ИННОВАЦИОННАЯ ОБРАЗОВАТЕЛЬНАЯ ПРОГРАММА

Проект 2: индивидуальная траектория обучения и качество образования

Цель: ориентированное на требования рынка образовательных услуг улучшение качества подготовки и переподготовки специалистов

Федеральное агентство по образованию

Государственное образовательное учреждение
высшего профессионального образования

Владимирский государственный университет

Р.И. Макаров, Е.Р. Хорошева

МЕТОДОЛОГИЯ ПРОЕКТИРОВАНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

Учебное пособие

Владимир 2008

УДК 681.518.001.2(075.8)
ББК 32.965я73
М15

Рецензенты:
Доктор технических наук, профессор зав. кафедрой вычислительной техники Костромского государственного технологического университета
В.Н. Шведенко

Доктор технических наук, профессор кафедры менеджмента Владимирского государственного гуманитарного университета
Н.Г. Наянзин

Печатается по решению редакционного совета
Владимирского государственного университета

Макаров, Р.И.
М15 Методология проектирования информационных систем: учеб. пособие / Р. И. Макаров, Е. Р. Хорошева; Владим. гос. ун-т. – Владимир: Изд-во Владим. гос. ун-та, 2008. – 334 с.
ISBN 978-5-89368-817-7.

Широкое использование информационных технологий во всей сфере деятельности человека делает актуальной задачу разработки и проектирования информационных систем. Эффективность использования информационных систем во многом зависит от уровня разработки и проектной проработки, а также квалификации обслуживающего персонала.

Данный курс знакомит будущих магистров с методологией разработки и проектирования информационных систем. В курсе рассматриваются особенности информационных систем как объекты проектирования, технология проектирования, вопросы управления процессом проектирования, отладки и обслуживания информационных систем. Особое внимание уделяется применению компьютерных автоматизированных технологий проектирования.

Учебное пособие составлено на материале лекций, прочитанном магистрантам по специализированной подготовке по направлению 230200 – «Информационные системы» по программе 230218 - «Анализ и синтез информационных систем».

Табл. 15. Ил.97. Библиогр.: 14 назв.

УДК 681.518.001.2(075.8)
ББК 32.965я73

ISBN 978-5-89368-817-7

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Данное учебное пособие написано на основе прочитанных лекций по дисциплине «Проектирование информационных систем» во Владимирском государственном университете для студентов, специализирующихся по направлению «Информационные системы». Появление учебного пособия «Методология проектирования информационных систем» обусловлено необходимостью специализированной подготовки магистрантов по направлению 230200 - Информационные системы по программе 230218 - Анализ и синтез информационных систем.

Широкое использование информационных технологий во всей сфере деятельности человека делает актуальной задачу проектирования информационных систем. Эффективность использования информационных систем во многом зависит от уровня разработки, проектной проработки и квалификации обслуживающего персонала. Широта охвата и интегрированность на основе функций управления, способность оперативно подготавливать управленческие решения и адаптируемость к изменениям внешней среды и информационным потребностям являются отличительными особенностями современных информационных систем.

Усложнение архитектуры современных информационных систем требует использования эффективных технологий проектирования, обеспечивающих ускорение создания, внедрения и развития проектов информа-

ционных систем, повышение их функциональной и адаптивной надежности. Методологическую основу проектирования информационных систем составляет системный подход.

В данном пособии рассматриваются принципы, методы и приемы, способные обеспечить надлежащий анализ и надлежащее проектирование. В пособии приведено обобщение достижений отечественной и зарубежной науки и практики в области разработки и использования технологий проектирования информационных систем, с общей методологической точки зрения рассматриваются вопросы выбора и применения методов и средств проектирования информационных систем в рамках различных технологий канонического и системного проектирования.

В каноническом проектировании информационных систем рассматриваются вопросы разработки отдельных подсистем: информационного, технического, программного, организационного обеспечения с использованием методов и средств оригинального проектирования.

Системное проектирование рассматривается в аспектах эффективного реинжиниринга деятельности объектов информатизации. В качестве методов и средств системного проектирования рассматриваются функционально-ориентированные и объектно-ориентированные *CASE*- и *RAD*-технологии автоматизированного проектирования, параметрически-ориентированные и модельно-ориентированные компонентные технологии типового проектирования.

