщения.

Кооперативные диаграммы

На кооперативных *диаграммах объекты* (или *классы*) показываются в виде прямоугольников, а стрелками обозначаются сообщения, которыми они обмениваются в рамках одного варианта использования (рис. 3.6). Временная последовательность сообщений отражается их нумерацией.

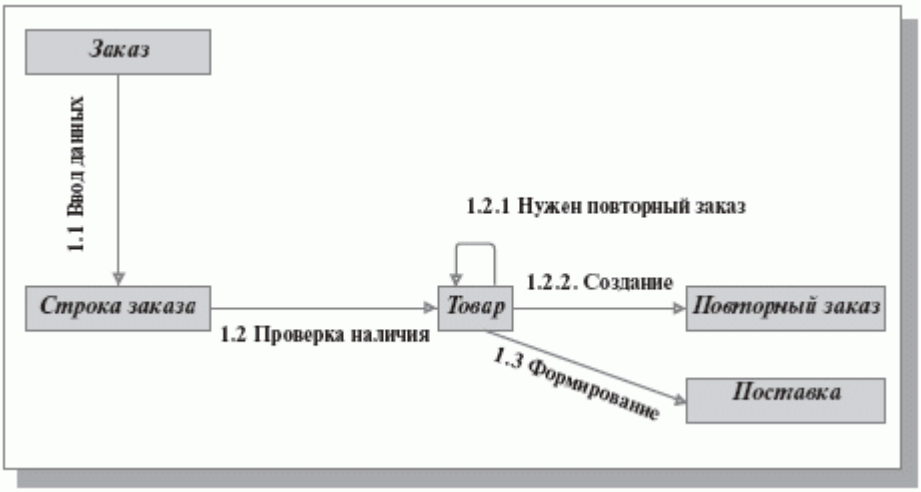


Рис. 3.6. Кооперативная диаграмма прохождения заказа

Диаграммы состояний

Диаграммы состояний используются для описания поведения сложных систем. Они определяют все возможные состояния, в которых может находиться объект, а также процесс смены состояний объекта в результате некоторых событий. Эти диаграммы обычно используются для описания поведения одного объекта в нескольких прецедентах.

Прямоугольниками представляются состояния, через которые проходит объект во время своего поведения. Состояниям соответствуют определенные значения *атрибутов объектов*. Стрелки представляют переходы от одного состояния к другому, которые вызываются выполнением некоторых функций объекта. Имеется также два вида псевдо-состояний: начальное состояние, в котором находится только что созданный объект, и конечное состояние, которое объект не покидает, как только туда перешел. Переходы имеют метки, которые синтаксически состоят из трех необязательных частей (рис. 3.7):

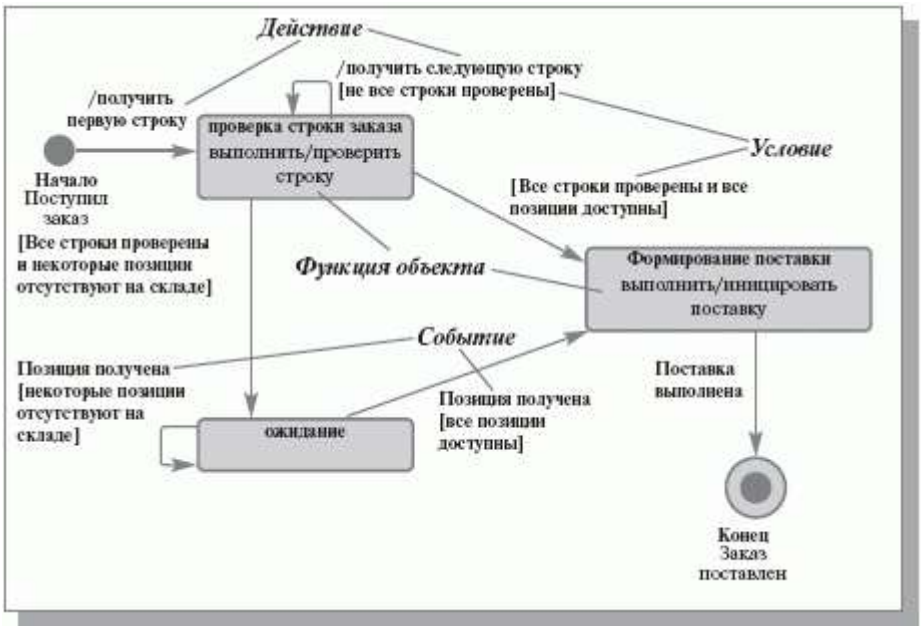


Рис. 1.7. Диаграмма состояний объекта "заказ"

<Событие> <[Условие]> < / Действие>.

На диаграммах также отображаются функции, которые выполняются объектом в определенном состоянии. Синтаксис метки деятельности:

выполнить/< деятельность >.

Диаграммы деятельности

Диаграмма деятельности — это частный случай *диаграммы состояний*. На диаграмме деятельности представлены переходы *потока управления* от одной деятельности к другой внутри системы. Этот вид диаграмм обычно используется для описания поведения, включающего в себя множество параллельных процессов.

Основными элементами диаграмм деятельности являются (рис. 3.8):

- овалы, изображающие действия объекта;
- линейки синхронизации, указывающие на необходимость завершить или начать несколько действий (модель логического условия "И");
- ромбы, отражающие принятие решений по выбору одного из маршрутов выполнения процесса (модель логического условия "ИЛИ");
- стрелки — отражают последовательность действий, могут иметь метки условий.

На диаграмме деятельности могут быть представлены действия, соответствующие нескольким вариантам использования. На таких диаграммах появляется множество начальных точек, поскольку они отражают теперь реакцию системы на множество внешних событий. Таким образом, диаграммы деятельности позволяют получить полную картину поведения системы и легко оценивать влияние изменений в отдельных вариантах использования на конечное поведение системы.

Рис. 3.8. Диаграмма деятельности — обработка заказа

Любая деятельность может быть подвергнута дальнейшей декомпозиции и представлена в виде отдельной диаграммы деятельности или спецификации (словесного описания).

Диаграммы компонентов

Диаграммы компонентов позволяют изобразить модель системы на физическом уровне. Элементами диаграммы являются компоненты — физические замещаемые модули системы. Каждый компонент является полностью независимым элементом системы. Разновидностью компонентов являются узлы. Узел — это элемент реальной (*физической) системы*, который существует во время функционирования программного комплекса и представляет собой вычислительный ресурс, обычно обладающий как минимум некоторым объемом памяти, а часто еще и способностью обработки.

Узлы делятся на два типа:

- устройства — узлы системы, в которых данные не обрабатываются.
- процессоры — узлы системы, осуществляющие обработку данных.

Для различных типов компонентов предусмотрены соответствующие стереотипы в языке *UML*.

Компонентом может быть любой достаточно крупный модульный объект, такой как таблица или экстент базы данных, подсистема, бинарный исполняемый файл, готовая к использованию система или приложение. Таким образом, *диаграмму компонентов* можно рассматривать как *диаграмму классов* в более крупном (менее детальном) масштабе. Компонент, как правило, представляет собой физическую упаковку логических элементов, таких как *классы*, интерфейсы и кооперации.

Основное назначение *диаграмм компонентов* — разделение системы на элементы, которые имеют стабильный интерфейс и образуют единое целое. Это позволяет создать ядро системы, которое не будет меняться в ответ на изменения, происходящие на уровне подсистем.

На рис. 3.9 показана упрощенная схема элементов фрагмента корпоративной системы. "Коробки" представляют собой компоненты — приложения или внутренние подсистемы. Пунктирные линии отражают зависимости между компонентами.

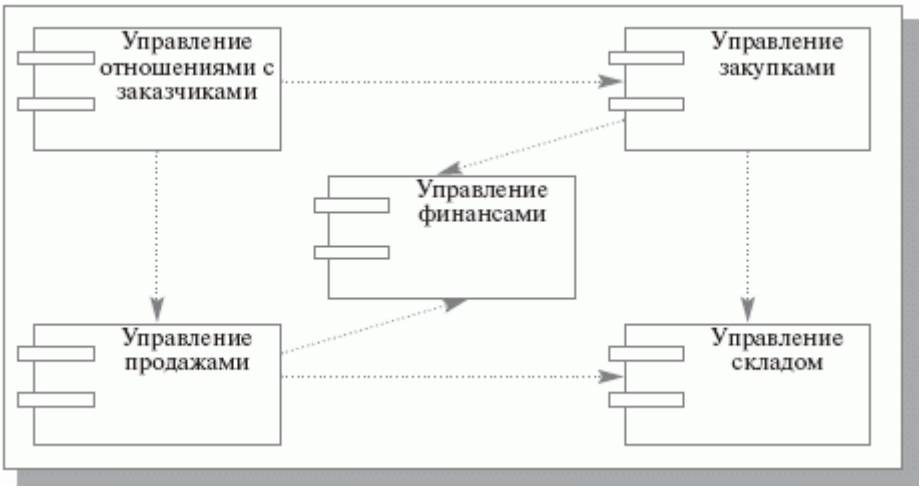


Рис. 3.9. Диаграмма компонентов фрагмента КИС

Каждый компонент диаграммы при необходимости документируется с помощью более детальной *диаграммы компонентов*, диаграммы сценариев или *диаграммы классов*.

Пакеты UML

Пакеты представляют собой универсальный механизм организации элементов в группы. В пакет можно поместить диаграммы различного типа и назначения. В отличие от компонентов, существующих во время работы программы, пакеты носят чисто концептуальный характер, то есть существуют только во время разработки. Изображается пакет в виде папки с закладкой, содержащей, как правило, только имя и иногда — описание содержимого.

Диаграмма пакетов содержит пакеты *классов* и зависимости между ними. Зависимость между двумя пакетами имеет место в том случае, если изменения в определении одного элемента влекут за собой изменения в другом. По отношению к пакетам можно использовать механизм обобщения (см. выше раздел "*Диаграммы классов*").

Этапы проектирования ИС с применением UML

UML обеспечивает поддержку всех *этапов жизненного цикла* ИС и предоставляет для этих целей ряд графических средств – диаграмм.

На этапе создания *концептуальной модели* для описания бизнес-деятельности используются *модели бизнес-прецедентов* и *диаграммы видов деятельности.*

На этапе описания *бизнес-объектов* используются – *модели бизнес-объектов* и *диаграммы последовательностей*.

На этапе создания логической модели ИС описание требований к системе задается в виде модели и описания *системных прецедентов*.

Предварительное проектирование осуществляется с использованием *диаграмм классов*, *диаграмм последовательностей и диаграмм состояний*.

На этапе создания *физической модели* детальное проектирование выполняется с использованием *диаграмм классов*, *диаграмм компонентов*, *диаграмм развертывания*.

Ниже приводятся определения и описывается назначение перечисленных диаграмм и моделей применительно к задачам *проектирования ИС* (в скобках приведены альтернативные названия диаграмм, использующиеся в современной литературе).