Вопросы организации и управления процессом проектирования информационных систем рассмотрены в аспектах обоснования организационной структуры проекта и применения методов планирования, организации и контроля за проведением проектных работ

В главе 1 описывается стратегия развития информационных технологий на предприятии. Рассматривается архитектура предприятия и место в ней информационной системы, основные пути развития современных ИС организационного управления предприятием. Приводятся методы оценки эффективности информационных систем.

Глава 2 посвящена знакомству с методологиями и технологиями разработки информационных систем. Рассматривается *RAD* – методология, отечественные, зарубежные и внутрифирменные стандарты и методики разработки информационных систем, профили открытых информационных систем.

В главе 3 рассматриваются структурные характеристики систем управления. Описываются методики синтеза организационной структуры и функциональной структуры информационных систем. Дается методика синтеза структуры ИС с учетом затрат на обмен информацией между подсистемами.

Глава 4. знакомит с методологией расчета потребностей программно-технического обеспечения разрабатываемых интегрированных ИС на базе системы *R*/3. Определяется оптимальная конфигурации программно-технического комплекса системы *R/*3, приводится ее математическая модель.

В главе 5 описывается информационный процесс обработки данных в ИС. Рассматриваются вопросы организация вычислительных процессов и обслуживания вычислительных задач, методы преобразования данных, управление ресурсами вычислительных систем. Приводится методика оценки точности процесса обработки информации. Рассматриваются вопросы оценки надежности и контроля информации в информационных системах. Описывается методика расчетного определения надежности ИС, распределения надежности и резервирования устройств.

Глава 6 посвящена методологии и технологии разработки программного обеспечения ИС. Рассматриваются общие принципы проектирования, в том числе объектно-ориентированные методы анализа и проектирования программного обеспечения. Описывается моделирование бизнес процессов и спецификаций требований к программному обеспечению с использованием языка *UML*. Описывается унифицированный язык моделирования *UML*. Приводятся методы оценки трудоемкости создания программного обеспечения.

В главе 7 обсуждаются вопросы организации работ по проектированию, методы планирования и управления проектами и ресурсами. Описывается технология применения метода СПУ для разработки проекта информационной системы. Приводятся рекомендации по выбору системы для управления проектами.

Вопросы отладки, испытания и обслуживания информационных систем рассмотрены в главе 8. Описываются методы проверки работоспособности, правильности функционирования и организации эксплуатации информационных систем. Приводится методика обеспечения информационных систем запасными устройствами. Рассматриваются вопросы организации обучения, тренировки пользователей информационных систем.

ГЛАВА 1

СТРАТЕГИЯ РАЗВИТИЯ ИТ ПРЕДПРИЯТИЯ

1.1. Стратегия развития информационных технологий на предприятии

Информационные технологии (ИТ) используются на предприятии, как правило, в совокупности и образуют при этом систему.

Согласно ГОСТ 34.003-90 «Информационная технология. Комплекс стандартов и руководящих документов на автоматизированные системы. Термины и определения», системой называется совокупность технических средств, людей и бизнес процессов, совместное использование которых способствует достижению определенных целей.

В литературе встречаются два определения понятия бизнес процесса, данные Хаммером и Чампи, и Шеером. Оба определения не противоречат друг другу. Приведем определение Шеера. Бизнес процесс - описание порядка работ, направленных на достижение определенной бизнес-цели и обладающих следующими признаками:

- интегрированное описание функций, документов, опосредующих эти работы, и организационных подразделений;
- иерархический характер описания.

Процесс характеризуется тем, что он может быть нарисован в виде схемы, его производительность может быть измерена, он может быть улучшен путем выявления соотношений между его компонентами и последующим изменением этих компонентов и их взаимосвязей.

На сегодняшний день существующие на предприятии ИТ способны помочь в решении следующих задач:

- достижение цели организации;

- управление рисками;

- изменение деятельности.

Известно, что руководство предприятия и пользователи воспринимают ценность ИТ различным образом. Чем выше уровень в иерархии руководства, тем больше ослабевает внимание к ИТ и растет внимание к деятельности предприятия.

ИТ требуется создавать таким образом, чтобы они поддерживали все потребности предприятия с позиции улучшения временных и функциональных показателей, а инфраструктура ИТ обеспечивала бы рост всего предприятия за счет улучшения уровня удовлетворенности клиентов.

Для реализации этих целей необходимо:

- обладать своевременной, пригодной к использованию и надежной информацией;

- использовать производительные и эффективные методы и средства (измерение характеристик, управление знаниями и т.д.);

- предусматривать возможность интеграции применяемых технологий.

Один из способов достижения поставленных целей заключается в создании и реализации ИТ-стратегии предприятия. ИТ-стратегия состоит из рационального использования имеющихся в организации ИТ, которые отвечают и поддерживают миссию предприятия.

Под миссией предприятия понимается основная общая цель или задача предприятия, четко выраженная причина его существования. Миссия объединяет задачу и конкретную причину, определяющую существование данного конкретного предприятия. Она позволяет потребителю отделить одно предприятие от другого, занимающегося аналогичной деятельностью.

Пример миссии-предназначения:

«Деятельность направлена на сохранение и развитие н/т потенциала отрасли, поддержание высокого уровня разработок, создание новых рабочих мест и культуры производства, сохраняющей и защищающей окружающую среду» (опытно-конструкторское бюро).

Миссия способствует формированию имиджа предприятия во внешнем мире. Пример формулировки миссии:

«Мы стремимся быть лидерами в инновациях, развитии и производстве самых наукоемких технологий» (*IBM*).

Процесс создания ИТ-стратегии начинается с установления целей для имеющихся на предприятии ИТ и определения начальных направле-

ний развития. Затем выясняются измеряемые показатели деятельности, значение которых сравнивается с желаемыми значениями. После установления целей и перечня измеряемых показателей основной задачей для правления предприятия («законодательная» власть на предприятии, принадлежит представителям органа «владельца предприятия») становится достижение этих целей, а для дирекции (исполнительное управление, высший менеджмент) – проведение изменений, способствующих их реализации.

Основными целями являются:

- соответствие развития ИТ деятельности предприятия;
- создание благоприятных условий основной деятельности для получения максимальных выгод;
- рациональное использование ИТ-ресурсов;
- надлежащее управление ИТ-рисками.

К числу основных направлений по изменению функций и состава ИТ можно отнести:

- повышение степени автоматизации (чтобы сделать деятельность более эффективной);
- уменьшение затрат (чтобы сделать предприятие более результативным);
- управление рисками (обеспечение безопасности, надежности и соответствия).

Для оценки эффективности ИТ служат разные показатели, которые зависят от зрелости предприятия в области использования ИТ [1].

В случае, когда предприятие находится на первом уровне зрелости, где ИТ-подразделение обеспечивает работу оборудования и программ, показателем, косвенно отражающим эффективность применения ИТ, является совокупная стоимость владения (*TCO – total cost of ownership*). Этот показатель характеризует достаточность уровня затрат на ИТ при сравнении предприятия с аналогичными структурами (по размеру и области бизнеса). Показатель *TCO* не связан прямо с целями использования ИТ на предприятии. Как правило рассматриваются затраты за год, отнесенные к рабочему месту.

Предприятия со вторым уровнем зрелости использования ИТ характеризуется тем, что в них ИТ-подразделение обеспечивают сервисы. На предприятиях с уровнем зрелости использования ИТ выше второго, показателем эффективности использования ИТ является отношение себестои-

мости сервиса к экономически оцениваемым преимуществам, которые получат предприятия за счет сервиса. Сервис непосредственно связан с бизнесом в отличие от показателя *TCO*.

В современных условиях многие проблемы бизнеса могут быть решены с использованием ИТ. Вместе с тем на предприятии существует ряд проблем, связанных с отсутствием единой корпоративной политики в области ИТ и стратегии создания корпоративной информационно-управляющей системы (КИУС) предприятия в целом.

Под корпоративной стратегией понимается долгосрочное направление развития предприятия, следование которому приведет к достижению стратегических целей. При разработке стратегии формулируют общие направления развития предприятия, касающиеся производимой продукции и каналов ее продвижения. Стратегия должна обеспечить концентрацию усилий в области с устойчивыми конкурентными преимуществами.