- *Диаграммы прецедентов* (*диаграммы вариантов использования*, use case diagrams) – это обобщенная модель функционирования системы в окружающей среде.
- *Диаграммы видов деятельности* (диаграммы деятельностей, *activity diagrams*) – модель бизнес-процесса или поведения системы в рамках *прецедента*.
- *Диаграммы взаимодействия* (interaction diagrams) – модель процесса обмена сообщениями между объектами, представляется в виде диаграмм последовательностей (*sequence diagrams*) или кооперативных диаграмм (*collaboration* diagrams).
- *Диаграммы состояний* (statechart diagrams) – модель динамического поведения системы и ее компонентов при переходе из одного состояния в другое.
- *Диаграммы классов* (*class diagrams*) – логическая модель базовой структуры системы, отражает статическую структуру системы и связи между ее элементами.
- *Диаграммы базы данных* (database diagrams) — модель *структуры базы данных*, отображает таблицы, столбцы, ограничения и т.п.
- *Диаграммы компонентов* (component diagrams) – модель иерархии подсистем, отражает физическое размещение баз данных, приложений и интерфейсов ИС.
- *Диаграммы развертывания* (диаграммы размещения, deployment diagrams) – модель физической архитектуры системы, отображает аппаратную конфигурацию ИС.

На рис. 3.10 показаны отношения между различными видами диаграмм UML. Указатели стрелок можно интерпретировать как отношение "является источником входных данных для..." (например, диаграмма *прецедентов* является источником данных для диаграмм видов деятельности и последовательности).

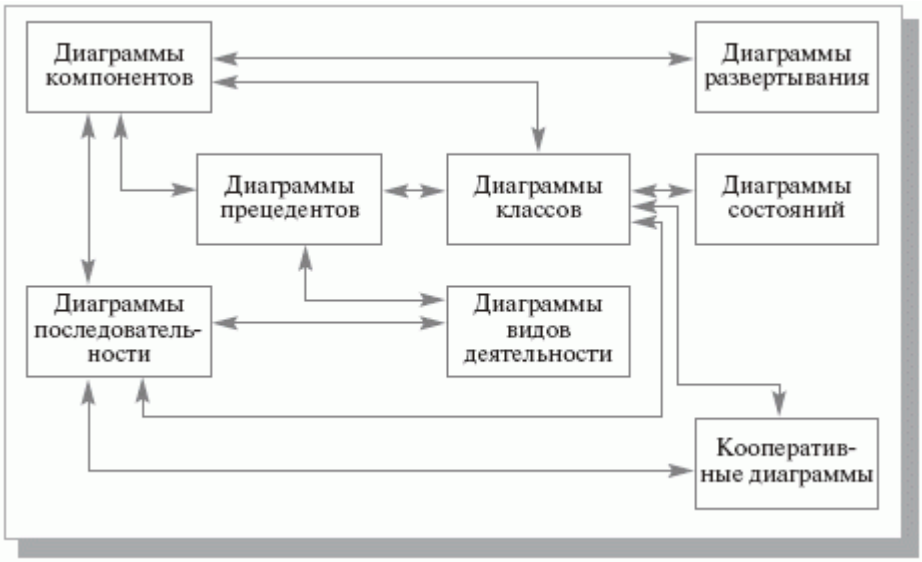


Рис. 3.10. Взаимосвязи между диаграммами UML

Приведенная схема является наглядной иллюстрацией итеративного характера разработки моделей с использованием UML.

Анализ требований и предварительное проектирование системы

Основные задачи этапа:

- определить проект системы, который будет отвечать бизнес-требованиям;
- разработать общий предварительный проект для всех команд разработчиков (проектировщиков баз данных, разработчиков приложений, системных архитекторов и др.).

Основным инструментом на данном этапе являются *диаграммы классов* системы, которые строятся на основе разработанной ***модели системных прецедентов***. Одновременно на этом этапе уточняются *диаграммы последовательностей* выполнения отдельных *прецедентов*, что приводит к изменениям в составе объектов и *диаграммах классов*. Это естественное отражение средствами UML итеративного процесса разработки системы.

Диаграммы классов системы заполняются объектами из *модели системных прецедентов*. Они содержат описание этих объектов в виде классов и описание взаимодействия между классами.

Пример диаграммы классов, описывающей процедуры *защиты доступа* к данным, приведен на рис. 3.11.

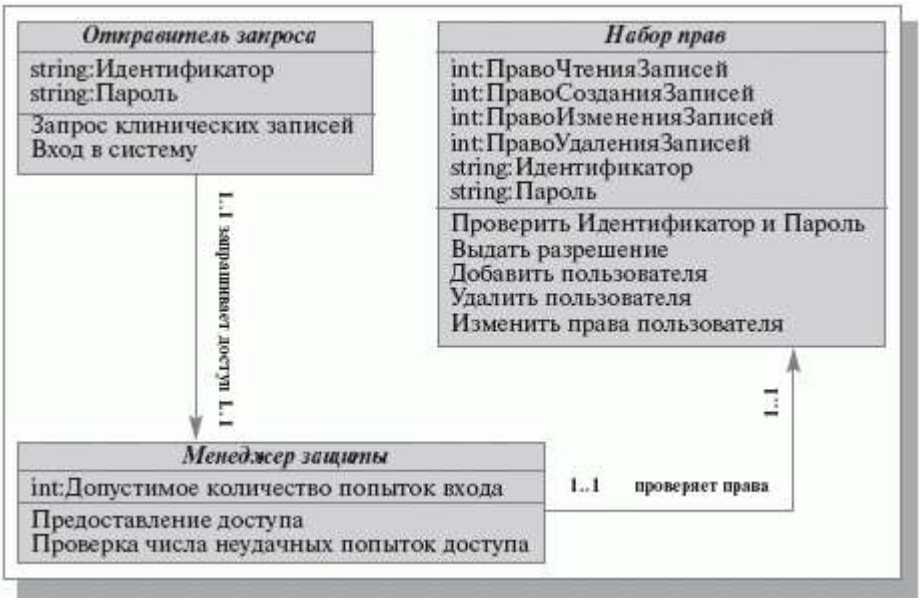


Рис. 3.11. Диаграмма классов "Защита доступа"

Таким образом, в результате этого этапа проектирования появляется достаточно подробное описание состава и функций проектируемой системы, а также информации, которую необходимо использовать в базе данных и в приложениях.

Поскольку *диаграммы классов* строятся на основе разработанных ранее *бизнес-моделей*, появляется уверенность в том, что разрабатываемая система будет действительно удовлетворять исходным *требованиям заказчика*. В то же время, благодаря своему синтаксису, *диаграммы классов* оказываются хорошим средством структурирования и *представления требований* к функциональности, интерфейсам и данным для элементов проектируемой системы.

Цель анализа требований к ИС заключается в разработке и подтверждении функциональных, нефункциональных требований, требований к внешним интерфейсам и безопасности системы, определении технических и программных ограничений, которые позволят наиболее точно произвести оценку объема работ и стоимости ИС.

Процесс сбора и анализа требований включает следующие шаги:

1) анализ и формирование программных требований;
2) определение архитектуры ИС;
3) определение ограничений реализации ИС;
4) анализ архитектуры аппаратного обеспечения, коммуникаций.

Анализ и формирование требований к ИС

Функциональные требования

Функциональные требования описывают действия, для выполнения которых ИС предназначена, и могут быть представлены в виде услуг, заданий или функций, которые надлежит выполнять системе.

Модель требований определяет комплекс взаимосвязанных функций между внешними участниками и рассматриваемой системой. Внешние участники - это объекты вне системы, которые взаимодействуют с системой. Внешними участниками могут быть пользователи, их роли (определяющие доступ к функциям системы, ресурсам системы и т.д.) или другие системы.

Главной целью определения функциональных требований является установка информационных границ для эффективного функционирования системы в рамках решаемых задач и полного комплекса функциональных возможностей, предоставляемых конечному пользователю.

Функциональные требования, как минимум должны включать:

- общесистемные требования, согласованные всеми заинтересованными представителями бизнес-процессов:

- характеристику ИС;
- основные цели создания ИС и ее назначение;
- планируемое развитие ИС;
- требования к ИС в целом;
- требования к структуре данных ИС;
- требования к функциональности ИС;
- требования к интерфейсу пользователя;
- требования к отчетам;
- требования к средствам администрирования;
- требования к средствам разработки;
- требования к взаимодействию с другими программными продуктами;
- требования к документированию;
- требования к программному и аппаратному обеспечению по объему требуемых ресурсов, быстродействию, надежности.

- описание спецификации различных приложений, в которых применяется исследуемая система;

- документирование форматов входных и выходных данных;
- методы реализации контроля подготовки, ввода данных и обработки ошибок ввода данных, позволяющих реализовать корректировку ошибок;
- методы реализации контроля проверки данных транзакций на точность, полноту и достоверность;
- методы обнаружения и исключения ошибочных транзакций по ходу их обработки;
- средства, поддерживающие неотказуемость транзакций;

- формализованные требования, согласованные всеми заинтересованными представителями бизнес-процессов:

- описание детальных спецификаций ПО в части функциональности системы;
- требования к разработке и документированию интерфейсов с внутренними подсистемами;
- требования к разработке и документированию алгоритмов обработки данных;
- требования обеспечения безопасности, целостности и доступности (в том числе при чрезвычайных ситуациях).

Требования к внешним интерфейсам

Требования к внешним интерфейсам ИС, в зависимости от конкретных условий эксплуатации, включают:

– требования к типу интерфейса (функционирование в режиме реального времени, передача данных, хранение и извлечение данных);

– требуемые характеристики к сервисам, которые должна реализовать ИС для обеспечения хранения, передачи, доступа, приема данных;

– требуемые характеристики структурных элементов данных (записи, сообщения, файлы, отчеты, запросы), которые ИС должна предоставлять, хранить, передавать, принимать и обеспечивать к ним доступ;

– требуемые характеристики для методов коммуникаций, которые ИС должна использовать для интерфейса;

– требования к характеристикам, которыми должны обладать протоколы ИС для взаимодействия;

– другие требуемые характеристики, в том числе, физическая совместимость взаимодействующих информационных объектов (размеры, интервалы, способы загрузки и т.д.).

Нефункциональные требования

Нефункциональные требования должны быть использованы для определения требований и ограничений ИС. Требования должны быть использованы в качестве основы на ранней стадии выполнения проекта по созданию системы и оценки ее жизнеспособности.

Нефункциональные требования включают:

– ограничения инструментальной среды и реализации;

– производительность (скорость, пропускная способность, время отклика, используемая память);

– зависимость от платформы;

– расширяемость;

– надежность (пригодность, точность получаемых результатов, средняя наработка на отказ, число ошибок в программном коде, число необработанных ситуаций).

Нефункциональные требования используются для предписания качественных атрибутов разрабатываемого приложения. Данные качественные атрибуты применяются для составления планов тестирования приложений.

Основные требования пользователей к информационным системам

ИС должны обеспечивать:

– гарантированный и безопасный доступ к информационным ресурсам;

– возможность поиска, сбора, обработки и хранения данных;

– возможность предоставления доступа к локальным и глобальным информационным ресурсам;

– надежность функционирования и устойчивость к программно–аппаратным сбоям, включая случаи некорректной работы пользователей;

– оперативный и безопасный обмен данными между пользователями;

– возможность дальнейшего расширения путем модернизации аппаратных и программных средств, наращивания системы новыми компонентами;

– функциональность, связанная с основным направлением деятельности пользователей.

Требования к ИС пользователей должны быть документированы и необходимо проверять соответствие работ по следующим категориям систем:

– транзакционные и учетные системы, обеспечивающие поддержку выполнения пользователями своих основных задач и функций;

– корпоративные системы информационного взаимодействия между ИС;

– системы управления ресурсами, обеспечивающими поддержку деловых процессов и административных регламентов в деятельности организаций;

– информационно–аналитические системы, обеспечивающие сбор, обработку, хранение и анализ данных о результатах выполнения пользователями своих основных задач и функций;

– системы электронного документооборота, в том числе, системы управления электронными архивами документов;

– системы управления эксплуатацией (включая системы управления отдельными компонентами);

– системы взаимодействия с физическими и юридическими лицами, обеспечивающими предоставление справочной информации и услуг через интернет или другие информационно - коммуникационные технологии, включая облачные вычисления, информационные порталы, мобильные приложения;

– системы обеспечения информационной безопасности;

– офисные системы, используемые работниками организации в своей производственной деятельности для подготовки документов и обмена информацией.

Общими требованиями к ИС являются:

– доступность, достоверность, полнота и своевременность предоставления информации;

– конфиденциальность - обеспечение разграничения доступа к информационным ресурсам ИС в пределах предоставленных пользователям полномочий (ролей);

– открытость - возможность интегрирования с другими ИС;

– масштабируемость - возможность дальнейшего расширения функциональных возможностей ИС путем добавления новых функций или расширения существующих;

– защищенность - обеспечение сохранности и целостности информационных ресурсов, входящих в состав ИС;

– надежность функционирования и устойчивость системы.

Требования по обеспечению информационной безопасности

Информационная безопасность в ИС должна обеспечиваться организационно–техническими и аппаратно – программными средствами.

Программно – аппаратные средства защиты информации должны быть лицензионными, сертифицированными и обеспечивать:

– идентификацию защищаемых информационных ресурсов;

– аутентификацию пользователей ИС;

– конфиденциальность информации, циркулирующей в системе;
– аутентифицированный обмен данными;
– целостность данных при формировании, передаче, использовании, обработке и хранении информации;
– авторизированную доступность всех ресурсов системы в условиях эксплуатации;
– разграничение доступа пользователей к ресурсам ИС;
– администрирование (назначение прав доступа к ресурсам ИС, обработка информации из регистрационных журналов, настройка параметров);
– регистрацию действий по входу и выходу пользователей, а также нарушений прав доступа к ресурсам ИС;
– контроль целостности и работоспособности системы обеспечения информационной безопасности;
– безопасность и бесперебойное функционирование системы в чрезвычайных ситуациях.

Определение архитектуры информационной системы

http://dit.isuct.ru/Publish_RUP/core.base_rup/roles/rup_designer_BA8DE74E.html

"Архитектура системы: единая фундаментальная структура системы, описанная в терминах поведения, ограничений, процессов, интерфейсов и элементов системы" [базовое определение, одобренное INCOSE Systems Architecture Working Group на конференции INCOSE 1996 в Бостоне (Массачусетс) 8 июля 1996 года].

Архитектура системы определяет взаимосвязь между наборами ее функций и компонентами аппаратного и программного обеспечения, взаимосвязь между архитектурой программного обеспечения и архитектурой аппаратного обеспечения и правила взаимодействия пользователей с этими компонентами.

Архитектура описывает следующие компоненты:

- *структуру* системы: элементы, компоненты и составляющие
- *взаимосвязь* между этими объектами
- *ограничения*, установленные для элементов и их взаимосвязей
- *поведение* системы и *взаимодействие* между элементами, необходимое для обеспечения такого поведения
- *принципы, правила* и *причины* создания системы такой, какая она есть
- физические и логические *характеристики* и *свойства* системы
- *предназначение* системы.

Для полного описания всех этих аспектов нужен богатый и потенциально сложный набор данных, однако необходимо помнить о том, что разные заинтересованные лица могут видеть и понимать эти аспекты по-разному. Концепция *точек наблюдения* позволяет создать несколько срезов архитектуры для того, чтобы показывать заинтересованным лицам только те аспекты, которые нужны им для эффективного участия в проекте. Кроме того, можно рассматривать систему в различных разрешениях - от низкого разрешения, соответствующего высокому *уровню абстракции* до высокого разрешения, при котором видны конкретные спецификации (компонентов и т.п.) для реализации.

Точки наблюдения

Точка наблюдения (в контексте системы) - это "форма абстракции, которая достигается с помощью определенного набора архитектурных концепций и правил структурирования для того, чтобы сфокусироваться на определенных аспектах системы" [ISO/IEC 10746-2: Information Technology - Open Distributed Processing - Reference Model: Foundations]. В таблице 3.1 приведены примеры точек наблюдения для различных аспектов системы. Эти точки наблюдения соответствуют стандарту ISO/IEC 10746-1: Information technology - Open Distributed Processing - Reference Model: Overview.

Таблица 3.1 - Примеры точек наблюдения для различных аспектов системы

Точка наблюдения	**Аспект**	**Сфера**
Исполнитель	Исполнителя интересуют роли и сферы ответственности организации и сотрудников (и установленные для них правила).	Деятельность исполнителей, взаимодействие пользователей с системой. Человеческий фактор и производительность труда.
Информация	Виды информации, обрабатываемой системой, и ограничения на использование и интерпретацию этой информации.	Целостность информации, ограничения на объем. Доступность и актуальность информации.
Логика	Декомпозиция системы на набор подсистем, взаимодействующих через интерфейсы и обеспечивающих выполнение функций системы.	Поведение системы отвечает требованиям. Система поддерживает расширение, адаптацию, обслуживание. Активы допускают повторное использование.
Комплектация и физические характеристики	Инфраструктура, необходимая для обеспечения нормальной работоспособности системы.	Соответствие физических характеристик системы предъявляемым к ней функциональным и нефункциональным требованиям.
Процесс	Параллелизм, расширяемость, производительность, пропускная способность, надежность.	Соответствие системы требованиям к времени отклика, пропускной способности и отказоустойчивости.

Это одни из самых распространенных точек наблюдения для систем, ориентированных на программное обеспечение. Для многих систем также требуются дополнительные точки наблюдения, характерные для их предметных областей. Примерами таких предметных областей могут служить безопасность, защита и механическая организация системы. Каждая точка наблюдения охватывает определенный набор характеристик, которые должны быть учтены при разработке архитектуры и проектировании системы. Если архитектура системы во многом зависит от потребностей или точек зрения отдельных заинтересованных лиц или экспертов, целесообразно создать специальные точки наблюдения для них.

Уровни абстракции

Помимо точек наблюдения, в архитектуре системы необходимо создать уровни спецификаций. По мере разработки архитектура становится все менее общей и все более детализированной. В методологии разработки программного обеспечения RUP (Rational Unified Process) различают четыре уровня архитектуры, перечисленные в таблице 3.2.

Таблица 3.2 – Уровни архитектуры по методологии RUP

Уровень модели	**Выражает**
Контекст	Система и ее субъекты.
Анализ	Описание начального набора компонентов системы на уровне концепции.
Эскиз	Реализация модели анализа на уровне аппаратного обеспечения, программного обеспечения и сотрудников.
Реализация	Реализация модели проекта в конкретных конфигурациях.

Проходя через эти уровни, проект системы превращается из абстрактной модели в конкретный план реализации. В модели контекста описываются все внешние элементы (субъекты), взаимодействующие с системой. Эти субъекты могут быть внешними или внутренними по отношению к организации, в которой развертывается система. В обоих случаях субъектами могут быть физические исполнители и другие системы. На уровне анализа разделы системы еще не привязаны к конкретным исполнителям, программному и аппаратному обеспечению. Вместо этого они отражают подходы к распределению функционального наполнения системы по отдельным компонентам. На уровне

проектирования происходит выбор типов аппаратного и программного обеспечения и ролей исполнителей. На уровне реализации принимаются решения относительно конкретных видов программного и аппаратного обеспечения, которые будут применяться для реализации системы. Например, на уровне проекта может быть установлено, что потребуется сервер данных. На уровне реализации происходит выбор конкретной платформы и системы управления базой данных.

Модели и диаграммы

Модель – это *представление* системы с определенного ракурса. Как правило, для системы создается несколько моделей, поскольку как на разработку, так и на эксплуатацию системы можно взглянуть с различных точек зрения. Модели могут создаваться на различных уровнях абстракции - как на сравнительно общих, на которых скрыты, или инкапсулированы, детали реализации, так и на сравнительно детальных, на которых отражены подробные характеристики системы.

Точка наблюдения, как следует из самого названия, это определенная "позиция", с которой наблюдатель видит определенные аспекты или компоненты системы. Для формирования точек наблюдения применяются фильтры, представляющие собой наборы концепций и правил. Как правило, для понимания системы недостаточно изучить модели существующих систем или систем, которые будут созданы в будущем.

Представления - это проекции моделей, в которые входят элементы, значимые для тех или иных точек наблюдения. Обычно для иллюстрации этих проекций применяются *диаграммы*.

Пересечения точек наблюдения и уровней абстракции содержат представления одной или нескольких моделей, значимых для соответствующих точек наблюдения на соответствующих уровнях абстракции.

Архитектуру системы можно представить в виде набора моделей (и соответствующих диаграмм), характеризующих архитектуру с различных уровней и точек наблюдения. Как показано в таблице 3.3, модели и диаграммы существуют не для всех сочетаний точек наблюдения и уровней абстракции. На уровне реализации одна модель охватывает реализацию как аппаратного, так и программного обеспечения для всех конфигураций системы.

Таблица 3.3 – Архитектура ИС с различных уровней и точек наблюдения

Уровень модели	**Точки наблюдения**				
	Исполнитель	**Логика**	**Информация**	**Комплектация**	**Процесс**

				и физические характеристики	
Контекст	Организационное представление UML	Диаграмма контекста системы	Представление данных организации	Представление локальности организации	Представление бизнес-процессов
Анализ	Общее представление исполнителей	Представление подсистем	Представление данных системы	Представление локальности системы	Представление процессов системы
Эскиз	Представление исполнителей	Представления классов подсистем Представления компонентов программного обеспечения	Схема данных системы	Представление узлов дескрипторов	Подробное представление процессов
Реализация	Инструкции и спецификации ролей исполнителей	Конфигурации: диаграммы развертывания с компонентами программного и аппаратного обеспечения системы.			