Разработка корпоративной стратегии позволяет перейти от управления предприятием, зависящим от воздействия случайно возникающих внешних и внутренних факторов, к планомерной деятельности по достижению конкретных результатов с возможностью оценки их достижимости по определенным критериям и применения адекватных управляющих воздействий. При этом подразумевается необходимость разработки бизнес-плана и создания системы управления предприятием. Корпоративная культура направлена на формирование общих для всех сотрудников предприятия целей, ценностей и принципов поведения. Она должна способствовать развитию предприятия и достижению им своих бизнес-целей.

В последние годы в нашей стране возник повышенный интерес к разработке бизнес-планов предприятий. Бизнес-план представляет собой всестороннее описание бизнеса и среды, в которой он действует, а также части системы управления, необходимой для достижения поставленных целей. Бизнес-план описывает все основные аспекты развития предприятия, анализирует проблемы, с которыми оно может столкнуться, а также определяет способы их решения. Существует много разновидностей бизнес-планов: для себя, для получения кредита, для привлечения средств сторонних инвесторов, для заключения крупного контракта, для реорганизации дела и оптимизации операций и др.

Под ИТ-стратегией понимают формализованную систему подходов, принципов и методов, на основе которых будут развиваться все компоненты КИУС.

При создании ИТ-стратегии целесообразно пользоваться показателями деятельности (не путать с показателями производительности), которые позволяют понять, как ИТ-стратегия улучшает деятельность предприятия. Этими показателями могут быть:

- улучшение рентабельности ИТ-процессов;
- продолжительность системных простоев;
- пропускная способность информационной системы и время ответа на запросы клиентов (каждый, кто пользуется результатами процессов, связанных с ИТ на предприятии);
- снижение затрат на менеджмент;
- доля вклада ИТ в продукты и услуги;
- рост числа стандартизированных процессов предприятия;
- рост удовлетворения ожиданий заинтересованных лиц; выполнение требований по бюджету, по временным характеристикам и по графикам;
- соответствие законам;
- прозрачность в определении риска и соответствие согласованному профилю рисков в организации;
- создание новых каналов предоставления услуг клиентам;
- примеры деятельности, демонстрирующие высокий возврат инвестиций.

Целью проекта по разработке ИТ-стратегии является организация интегрированного корпоративного процесса по развитию ИТ для обеспечения их соответствия основным целям и направлениям развития бизнеса предприятия. Достижение указанной цели позволит обеспечить:

- совершенствование системы управления;
- целенаправленное планирование и внедрение ИТ;
- ориентацию ИТ для решения проблем бизнеса;
- создание единого информационного пространства предприятия;
- снижение совокупной стоимости владения ИТ;
- сокращение сроков внедрения новых ИТ, получение быстрых и тиражируемых результатов;
- повышение эффективности использования ИТ и отдачи от инвестиций в информатизацию;
- возможность быстро и экономично решать информационную инфраструктуру в будущем;
- повышение конкурентоспособности и акционерной стоимости.

Документ, содержащий стратегию развития ИТ, включает следующие основные компоненты:

- Цель и назначение стратегии.

- Роль ИТ в деятельности предприятия. Определяется роль ИТ в развитии бизнеса, организации управления. Формулируются задачи ИТ, поддерживающие решение бизнес-проблем.

- Краткая характеристика состояния информатизации. Анализируются результаты аудита существующих ИС, их соответствие бизнес-процессам, выявляются недостатки. Квалифицируются пользователи, оценивается степень их удовлетворенности. Оценивается уровень квалификации персонала ИТ-службы. Приводятся экономические параметры текущего состояния информатизации.

- Анализ имеющихся инициатив и проблемных областей. Приводятся планы развития и предлагаемые проекты развития с учетом информационных потребностей, стратегии развития бизнеса и организации управления предприятия.

- Оценка готовности к изменениям. Анализируется готовность руководства и структурных подразделений к внедрению новых или модернизацию существующих ИС. Оценивается необходимость реорганизации системы управления, бизнес-процессов, наличие необходимых ресурсов для выполнения перечисленных работ.

- Основные направления развития информатизации. Определяется будущее состояние ИТ предприятия, направления развития информатизации с учетом корпоративной стратегии. Указывается приоритетность направлений с учетом общей стратегии развития бизнеса и организации управления.