Многие представления из числа перечисленных в таблице построены на основе стандартных моделей UML. Например, на уровне анализа точки наблюдения логики система представлена в виде множества взаимодействующих подсистем, обеспечивающих выполнение требований пользователей. Подсистемы в данном случае используются в той же роли, что в стандарте UML. Эти подсистемы, в свою очередь, состоят из подсистем или (в конечном итоге) классов. Уровень проекта точки наблюдения логики соответствует подробному представлению модели классов. Модель процессов - это также стандартная модель классов UML, в которой основное внимание уделяется активным классам системы. В точки наблюдения определенных предметных областей также входят представления рабочих продуктов для одного или нескольких уровней. Набор рабочих продуктов проекта в данной среде должен представлять собой вариант разработки проекта.

Определение ограничений реализации ИС

На данном этапе необходимо провести анализ проекта и плана разработки или модернизации ИС, определить программные ограничения и требования, сформировать решения о покупке или разработке требуемых программных компонент. Определяется полный набор системных элементов, из которых конфигурируется ИС.

Необходимо рассмотреть архитектуру ИС и ее структурную схему с взаимосвязями между элементами, определить внешние интерфейсы по отношению к системе и между компонентами самой системы.

Описывается программное окружение (операционная среда) разрабатываемой ИС. Если разрабатываемая ИС является расширением уже существующей, то описываются главным образом ее дополнительные характеристики. Приводится перечень аппаратного и программного обеспечения, необходимого для работы ИС, решается задача о совместимости компонентов. Необходимо рассматривать несколько аспектов совместимости компонентов ИС: неоднородность программной среды, языки программирования, форматы данных и сообщений, форматы отчетов, совместимость с различным прикладным и системным ПО, которое используется для создания и функционирования ИС.

Анализ архитектуры аппаратного обеспечения, коммуникаций

На данном этапе проводится анализ характеристик аппаратного обеспечения ИС, которое обеспечивает эффективное функционирование системы. На этапе проектирования определяются общая архитектура аппаратного обеспечения, а также требования низкого уровня. Разработка общей архитектуры аппаратного обеспечения заключается в принятии ряда взаимосвязанных решений о выборе компонентов, которые обеспечивают требуемый режим работы ИС, необходимый уровень быстродействия и надежности. Разработка требований низкого уровня заключается в формировании требований к отдельным аппаратным компонентам, которые взаимодействуют с системным ПО.

Архитектура аппаратного обеспечения ИС включает описание:

- физических сущностей, из которых состоит система;
- информационных потоков между физическими сущностями;
- спецификаций каналов передачи информации;
- выбора платформы (платформ) и операционной системы (операционных систем);
- выбора архитектуры "файл–сервер" или "клиент–сервер";
- выбора 3–уровневой архитектуры со следующими слоями: сервер, ПО промежуточного слоя (сервер приложений), клиентское ПО;
- выбора базы данных централизованной или распределенной.

Результаты этого этапа формирования требований к ИС включают:

- определение аппаратных и программных требований и ограничений;
- определение функциональных, нефункциональных и требований по безопасности ИС;
- создание программной архитектурной иерархии модулей и функций системы;
- создание аппаратной архитектуры.

Разработка моделей базы данных и приложений

На этом этапе осуществляется отображение элементов полученных ранее моделей классов в элементы моделей базы данных и приложений:

- классы отображаются в таблицы;
- атрибуты – в столбцы;
- типы – в типы данных используемой СУБД;
- ассоциации – в связи между таблицами (ассоциации "многие-ко-многим" преобразуются в ассоциации "один-ко-многим" посредством создания дополнительных таблиц связей);
- приложения – в отдельные классы с окончательно определенными и связанными с данными в базе методами и атрибутами.

Поскольку модели базы данных и приложений строятся на основе единой логической модели, автоматически обеспечивается связность этих проектов (рис. 3.12).

Рис. 3.12. Связь между проектами базы данных и приложений

В модель базы данных отображаются только перманентные классы, из которых удаляются атрибуты, не отображаемые в столбцах (например, атрибут типа "Общий объем продаж", который получается суммированием содержимого множества полей базы данных). Некоторые атрибуты (например,АДРЕС) могут отображаться в множество столбцов (СТРАНА, ГОРОД, УЛИЦА, ДОМ, ПОЧТОВЫЙ ИНДЕКС).

Для каждого простого класса в модели базы данных формируется отдельная таблица, включающая столбцы, соответствующие атрибутам класса.

Отображение классов подтипов в таблицы осуществляется одним из стандартных способов:

- одна таблица на класс;
- одна таблица на суперкласс;
- одна таблица на иерархию.

В первом случае для каждого из классов создается отдельная таблица, между которыми затем устанавливаются необходимые связи. Во втором случае создается таблица для суперкласса, а затем в каждую таблицу подклассов включаются столбцы для каждого из атрибутов суперкласса. В третьем – создается единая таблица, содержащая атрибуты как суперкласса, так и всех подклассов (рис. 3.13). При этом для выделения исходных таблиц подклассов в результирующую таблицу добавляется один или более дополнительных столбцов (на рисунке показан курсивом).

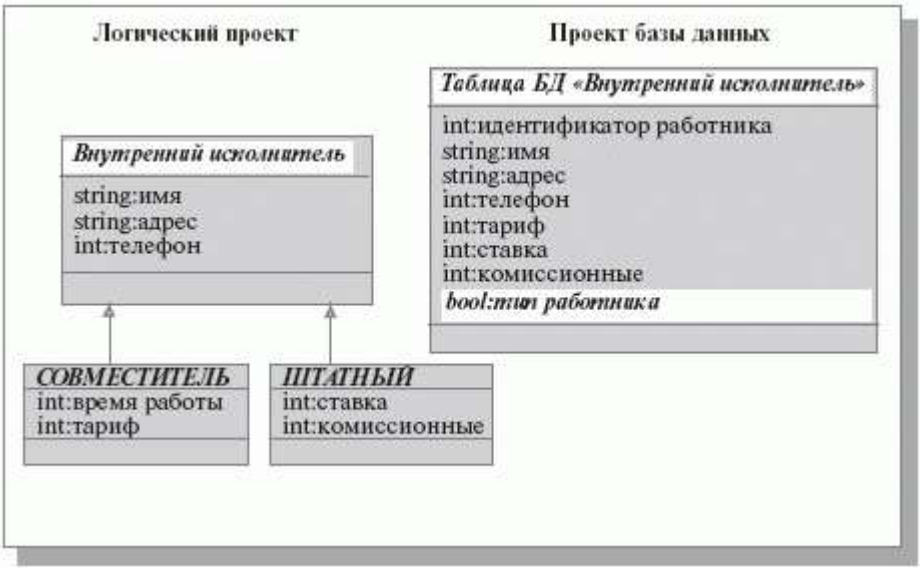


Рис. 3.13. Преобразование иерархии в таблицу

Разработка проекта базы данных осуществляется с использованием ***специального UML-профиля*** (Profile for Database Design), который включает следующие основные компоненты диаграмм:

- таблица – набор записей базы данных по определенному объекту;
- столбец – элемент таблицы, содержащий значения одного из атрибутов таблицы;
- первичный ключ (PK) – атрибут, однозначно идентифицирующий строку таблицы;
- внешний ключ (FK) – один или группа атрибутов одной таблицы, которые могут использоваться как первичный ключ другой таблицы;
- обязательная связь – связь между двумя таблицами, при которой дочерняя таблица существует только вместе с родительской;
- необязательная связь – связь между таблицами, при которой каждая из таблиц может существовать независимо от другой;
- представление – виртуальная таблица, которая обладает всеми свойствами обычной таблицы, но не хранится постоянно в базе данных;
- хранимая процедура – функция обработки данных, выполняемая на сервере;
- домен – множество допустимых значений для столбца таблицы.

На рис. 3.14 представлен фрагмент модели базы данных — две таблицы, соответствующие классам "пациент" и "минимальный набор данных". Связь между ними обязательная, поскольку "минимальный набор данных" не может существовать без "пациента".

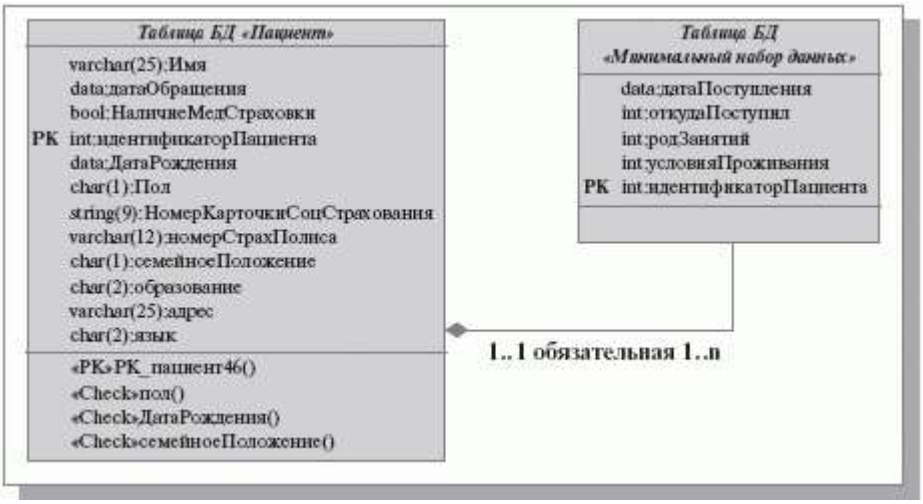


Рис. 3.14. Фрагмент модели базы данных

На диаграммах указываются дополнительные характеристики таблиц и столбцов:

- ограничения – определяют допустимые значения данных в столбце или операции над данными (ключ (PK,FK) – ограничение, определяющее тип ключа и его столбец; проверка (Check) – ограничение, определяющее правило контроля данных; уникальность (Unique) – ограничение, определяющее, что в столбце содержатся неповторяющиеся данные);
- триггер – программа, выполняющая при определенных условиях предписанные действия с базой данных;
- тип данных и пр.

Результатом этапа является детальное описание проекта базы данных и приложений системы.

Проектирование физической реализации системы

На этом этапе проектирования модели баз данных и приложений дополняются обозначениями их размещения на технических средствах разрабатываемой системы. Основными понятиями UML, которые используются на данном этапе, являются следующие:

- компонент – самостоятельный физический модуль системы;
- зависимость – связь между двумя элементами, при которой изменения в одном элементе вызывают изменения другого элемента;
- устройство – узел, не обрабатывающий данные;
- процессор – узел, выполняющий обработку данных;
- соединение – связь между устройствами и процессорами.