- Портфель инвестиционных проектов по развитию информатизации. Формируется перечень проектов по основным направлениям развития информатизации, выбираются основные системные решения по их реализации. Формируется план развития информатизации на требуемый период.

- Ожидаемые результаты. Приводятся ожидаемые результаты от реализации проектов. Прогнозируется их влияние на основную деятельность предприятия.

- Оценка необходимых ресурсов. Оцениваются сроки и стоимость реализации выбранных проектов в зависимости от организации разработки и внедрения (внутренними силами, с привлечением внешних исполнителей, путем выбора генерального системного интегратора в качестве стратегического партнера и т.п.).

- Требования к организации работ по развитию информатизации. Предлагается организационная модель развития ИТ. Определяется роль руководства и структурных подразделений, вовлекаемых в процесс развития информатизации. Определяются принципы управления процессом развития и контроля за соответствием получаемых результатов ожидаемым.

- Стратегия переходного периода. Приводится анализ рисков, связанных с реализацией проектов. Определяются методы принятия управляющих решений.

С позиции корпоративной философии создание стратегии позволяет обеспечить [1]:

1) понимание того, что ИТ должны способствовать совершенствованию управленческого процесса;

2) осознать факт, что развитие ИТ требует постоянного внимания со стороны высшего руководства;

3) создание культуры управления с использованием ИТ;

4) преодоление психических барьеров персонала, выработку новой мотивации труда, необходимого настроя на перемены, понимания и поддержки происходящего;

5) воспитание собственной группы специалистов, способной квалифицированно решать организационные, технические и прочие вопросы реформирования предприятия и проведения автоматизации.

1.2. Архитектура предприятия и информационной системы

1. Определение архитектуры предприятия

Архитектурой предприятия называются информационные составляющие, которые определяют структуру бизнеса, информацию, необходимую для ведения бизнеса, технологию, поддерживающую деловые операции, процессы преобразования и развития, необходимые для реализации новых технологий с учетом изменения бизнес-процесов.

Ответственными за интеграцию предприятия являются два типа архитектур – это, собственно, архитектура предприятия, которая отвечает за организацию развертывания и выполнения такого проекта, как интеграция предприятия или иной программы и системная архитектура, отвечающая за конструирование некоторой системы, например компьютерной системы контроля и управления, как части интегрированной системы предприятия.

Цель введения понятия «архитектура предприятия» состоит в том, чтобы информировать, управлять и осуществлять решения, связанные с инвестициями в ИТ.

При описании архитектуры используется понятие «слой», представляющее способ структуризации информации, содержащейся в понятии архитектуры и указывающей, какой аспект деятельности предприятия отражается. Существует несколько моделей архитектуры предприятия, различающиеся количеством слоев, деятельностью и используемой терминологией.

На рисунке 1.1 показана модель архитектуры предприятия, предложенная национальным институтом стандартов и технологий (*NIST*).

Использование понятия архитектура предприятия позволяет устанавливать связь между бизнесом предприятия и параметрами ИС – функциями системы и интероперабельностью*) данных.

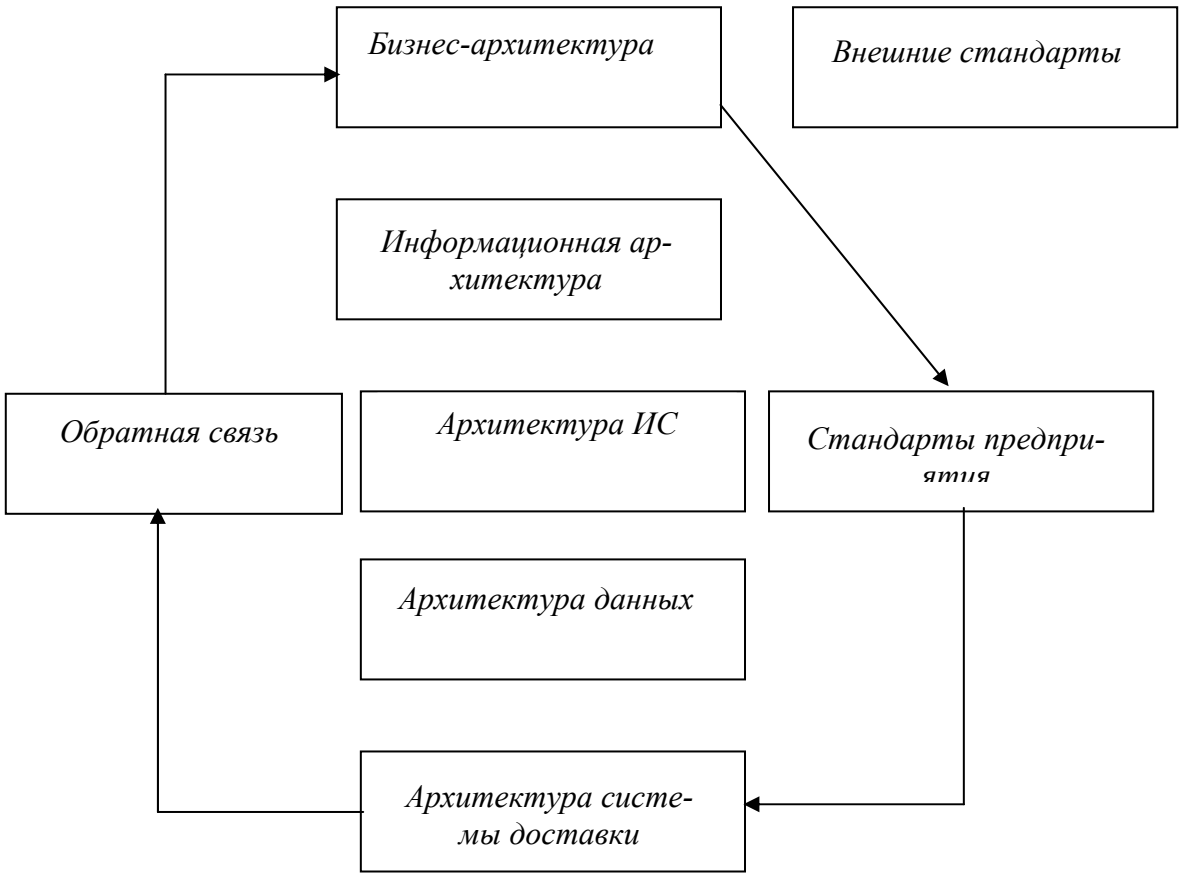


Рис. 1.1. Модель архитектуры предприятия

Основными предпосылками к использованию понятия архитектуры являются стандарты и унификация методов сбора данных.

*) интероперабельность данных – возможность использования данных приложениями

Основное требование в том, чтобы любая информация, создаваемая в ИС, была независима от программного обеспечения разработки. Основное внимание должно быть сосредоточено на использовании Internet и *Web*-стандартов, языка *XML*, порталов, *Web*-сервисов, а также увеличении использования услуг провайдеров приложений.

Основным принципом руководителей подразделений ИС должно быть устранение использования программного обеспечения собственного изготовления. Это требование необходимо включать в текущие и будущие планы.

Стандартизация данных устраняет избыточность и гарантирует их согласованность.

Архитектура предприятия является компонентом стратегического планирования, гарантирующим на стратегическом уровне согласованность развития ИТ и стратегического развития предприятия. Она содействует совместному использованию информации между разными предприятиями, а также в государственных учреждениях, обеспечивая межведомственное взаимодействие.

Архитектура предприятия содержит схему о состоянии ИТ в разные периоды времени. Имея такую информацию, специалисты получают возможность быстрее реагировать на изменяющуюся ситуацию, минимизировать количество шагов при внедрении изменений, упростить процессы переосмысливания потребностей и анализа принимаемых решений.

Сокращение затрат при использовании стандартизированной архитектуры предприятия реализуется благодаря использованию типовых архитектур и за счет уменьшения необходимости начинать разработку с нуля тех решений, которые уже известны.

Укажем мотивы, которые стимулируют разработку и использование архитектуры предприятия:

1) приведение предприятия в соответствие с намерениями, чтобы преобразованное предприятие соответствовало исходным требованиям;

2) интеграция. Бизнес-процедуры и правила являются непротиворечивыми, данные – защищенными, интерфейсы и потоки информации – стандартизированными, коммуникации и взаимодействия (интероперабельность) поддерживаются в пределах всего предприятия и государства;

3) облегчение управления изменениями в любых аспектах предприятия (конвергенция, связь между подразделениями предприятия, наглядное представление предприятия, ориентация на стратегическое ис-

пользование современных технологий управления большими потоками информации);

4) улучшение качественных показателей совместно используемой информации;

5) совершенствование планирования капвложений и инвестиционного управления;

6) повышение качества и гибкости использования приложений без увеличения затрат;

7) достижение экономии на основе совместного использования услуг в масштабах предприятия;

8) упрощение интеграции наследуемых, перемещаемых и новых систем.