Диаграммы развертывания позволяют отобразить на единой схеме различные компоненты системы (программные и информационные) и их распределение по комплексу технических средств (рис. 3.15).

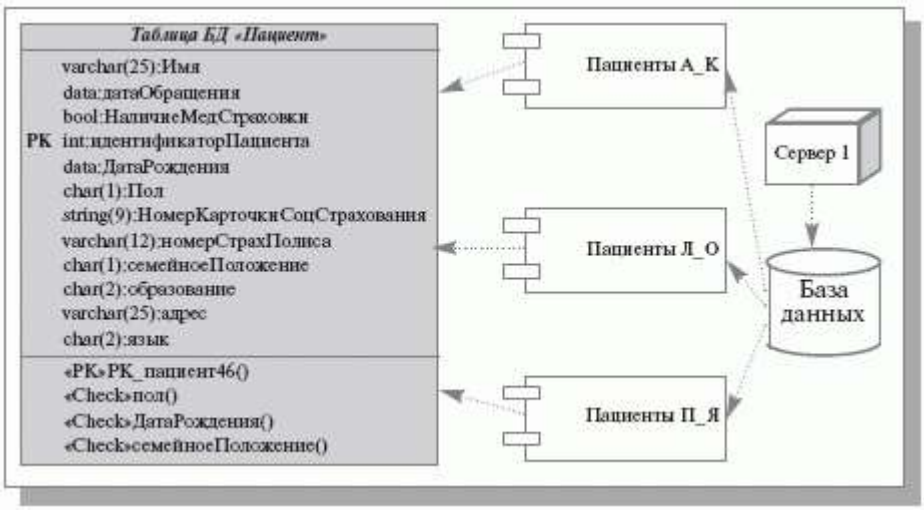


Рис. 12.15. Фрагмент диаграммы развертывания ИС

Таким образом, при проектировании сложной ИС она разделяется на части, и каждая из них затем исследуется и создается отдельно. В настоящее время используются два различных способа такого разбиения ИС на подсистемы: структурное (или функциональное) разбиение и объектная (компонентная) декомпозиция.

С позиций *проектирования ИС* суть функционального разбиения может быть выражена известной формулой:

" Программа = Данные + Алгоритмы ".

При функциональной декомпозиции системы ее структура описывается блок-схемами, узлы которых представляют собой "обрабатывающие центры" (функции), а связи между узлами описывают движение данных.

При объектном разбиении в системе выделяются "активные сущности" – объекты (или компоненты), которые взаимодействуют друг с другом, обмениваясь сообщениями и выполняя соответствующие функции (методы) объекта.

Если при проектировании ИС разбивается на объекты, то для ее визуального моделирования следует использовать UML. Если в основу проектирования положена функциональная декомпозиция ИС, то UML не нужен и следует использовать рассмотренные ранее структурные нотации. В то же время, при выборе подхода к разработке ИС следует учитывать, что визуальные модели все более широко используются в существующих технологиях управления проектированием систем, сложность, масштабы и функциональность которых постоянно возрастают. Они хорошо приспособлены для решения таких часто возникающих при создании систем задач как: физическое перераспределение вычислений и данных, обеспечение параллелизма вычислений, репликация БД, обеспечение безопасности, оптимизация балансировки

нагрузки ИС, устойчивость к сбоям и т.п. Визуализированные средствами UML модели ИС обеспечивают ясность представления выбранных архитектурных решений и позволяют понять разрабатываемую систему.

Оценка размера программной части ИС

Определение затрат на разработку ПО

В отличие от большинства отраслей материального производства, в вопросах проектов создания ПО недопустимы простые подходы, основанные на умножении трудоемкости на среднюю производительность труда. Это вызвано, прежде всего, тем, что экономические показатели проекта нелинейно зависят от объема работ, а при вычислении трудоемкости допускается большая погрешность. Поэтому для решения этой задачи используются комплексные и достаточно сложные методики, которые требуют высокой ответственности в применении и определенного времени на адаптацию (настройку коэффициентов).

Современные комплексные системы оценки характеристик проектов создания ПО могут быть использованы для решения следующих задач:

- предварительная, постоянная и итоговая оценка экономических параметров проекта: трудоемкость, длительность, стоимость;
- оценка рисков по проекту: риск нарушения сроков и невыполнения проекта, риск увеличения трудоемкости на этапах отладки и сопровождения проекта и пр.;
- принятие оперативных управленческих решений – на основе отслеживания определенных метрик проекта можно своевременно предупредить возникновение нежелательных ситуаций и устранить последствия непродуманных проектных решений.

Существуют различные метрики для оценки сложности и объема программ, ниже рассматриваются размерно - ориентированные метрики (показатели оценки объема), которые получили наибольшее распространение в практике создания программного обеспечения.

Размерно-ориентированные метрики (показатели оценки объема)

LOC-оценка (Lines Of Code).

Количество строк исходного кода (Lines of Code - LOC, Source Lines of Code - SLOC) — это метрика программного обеспечения, используемая для измерения его объёма с помощью подсчёта количества строк в тексте исходного кода. Как правило, этот показатель используется для прогноза трудозатрат на разработку конкретной программы на конкретном языке программирования, либо для оценки производительности труда уже после того, как программа написана. LOC является наиболее простым и распространенным способом оценки объема работ по проекту.

Промышленной мерой оценки размера ПО является количество тысяч строк исходного кода – ***KLOC*** (*kilo lines of code*).

Оценку размера ПО можно представить в виде 3-х шагов:

1) разделение и группировка программных функциональных требований и требований к ИС в категории программного наследования, которая определяет коэффициент сложности по реализации ПО. Категории программного наследования

включают:

- новый проект и новый код;
- аналогичный проект и новый код;
- аналогичный проект и многократно используемый код;
- аналогичный проект и расширенный код, используемый многократно.

2) оценка размера каждой программной функции.

Для определения количества тысяч строк исходного кода KLOC можно использовать следующую формулу:

$$KLOC = \frac{(SLOC_p \times K_p + SLOC_I \times K_i + SLOC_R \times K_r)}{1000},$$

где: $SLOC_P$ - код, обеспечивающий определенную логику работы системы. Обычно, это классы, описывающие свойства объектов системы, взаимосвязь между бизнес–функциями;

$SLOC_I$ - код, обрабатывающий элементы интерфейса и связывающий его с другими частями системы. Обычно это классы обработки ошибок ввода данных, записи данных в базу данных и т.п.;

$SLOC_R$ - код, реализующий интерфейс взаимодействия пользователя с системой;

K_p - фактор изменения логики работы системы;

K_i - фактор изменения обработки интерфейсных элементов системы;

K_r - фактор изменения интерфейса пользователя.

Значения факторов *K* подбираются опытным путем и измеряются обычно от 0,01 до 0,6, но для новой и сложной ИС часто принимается значение и свыше 0,6. Для определения значений *K* применяется статистический подход, который позволяет изменять априорные оценки с учетом данных новых исследований;

3) расчет общего размера программной части ИС в KLOC – сумма данных, полученных в пункте 2 по каждой программной функции.

В организациях, занятых разработкой программной продукции для каждого проекта принято регистрировать следующие показатели:

- общие трудозатраты (в человеко-месяцах, человеко-часах);
- объем программы (в тысячах строках исходного кода -KLOC);
- стоимость разработки;
- объем документации;
- количество людей, работавших над изделием;
- срок разработки.

На основе этих данных обычно подсчитываются простые метрики для оценки производительности труда (KLOC/человеко-месяц) и качества изделия.

Изначально показатель LOC возник как способ оценки объема работы по проекту, в котором применялись языки программирования, обладающие достаточно простой

структурой: "одна строка кода = одна команда языка". Как известно, что одну и ту же функциональность можно написать разным количеством строк, например, если возьмем язык высокого уровня (C++, Java), то возможно и в одной строке написать функционал 5-6 строк. Кроме того, современные средства программирования сами генерируют тысячи строк кода на простую операцию. Поэтому метод LOC является только примерным оценочным методом, который надо принимать к сведению, но не опираться в окончательных оценках.

Затраты на разработку программного обеспечения ИС по модели COCOMO

В настоящее время для оценки стоимости разработки программного обеспечения ИС применяются более современные подходы на основе модели конструктивных затрат Constructive COst Model (COCOMO – модель издержек разработки). Это алгоритмическая модель оценки стоимости разработки программного обеспечения, разработанная Барри Боэмом (Barry Boehm). Модель была впервые опубликована в 1981 году в книге Барри Боэма "Экономика разработки программного обеспечения".

Модель использует формулу регрессии с параметрами, определенными из данных, собранных по ряду проектов (базируется на результатах исследования 63 проектов аэрокосмической компании TRW).

В 1997 году была разработана модель COCOMO II и опубликована в 2000 году в книге "Оценка стоимости разработки ПО с COCOMO II". Модель COCOMO II является доработанной и подходящей для оценивания современных проектов разработки ПО, обеспечивает более полную поддержку современных процессов разработки ПО.

Режимы модели COCOMO

В модели COCOMO используются три режима, с помощью которых классифицируется сложность системы, размер программного продукта, сложность среды разработки (таблица 3.4).

Органический режим. Органический режим обычно предполагает решение отдельных вычислительных задач, когда работает небольшая команда по разработке проекта, требуются небольшие нововведения, имеются нестрогие ограничения и конечные сроки, а среда разработки является стабильной.

Сблокированный режим (*полуразделенный*). Сблокированный режим характеризуется прикладными системами, небольшой командой по разработке проекта среднего размера, требуются некоторые инновации, умеренные ограничения и конечные сроки, а среда разработки характеризуется относительной нестабильностью.

Внедренный режим (*встроенный*). Внедренный режим характеризуется режимами реального времени, например, системами контроля воздушного движения, диспетчерскими системами, военными системами. Другие характеристики: большая команда разработчиков проекта, большой объем требуемых инноваций, жесткие ограничения и сроки сдачи. Среда разработки в этом случае состоит из многих сложных интерфейсов, включая те из них, которые поставляются заказчикам вместе с аппаратным обеспечением.

Таблица 3.4 - Характеристики режимов COCOMO

Режим	**Размер программ-ного продукта**	**Проект/команда**	**Потреб-ность в инновациях**	**Срок сдачи и ограничения**	**Среда разработки**
Органический	Обычно 2-50 KLOC	Небольшой проект и команда – разработчики знакомы с инструментами и языком програм-мирования	Незначительная	Либеральные	Стабильная,
Сблокирован-ный (полураздe-ленный)	Обычно 50-300 KLOC	Средние проекты, средняя команда, обладающая средним уровнем возможностей	Средняя	Средние	Средняя
Внедренный (встроенный)	Обычно более 300 KLOC	Большие проекты, требующие большой команды	Максимальная	Серьезные ограничения	Сложный аппаратный интерфейс / Интерфейс пользовател я

Уровни модели COCOMO

Выделяется три уровня детализации характеристик проекта по разработке ПО, что обеспечивает последовательное повышение степени точности в оценке стоимости проекта на каждом последующем уровне.

Базовый уровень. На этом уровне для определения необходимых трудозатрат и графика разработки используются лишь значение размера программного кода и сведения о текущем режиме. Он пригоден при выполнении быстрых и приближенных оценок при выполнении небольших и средних по объему проектов.

Промежуточный уровень. На этом уровне применяются сведения о размере программного кода, режиме, а также 15 дополнительных переменных с целью более точного определения необходимых трудозатрат. Дополнительные переменные называются

"драйверами затрат" и имеют отношение к атрибутам продукта, персонала, компьютера и проекта, которые могут являться результатом более ли менее значительных трудозатрат. Произведение драйверов затрат называется корректировочным множителем среды EAF (Environmental adjustment factor).

Ниже приводятся 15 атрибутов, которым ставятся в соответствие дополнительные переменные (драйверы затрат), используемые для оценки стоимости разработки ПО:

Атрибуты продукта

1. Требуемая надежность ПО
2. Размер БД приложения
3. Сложность продукта

Атрибуты аппаратного обеспечения

4. Ограничения быстродействия при выполнении программы
5. Ограничения памяти
6. Неустойчивость окружения виртуальной машины
7. Требуемое время восстановления

Атрибуты персонала

8. Аналитические способности
9. Опыт разработки
10. Способности к разработке ПО
11. Опыт использования виртуальных машин
12. Опыт разработки на языках программирования

Атрибуты проекта

13. Применение методов разработки ПО
14. Использование инструментария разработки ПО
15. Требования соблюдения графика разработки

Детализированный уровень. Этот уровень надстраивается на промежуточный уровень COCOMO путем внедрения дополнительных множителей трудозатрат, чувствительных к фазе, и трехуровневой иерархии программных продуктов. Промежуточный уровень может быть настроен по фазе и по уровню разработки продукта с целью достижения детализированного уровня. В качестве примера множителей трудозатрат, чувствительных к фазе, можно рассматривать ограничения по памяти, которые могут применяться при попытках оценивания фаз кодирования или тестирования проекта. Множители, чувствительные к фазе, обычно используются специализированными организациями и требуют применения автоматизированных инструментов.

Ниже приводятся формулы для определения показателей проекта на разных уровнях модели COCOMO.

Базовый уровень.

Базовые уравнения COCOMO:

Трудоемкость = a (KLOC)^b **[человеко-месяцев],**

Срок разработки или длительность = c (Трудоемкость)^d **[месяцев],**

Число разработчиков = Трудоемкость/ Срок разработки **[человек],**

где KLOC – размер исходного кода программы в тысячах строк.

Коэффициенты *a*, *b*, *c* и *d* приведены в таблице 3.4.

Таблица 3.4 - Коэффициенты модели COCOMO Базового уровня

Тип проекта	***a***	***b***	***c***	***d***
Органический	2.4	1.05	2.5	0.38
Сблокированный (промежуточный)	3.0	1.12	2.5	0.35
Внедренный (встроенный)	3.6	1.20	2.5	0.32

Коэффициенты ***a, b, c, d*** могут определяться с помощью процедуры построения кривой по точкам (регрессионный анализ), причем данные проекта сравниваются с помощью уравнения. Этот метод может применяться для описания многих программных проектов. Ниже приводятся формулы расчета трудозатрат с использованием рекомендуемых значений коэффициентов.

Базовые формулы оценки трудоемкости для трех режимов модели COCOMO представляются следующим образом:

Трудоемкость для органического режима: $E=2,4\times(KLOC)^{1,05}$

Трудоемкость для сблокированного режима: $E=3,0\times(KLOC)^{1,12}$

Трудоемкость для внедренного режима: $E=3,6\times(KLOC)^{1,20}$

В случае использования различных режимов проекты одинакового масштаба требуют различных трудозатрат.

Оценка длительности базового проекта COCOMO

Оценка длительности проекта при использовании базовой модели COCOMO производится по следующим формулам:

Длительность проекта в органическом режиме: $TDEV=2,5\times(E)^{0,38}$

Длительность проекта *в сблокированном* режиме: $TDEV=2,5\times(E)^{0,35}$

Длительность проекта во внедренном режиме: $TDEV=2,5\times(E)^{0,32}$

В таблице 3.5 представлены базовые формулы, применяемые для оценки трудозатрат и времени разработки в каждом режиме.

Таблица 3.5 - Базовые формулы оценки необходимых для разработки времени и трудозатрат в модели COCOMO

Режим	a	b	Формула для оценки трудозатрат $E=a\times KLOC^b$ (человеко-месяцы)	c	d	Формула для определения времени разработки $TDEV=a\times E^b$ (месяцы)
Органический	2,4	1,05	$E=2,4\times(KLOC)^{1,05}$	2,5	0.38	$TDEV=2,5\times(E)^{0,38}$
Сблокированный (полуразделенный)	3,0	1,12	$E=3,0\times(KLOC)^{1,12}$	2,5	0.36	$TDEV=2,5\times(KLOC)^{0,36}$
Внедренный (встроенный)	3,6	1,20	$E=3,6\times(KLOC)^{1,20}$	2,5	0.32	$TDEV=2,5\times(KLOC)^{0,32}$

Если известны трудоемкость (E) и время разработки (TDEV), может быть вычислена средняя численность персонала (SS), необходимого для завершения проекта при использовании базовой модели COCOMO:

средняя численность персонала = трудозатраты/ время разработки, т.е. SS = E/TDEV

Если известна средняя численность персонала (SS), может быть определен уровень производительности (P) для базовой модели COCOMO:

производительность = размер кода /трудозатраты или **P = KLOC/E.**

В базовой модели COCOMO предлагается метод быстрых оценок трудозатрат, времени разработки, количества персонала, а также производительности. При этом исходными являются сведения о размере кода и режиме базовой модели. Для вычисления достаточно использовать обычный калькулятор. Однако достаточно просто выполнить оценку трудозатрат на базовом уровне, но полученные при этом результаты будут весьма приблизительными.

Метод быстрых оценок широко применяется на начальных этапах разработки ИС.

С целью улучшения процесса оценки Барри Боэм разработал руководство по "настройке" точности метода с помощью фактора корректировки сложности, описанного в промежуточной модели COCOMO.

Промежуточный уровень

Оценка трудозатрат в промежуточной модели COCOMO

В промежуточной модели COCOMO используются значения размера кода программы, а также режимы, которые применялись в базовой модели. Дополнительно применяются 15 переменных, называемых драйверами затрат, с помощью которых могут быть объяснены и модифицированы уравнения трудозатрат (таблица 1.7). Применение драйверов затрат позволяет управлять затратами (трудозатратами) проекта в зависимости от характеристик данного проекта.

Входными данными в промежуточной модели COCOMO являются показатели KLOC (аналогично базовой модели) и значения драйверов затрат, с помощью которых производится корректировка и улучшение оценки.

Драйверы затрат

Концепция, связанная с фактором корректировки трудозатрат, заключается в том, что он создает эффект увеличения либо уменьшения трудозатрат, а следовательно, и затрат, в зависимости от набора факторов среды. Факторы среды иногда называются факторами корректировки затрат [C_i] либо драйверами затрат. Определение этого фактора-множителя происходит в два этапа.

На этапе 1 драйверам затрат назначаются числовые значения.

На этапе 2 происходит перемножение драйверов затрат, в результате чего определяется фактор корректировки трудозатрат EAF (Effort adjustment factor, EAF).

Факторы корректировки затрат могут сказываться на оценках графика и затрат проекта, изменяя их в 10 и более раз!

Таким образом, произведение драйверов затрат C_i образует фактор корректировки затрат:

$EAF = C$,

где $C = C_1 \times C_2 \times \ldots \times C_n$

C_i = степень фактора корректировки затрат

$C_i = 1$ – драйвер затрат не применим

$C_i > 1$ – драйвер затрат увеличивает затраты

$C_i < 1$ – драйвер затрат уменьшает затраты

n – число учитываемых драйверов затрат (дополнительных переменных).

Общая формула определения трудозатрат для промежуточной модели COCOMO с учетом фактора корректировки затрат:

$$E = a \times (KLOC)^b \times C$$

Формула для промежуточной модели COCOMO (коэффициенты и экспоненты, изменены по сравнению с базовой моделью):

Трудоемкость для органического режима: $E = 3{,}2 \times (KLOC)^{1{,}05} \times C$

Трудоемкость для сблокированного режима: $E = 3{,}0 \times (KLOC)^{1{,}12} \times C$

Трудоемкость для внедренного режима: $E = 2{,}8 \times (KLOC)^{1{,}20} \times C$

В приведенных формулах коэффициенты *a* и *b* различаются для каждого режима, что показано в сводной таблице 4.

Таблица 3.6 - Формулы для оценки трудозатрат в промежуточной модели COCOMO с рекомендуемыми значениями коэффициентов

Режим	a	b	Формула для оценки трудозатрат $E=a\times(KLOC)^{b}\times C$
Органический	3,2	1,05	$E=3,2\times(KLOC)^{1,05}\times C$
Сблокированный (промежуточный)	3,0	1,12	$E=3,0\times(KLOC)^{1,12}\times C$
Внедренный (встроенный)	2,8	1,20	$E=2,8\times(KLOC)^{1,20}\times C$

Драйверы затрат выбираются в соответствии с их общей значимостью для всех программных проектов, причем они являются независимыми от размера проекта. Драйверы затрат группируются в виде четырех категорий, как показано в таблице 3.7. В скобках приводятся принятые обозначения переменных.

Таблица 3.7 - Категории драйверов затрат в промежуточной модели COCOMO в соответствии с атрибутами

Программный продукт	**Компьютер**	**Персонал**	**Проект**
Требуемая надежность ПО (RELY)	Ограничения времени выполнения (TIME)	Способности аналитика (ACAP)	Использование современных методов (MODR)
Размер базы данных (DATA)	Ограничения объема основной памяти (STOR)	Опыт в создании приложений (AEXP)	Использование программных инструментов (TOOL)
Сложность программного продукта (CPLX)	Изменчивость виртуальной машины (VIRT)	Способности программиста (PCAP)	Требуемые сроки разработки (SCED)
	Время реакции компьютера (TURN)	Опыт в области виртуальных машин (VEXP)	
		Опыт в области языков программи-рования (LEXP)	

Каждый драйвер затрат определяет умножающий фактор, который позволяет оценить эффект действия атрибута на величину трудозатрат. Числовые значения драйверов затрат при их совместном перемножении образуют фактор корректировки трудозатрат. Ниже приводится произведение драйверов затрат (дополнительных переменных) в соответствии с их предметным обозначением (таблица 5):

$$C = RELY \times DATA \times CPLX \times TIME \times STOR \times VIRT \times TURN \times ACAP \times AEXP \times PCAP \times VEXP \times LEXP \times MODP \times TOOL \times SCED.$$

Драйверы затрат выбираются в зависимости от атрибутов проекта. Ниже указаны некоторые рекомендации учета атрибутов в зависимости от условий реализации ИС.