2. Описание слоев архитектуры

Архитектура предприятия обычно имеет следующие слои:

1) бизнес-слой;
2) архитектура данных;
3) интеграция физических данных;
4) концептуальная модель/модель процессов;
5) архитектура приложений;
6) технологическая архитектура;
7) техническая модель.

Бизнес-слой идентифицирует функции, процессы, организацию и информационные потоки, обеспечивающие миссию организации. Он представляет исходную точку зрения для анализа данных, используемых приложений и их возможностей, а также реализацию технологий, необходимых для поддержания повторного использования компонентов и выполнения стандартов. В этом слое учитываются все потенциально возможные методы доступа к информации (персональные контакты, электронные средства, бумажные документы, сервис-провайдеры).

Архитектура данных определяет главные типы данных, которые необходимы для поддержки деятельности предприятия. Содержит словарь общих данных и определений, который особо важен для использования. В архитектуру данных включаются не только операционные, но и аналитические данные, содержание, представленное на *Web*-сайтах, присутствует модель интероперабельности (поддержки взаимодействия).

Для поддержки интероперабельности данных необходимо:

- использовать стандартные синтаксисы;

- применять промышленные словари;

- избегать создания всеобщих схем, применять их сегментирование и структурирование с учетом последующего расширения и распространения в масштабе всего проекта;

- обеспечить совпадение семантики разделяемых (совместно используемых) элементов данных;

- использовать стандартные интерфейсы для услуг документирования.

Модель интероперабельности описывает прикладные компоненты, поддерживающие концептуальную модель/модель процессов и способ взаимодействия в пределах конкретных решений. Она поддерживает интероперабельность для пользователей, данных и приложений.

Архитектура безопасности предусматривается для каждого компонента модели интероперабельности, начиная от услуг электронной аутентификации и заканчивая управлением доступом через запрашивающие приложения и транзакционные услуги. Секретность, как безопасность, отражается во всех компонентах модели интероперабельности.

Архитектура приложений определяет используемые приложения и средства, предназначенные для эффективного управления данными и информацией, для поддержки деятельности и достижения намеченных целей.

Технологическая архитектура определяет доступное компьютерное оборудование, программное обеспечение, а также физическое расположение с позиции поддержания приложений, данных, функций.

Техническая модель показывает, как взаимодействуют те или иные компоненты.

3. *Архитектура информационной системы*

Описание архитектуры ИС-системы представляет собой детальное руководство, которое определяет основные, стандартные или типовые элементы ИТ-систем, их взаимосвязь, а также процессы управления ИТ-системой.

ИТ-архитектура подразделяется на набор областей верхнего уровня (доменов), описывающих отдельные компоненты ИТ-систем. В состав списка доменов входят следующие области:

1) управление приложениями;
2) управление данными;
3) управление информацией;
4) управление пользователями и их доступом;

5) сети и коммуникации;
6) платформы;
7) управление системами;
8) информационная безопасность и др.

Домены в свою очередь включают несколько функциональных областей. Так, в домен «Управление системами» входят следующие области:

1) управление активами (*Asset management*);
2) управление изменениями (*Change management*);
3) управление событиями (*Event management*);
4) поддержка пользователей (*Help Desk*);
5) обеспечение непрерывности бизнеса (*Business continuity*) и др.

Для каждой области определяются возможные технологии. Так, для домена управления данными могут быть реляционные СУБД, почтовые базы, файловые каталоги, продукты и версии продуктов. Для каждой области, технологии и продукта могут устанавливаться «требования соответствия», определяющие необходимость соблюдения тех или иных международных рекомендаций (*RFC*), стандартов, российских законодательных актов, например по применению сертифицированных средств ЭЦП, внутренних инструкций и т.п.