Атрибуты программного продукта

- требуемая надежность ПО (обычно применяется в системах реального времени);
- размер базы данных приложения;
- сложность продукта (связана с ограничениями на время выполнения).

Атрибуты, связанные с аппаратными средствами

- ограничения времени выполнения – применяются в том случае, когда быстродействие процессора является ограниченным;
- ограничения объема основной памяти – применяются в случае, когда размер памяти является ограниченным;
- изменчивость виртуальной машины;
- время реакции компьютера – среднестатистическое время от момента передачи задания на выполнение и до момента завершения его выполнения.

Атрибуты проекта

- использование современных методов (например, объектно– ориентированные технологии);
- использование программных инструментов – CASE-инструменты, хорошие отладчики, инструменты, используемые при выполнении тестирования;
- требуемые сроки разработки (например, могут либеральные, средние или очень жесткие).

Атрибуты персонала

- способности аналитика;
- опыт в создании приложений;
- способности программиста;
- опыт в области виртуальных машин, включая операционную систему и аппаратное обеспечение;
- опыт в области языков программирования, включая инструменты и практику.

Другие драйверы затрат

Наиболее часто в рамках промежуточной модели COCOMO используются указанные четыре категории атрибутов, но менеджер проекта может еще добавлять дополнительные атрибуты, например:

- изменяемость требований – некоторые из них являются ожидаемыми, однако большинство из них могут представлять значительную проблему;
- изменяемость программных средств, предназначенных для разработки – нестабильные ОС, компиляторы, CASE-инструменты и т.д.;
- требования безопасности – применяются для классифицированных программ;
- доступ к данным – иногда является весьма затрудненным;
- влияние стандартов и навязанных методов;
- влияние физического окружения.

Некоторые из атрибутов, которые могут изменять величину затрат проекта могут применяться наравне с самим продуктом или выполняться в ходе соответствующей

работы. В таблице 3.8 приводится пример значений драйверов затрат с учетом конкретных показателей проекта и расчета фактора корректировки трудозатрат EAF.

Таблица 3.8 – Пример значений драйверов затрат для промежуточной модели COCOMO

Драйвер затрат	**Применение**	**Оценка**	**Множитель трудозатрат**
RELY	Локальное применение системы. Не возникают серьезные проблемы с восстановлением данных	Номинальная	1,00
DATA	16 Гбайт	Низкая	0,94
CPLX	Обработка коммуникаций	Очень высокая	1.30
TIME	Трехкратный запас по времени выполнения	Высокая	1,11
STOR	120 Гбайт из 160 Гбайт доступного хранилища (75 %)	Высокая	1.06
VIRT	Основано на коммерческом микропроцессорном аппаратном обеспечении	Номинальная	1,00
TURN	Среднее время обхода равно 10 секунд	Номинальная	1,00
ACAP	Опытный старший аналитик	Высокая	0,86
AEXP	3-летний опыт	Номинальная	1,00
PCAP	Опытные старшие программисты	Высокая	0,86
VEXP	6 месяцев	Низкая	1,10
LEXP	12 месяцев	Номинальная	1,00
MODP	Большинство технологий применяется более одного года	Высокая	0,91
TOOL	На уровне базового миникомьютерного инструмента	Низкая	1,10
SCED	10 месяцев	Номинальная	1,00
EAF	C = 1,00 × 0,94 × 1,30 × 1,11 × 1,06 × 1,00 × 1,00 × 0,86 × 1,00 × 0,86 × 1,10 × 1,00 × 0,91 × 1,10 × 1,00	C=1,17	

Детализированный уровень

Детализированный уровень включает в себя все характеристики промежуточного (среднего) уровня с оценкой влияния данных характеристик на каждый этап процесса разработки ПО. Каждому из указанных атрибутов ставится в соответствие рейтинг по шести бальной шкале, начиная от "очень низкий" и до "сверхвысокого" (по значению или важности атрибута). Далее значения рейтинга заменяются множителями трудоемкости из таблицы 3.9.

Таблица 3.9 – Рекомендуемые значения драйверов затрат при разработке ПО в рамках модели COCOMO

Драйверы затрат	**Показатели**					
	Очень низкий	**Низкий**	**Номинальный**	**Высокий**	**Очень высокий**	**Сверхвысокий**
Атрибуты продукта						
Требуемая надежность ПО (RELY)	0,75	0,88	1,00	1,15	1,40	
Размер базы данных (DATA)		0,94	1,00	1,08	1,16	
Сложность программного продукта (CPLX)	0,70	0,85	1,00	1,15	1,30	1,65
Ограничения времени выпол-нения (TIME)			1,00	1,11	1,30	1,66
Ограничения объема основной памяти (STOR)			1,00	1,06	1,21	1,56
Изменчивость виртуальной машины (VIRT)		0,87	1,00	1,15	1,30	
Время реакции компьютера (TURN)		0,87	1,00	1,07	1,15	
Атрибуты персонала						
Способности аналитика (ACAP)	1,46	1,19	1,00	0,86	0,71	
Опыт в создании приложений (AEXP)	1,29	1,13	1,00	0,91	0,82	
Способности	1,42	1,17	1,00	0,86	0,70	

Драйверы затрат	Показатели					
	Очень низкий	**Низкий**	**Номинальный**	**Высокий**	**Очень высокий**	**Сверхвысокий**
программиста (PCAP)						
Опыт в области виртуальных машин (VEXP)	1,21	1,10	1,00	0,90		
Опыт в области языков программирования (LEXP)	1,14	1,07	1,00	0,95		
Атрибуты проекта						
Использование современных методов (MODP)	1,24	1,10	1,00	0,91	0,82	
Использование программных инструментов (TOOL)	1,24	1,10	1,00	0,91	0,82	
Требуемые сроки разработки (SCED)	1,23	1,08	1,00	1,04	1,10	

Предварительная оценка затрат на разработку программного обеспечения ИС с применением модели COCOMO определяется в следующей последовательности:

1) Трудоемкость = a $(KLOC)^b$ ×C [человеко-месяцев],

   где $C = C_1 \times C_2 \times \ldots \times C_n$,
   C_i - драйвер затрат (степень фактора корректировки затрат);
   n – число учитываемых драйверов затрат.
2) Срок разработки или длительность = *c* $(Трудоемкость)^d$ [месяцев]
3) Число разработчиков = Трудоемкость/ Срок разработки [человек]
4) Объем затрат = Число разработчиков*средняя зарплата разработчика в месяц *срок разработки

Оценка эффективности проекта на разработку и внедрение ИС

Каждая стадия проекта разделяется на ряд этапов и работ, предусматривается возможность согласования любой разработанной части и составление документации, отражающей результаты работы. Пример последовательности выполнения этапов и работ проекта на создание ИС с учетом длительности представлен в таблице 3.10.

Таблица 3.10 - Пример выполнения этапов и работ проекта на создание ИС с учетом длительности

Наименование работ	Общая длительность работы, дней	Длительность работ, дней	
		Разработчик(и)	Руководитель
I - этап. Разработка технического задания			
1.1. Получение и согласование задания на разработку ИС			
1.2. Сбор данных для проектирования			
1.3. Анализ материалов			
1.4. Формирование требований			
1.5. Определение структуры входных и выходных данных			
1.6. Утверждение технического задания			
1.7. Утверждение технико-экономического обоснования			
1.8. Утверждение части по безопасности и экологичности проекта			
1.9. Согласование метода реализации			
Всего			
II - этап. Эскизный проект			
2.1. Разработка эскизов экранных форм			
2.2. Разработка общего описания алгоритма решения задачи			
Всего			
III - этап. Технический проект			

3.1.Определение форм входных и выходных данных			
3.2. Разработка структуры программы			
3.3. Согласование структуры программы			
3.4. Разработка системных моделей			
3.5. Согласование программного изделия			
Всего			
IV - этап. Рабочий проект			
4.1. Разработка методики испытаний			
4.2. Разработка программного обеспечения (с учетом оценки трудозатрат на разработку ПО)			
4.3. Тестирование и отладка ПО			
4.4. Разработка документации			
4.5. Утверждение документации			
Всего			
V - этап. Ввод в действие ИС			
5.1. Подготовка и передача программы			
5.2. Обучение персонала			
Всего			
Итого			

Продолжительность разработки ИС с целью достижения оптимального результата учитывает обязательные консультации со специалистами, непосредственно связанными с темой проекта. Также вносятся изменения в проект с учетом их требований и рекомендаций.

Определение затрат по проекту и себестоимости услуг, предоставляемых предлагаемой ИС

На практике для определения всех затрат на проект за определенный период времени применяют термин "затраты на производство". Издержки, относящиеся к выпущенной продукции (услугам), выражаются в ее себестоимости. Себестоимость продукции (услуг) представляет собой часть стоимости этой продукции, включающую потребленные ресурсы (время, финансы, материалы и т.д.).
Состав затрат, включаемых в себестоимость продукции, определяется в соответствии с действующими нормативами и классификаторами затрат. Положением о составе затрат определено, что себестоимость продукции (работ, услуг) представляет собой стоимостную оценку используемых в процессе производства продукции (работ, услуг) природных ресурсов, сырья, материалов, топлива, энергии, основных фондов, трудовых ресурсов, а также других затрат на ее производство и реализацию.
В соответствии с Положением о составе затрат и учетом отраслевых особенностей министерства (ведомства) межотраслевые государственные объединения, концерны, ассоциации разрабатывают и утверждают отраслевые методические рекомендации по вопросам планирования, учета и калькулирования себестоимости продукции (услуг) для подчиненных им предприятий (организаций). Эти методические рекомендации используются при разработке и внедрении ИС.
Положением о составе затрат по производству и реализации продукции (услуг) установлена единая для всех предприятий и организаций независимо от форм собственности и организационно-правовых форм номенклатура экономических элементов затрат, образующих себестоимость продукции:

- материальные затраты (за вычетом стоимости возвратных отходов);
- затраты на оплату труда;
- отчисления на социальные нужды;
- амортизация основных фондов;
- прочие затраты.

В таблице 3.11 приводится примерная структура затрат на разработку ИС.
Таблица 3.11 – Примерная структура затрат на разработку ИС

Прямые материальные затраты (ПМЗ)	
Канцелярские товары	
Носители информации (флеш – устройства и др.)	
Всего	
Основная заработная плата (ОЗП), фонд оплаты труда (ФОТ)	
Руководитель проекта	
Разработчик(и)	
Всего	

Отчисления на социальные нужды (ОСН) из ФОТ	
Отчисления в фонд социального страхования	2,9 %
Отчисления в пенсионный фонд	22 %
Отчисления в фонд медицинского страхования	5,1%

Страхование от несчастных случаев	0,2%
Всего	30,2%
ОСН = ФОТ *0,302	
Затраты на программное обеспечение (ПО), аппаратное обеспечение	
Затраты на программное обеспечение	
Затраты на аппаратное обеспечение	
Амортизация оборудования (А)	
Стоимость оборудования	
Амортизация %	10%
Амортизация руб. (на период разработки и внедрения ИС)	
Затраты на электроэнергию (Э)	
Количество рабочих дней	
Количество часов в день	8
Стоимость за 1 кВт/ч	руб.
Потребляемая мощность единицы оборудования (компьютера)	кВт
Число используемых единиц оборудования (компьютеров)	ед.
Потребляемый объем электроэнергии	кВт
Всего затраты на электроэнергию (Э)	
Всего затрат по проекту	
Накладные расходы (10%)	
ИТОГО	
Конечная цена проекта (КЦП)	
КЦП = ИТОГО + 18%НДС = руб.	

Ниже приводится методика расчетов данных, приведенных в таблице 3.11.

Основная заработная плата

За весь период разработки ИС будет определяться как сумма произведений среднедневной заработной платы участников проекта на число дней разработки:

$$C_{осн} = C_{дн.пр} * N_{пр} + C_{дн.рук.} * N_{рук},$$

где $C_{осн}$ - основная заработная плата, $C_{дн.пр.}$ - среднедневная заработная плата разработчика(ов), $N_{пр}$ - число дней разработки ИС, $C_{дн.рук.}$ - среднедневная заработная плата руководителя, $N_{рук}$ - число дней работы руководителя.

Средняя месячная зарплата разработчика _____ руб.,

руководителя _____ руб.

В месяце 26 рабочих дней.

Согласно таблице 3.10 руководителем затрачено _____ дня,

разработчиком (ами) _____ дней

Таким образом, основная заработная плата (ОЗП) будет следующая:

ОЗП = _____

Отчисления

Если у организации нет права на применение пониженных тарифов, то она должна начислять взносы (отчисления на социальные взносы) по **основным тарифам.**

С выплат работнику	Размер тарифа по взносам		
	В ПФР	В ФСС на ВНиМ	В ФФОМС
Не превышающих предельную базу	22%	2,9%	5,1%
В части, превышающей предельную базу	10%	-	

Плюс взносы 0,2% - по страхованию от несчастных случаев.

Итого отчисления составляют: 22% + 2,9% + 5,1% + 0,2%=30,2%

Затраты на приобретение программного обеспечения

Программное обеспечение	Сумма в рублях
Всего	

316 – число рабочих дней в текущем году.

Амортизация оборудования (компьютера)

Амортизация оборудования (компьютера - ПК) составляет 10% в год от его балансовой стоимости:

А = _____

Вычислим амортизационные отчисления из расчета количества рабочих дней:

А= _____

Затраты на электроэнергию

Для расчета затрат на электроэнергию (Э) берем число полных рабочих дней по 8 часов в каждом. Вычисления производим согласно тарифу ___ руб. за 1 кВт/ч., потребляемой мощности $M_{об}$ единицы оборудования (компьютера) ____ кВт. и числа единиц оборудования $N_{об}$:

$$Э = (\text{число дней} * 8 * M_{об} * N_{об}) * \text{Тариф} = \_\_\_\_\_$$

Накладные расходы

Накладные расходы – это процент отчислений, установленный в организации (например, 10 %) от суммы всех статей затрат, рассмотренных выше. Статьи учитываемых затрат приводятся в сводной таблице 3.12.

Таблица 3.12 - Статьи затрат

Статьи затрат	Сумма в рублях
Основная заработная плата (ОЗП)	
Отчисления на социальные нужды (ОСН)	
Амортизация персонального компьютера (А)	
Прямые материальные затраты (ПМЗ)	
Затраты на программное обеспечение (ПО), аппаратное обеспечение	
Электроэнергия (Э)	
Всего	
Накладные расходы	
Итого (сумма затрат)	

Цена на разработку проекта (разработка и внедрение ИС) составит:
Конечная цена проекта = сумма затрат + 18% НДС = ___ руб.
Для выводов по эффективности разработанной ИС проводится сравнение показателей существующей и предлагаемой системы по учету оказанных услуг (таблица 3.13).
Таблица 3.13 - Пример сравнения показателей существующей и предлагаемой ИС

Показатели сравнения	Существующая система	Предлагаемая система	Экономия времени в месяц
Импорт входных данных в общую базу данных, в день			
Расчет по оказанным услугам за один календарный день			
Расчет по оказанным услугам за месяц			
Экспорт выходных данных в день			
Ввод, редактирование данных по клиентам и их услугам в систему учета (в день)			
		ИТОГО в месяц	

Приведем расчет эффективности от сокращения времени выполнения работ.

Средняя месячная зарплата сотрудника структурного подразделения компании, занятого в работах указанных в таблице 3.13, составляет _____ руб.

Зарплата сотрудника за 1 час работы:

Месячная зарпл. сотр. / 26 дней / 8 = _____ руб.

Прямой эффект от внедрения новой системы при экономии времени _____ час. в месяц составляет:

Прямой эффект = Зарпл. сотр. за 1 час работы × экономия времени = _____ руб. в месяц

Срок окупаемости проекта на создание и внедрение ИС составляет

Затраты / Прямой эффект = _____ мес.

Методика расчета показателей экономической эффективности на этапе эксплуатации ИС

Стандартная методика расчета показателей экономической эффективности на этапе эксплуатации ИС включает в себя расчет суммы годовой экономии, коэффициента экономической эффективности капитальных вложений и срока окупаемости капитальных вложений.

Сумма годовой экономии от сокращения ручного труда по обработке информации рассчитывается по формуле:

$$S = OC_1 - OC_2$$

S – сумма годовой экономии от сокращения ручного труда по обработке информации, руб.;

OC_1, OC_2 – годовые эксплуатационные затраты при ручной и машинной обработке информации, руб.

$$OC_1 = \frac{\sum_i Z^1{}_i * T^1{}_i}{Q} * (1+\alpha)(1+\beta) * 12$$

$Z^1{}_i$ – месячная основная заработная плата i-го работника, руб.;

$T^1{}_i$ – месячные трудовые затраты i-го работника на решение задачи, человеко-дни;

Q – среднее количество рабочих дней в месяц, дни;

α – коэффициент накладных расходов;

β – коэффициент дополнительной заработной платы (отчисления на социальное страхование, в различные фонды и т.п.).

$$OC_2 = C_1+C_2+C_3$$

C_1 – годовые затраты машинного времени на решение задачи, руб.;

C_2 – годовые затраты на заполнение документов, анализ и корректировку данных (ручные операции), руб.;

C_3 – годовые затраты на обучение персонала, адаптацию и настройку оборудования, руб.

$$C_1 = 12\sum_q S_q * T_q$$

S_q – себестоимость часа работы оборудования q при решении задачи, руб.;

T_q – время работы оборудования q при решении задачи в течение месяца, машино-часы;

C_2 - рассчитывается по аналогичной формуле.

$$C_3 = K_3 * \gamma$$

K_3 – годовые единовременные затраты на обучение персонала, адаптацию, настройку оборудования при решении задачи, руб.

γ - коэффициент настройки оборудования.

$$K_3 = K_{31}+K_{32}+K_{33}$$

K_{31} – годовые единовременные затраты по заработной плате персонала на обучение, адаптацию и настройку оборудования для решения задачи, руб.

$$K_{31} = \frac{\sum_i Z_i^2 * T_i^2}{Q}(1+\alpha)(1+\beta)*12$$

Z^2_i – месячная основная заработная плата работника *i*, руб.;

T^2_i – месячные трудовые затраты работника *i* на обучение, настройку оборудования и т.п., человеко-дни;

K_{32} – годовые единовременные затраты машинного времени

$$K_{32} = 12 * \sum_q S_q * T_q^2$$

T^2_q – время работы оборудования q на обучение персонала, адаптацию и настройку оборудования, машино-часы.

$К_{33}$ – прочие единовременные расходы, руб.:

$$K_{33} = (K_{31} + K_{32}) * h$$

h – коэффициент прочих расходов, к прочим расходам относятся: расходы на приобретение машинных носителей, бумаги, краски и т.п.

К – единовременные затраты на решение задачи, руб.

$$К = К_1 + К_2 + К_3$$

$К_1$ – единовременные затраты на проектирование и разработку ИС, руб.:

$$К_1 = \frac{\sum_i Z^2{}_i * T^2{}_i}{Q} * (1+\alpha)(1+\beta) * n$$

T^2_i – месячные трудовые затраты специалиста на проектирование ИС, человеко-часы;

n – длительность проектирования и разработки ИС;

$К_2$ – единовременные затраты, связанные с использованием различных видов оборудования, руб.

$$К_2 = \frac{BV(1-(t*r)/100)(1+\alpha)T}{F},$$

Где BV – балансовая стоимость комплекта техники или ПЭВМ, руб.;

t – длительность эксплуатации ПЭВМ до начала решения задачи, годы;

r – годовая норма на реновацию оборудования (около 10%);

T - время работы оборудования при решении задачи в течении месяца, машино-часы;

α – коэффициент, определяющий стоимость вспомогательного оборудования;

F – планируемый годовой фонд времени работы ПЭВМ (оборудования);

$$F = t_c * T_c,$$

t_c – среднесуточная фактическая загрузка ПЭВМ (оборудования), часы;

T_c – среднее количество дней работы ПЭВМ (оборудования) в году.

Коэффициент экономической эффективности E_r рассчитывается по формуле:

$$E_r = S/К.$$

Если $E_r \geq E_n$, то проект можно считать эффективным.

E_n – нормативный коэффициент эффективности капитальных вложений для вычислительной техники, его значение определяет нижнюю границу годовой экономии, которую можно получить на один рубль капитальных затрат.

Т - срок окупаемости затрат на проект:

$$T=K/S.$$

Таким образом, для более точной оценки стоимости разработки и внедрения ИС необходимо выполнить следующие задачи:

1) Формирование требований к ИС - анализ функциональных и нефункциональных требований, требований к внешним интерфейсам ИС, аппаратному обеспечению и обеспечению информационной безопасности, требования к средствам и методам проектирования и разработки;

2) определение структурных единиц и стоимости поставок программного и аппаратного обеспечения - определение необходимого программного и аппаратного обеспечения для функционирования предлагаемой ИС и оценка стоимости поставок, аренды аппаратного и программного обеспечения;

3) оценка размера программной части ИС - определение размера программной части ИС;

4) оценка стоимости разработки программного обеспечения ИС – расчет показателей проекта (трудозатрат, длительность, трудовые ресурсы) с учетом факторов сложности проекта;

5) оценка стоимости ИС на этапе внедрения и эксплуатации ИС – сравнительный анализ показателей существующей и предлагаемой систем.