Все описание архитектуры представляется в виде гипертекстовой базы данных, что позволяет эффективно организовать процесс управления жизненным циклом отдельных документов, а также эффективно разграничить права доступа к некоторым разделам при сохранении целостности и единства описания.

4. *Метод Захмана*

Метод Захмана позволяет связывать характеристики ИС с бизнес-задачами предприятия. Суть метода сводится к формализованному представлению модели предприятия в виде матрицы (Таблица 1.1).

В строках матрицы отображаются различные категории специалистов, связанные с деятельностью предприятия, а в столбцах – основные аспекты производственной деятельности.

На рисунке 1.2 изображена диаграмма, иллюстрирующая несколько главных технологий моделирования, а также места пересечения при формировании матрицы Захмана.

Таблица 1.1. Формализованное представление модели предприятия по методу Захмана

От целей бизнеса к проекту		Объекты Что?	Действия Как?	Дислокация Где?	Люди Кто?	Время Когда?	Мотивы Почему?	
	Планировщик							Сфера действия
	Владелец предприятия							Модель предприятия
	Проектировщик							Системная модель
	Разработчик							Техническая модель
	Субподрядчик							Компоненты архитекту-ры
		Данные	Функции	Сеть	Организация	Расписание	Стратегия	
	Элементы архитектуры предприятия							

Каждая из этих моделей должна соответствовать контексту общего направления бизнеса, которая определяется его задачами, приоритетами и критическими для успеха факторами.

Использование моделей (таблица 1.2) на разных стадиях развития системы показано в таблице 1.3.

В 1987г. Джон Захман опубликовал схему развития архитектуры ИС, которая создает контекст для описания различных представлений архитектуры разрабатываемой системы. Эти представления соответствуют тому, как видят систему ее заказчик, проектировщик и разработчик в разрезе данных, функций и сетевой структуры.

Рис. 1.2. Типы моделирования

Таблица 1.2 Модель развития системы

Модели	Данные	Функции	Работы
Цели, масштаб проекта	Список того, что важно для бизнеса	Список процессов бизнеса	Список областей бизнеса
Модель бизнеса	Диаграммы «сущность-связь»	Диаграммы потоков данных	Связь работ
Модель информационной системы	Модель данных	Иерархия функций	Модель архитектуры системы
Технологическая модель	Проектирование данных	Модель организационных единиц	Оборудование, программное обеспечение
Детальное представление	Полное описание модели данных	Описание программ	Схемы адресации и используемых протоколов
Функционирование системы	Данные	Функции	Связи

В схеме Захмана (таблица 1.1) строке соответствует точка зрения участника проекта по созданию системы. Аспекты представлены колонками таблицы 1.2.

Архитектурное представление – это ячейки таблицы 1.2. Существует некоторое множество архитектур представления ИС с точки зрения заинтересованных лиц.

Заказчик видит систему с точки зрения общих стратегических и тактических аспектов. Архитектурное представление заказчика приведено в двух верхних строках таблицы 1.1.

Представление проектировщика – это проект системы, обеспечивающей удовлетворение требований представителя заказчика.

Таблица 1.3 Использование методов моделирования

Методы	Уровень бизнеса	Системный уровень	Программный/процедурный уровень
Функциональная иерархия	о	о	о
Анализ состояний	н	о	н
Диаграммы потоков данных	н	о	н
Событийное моделирование	н	о	о
Функциональная логика	о	о	н

Примечание: О - обязательное использование; Н - необязательное, возможное использование.

Представления проектировщика и заказчика остаются не зависимыми от технологий, которые будут использованы при реализации системы.

Структурный анализ, информационное моделирование и отдельные виды прототипирования являются теми методами, которые могут быть использованы для формирования архитектурного представления проектировщика. Точке зрения проектировщика соответствует третья строка (таблицы 1.1).

Физическое воплощение логических требований зависит от характеристик аппаратно-программной базы, выбранной для реализации системы. Технология ограничивает решение задач, а не их условия.

Три аспекта, рассмотренных в схеме, приводят к разным архитектурным представлениям каждой из точек зрения. Аспекты соответствуют вопросам «Что?», «Как?», «Где?», относящимся к информационной системе. Каждому аспекту соответствуют разные методы формирования представления.

Для ИС вопрос «Что?» относится к сущностям данных и их связям.

На вопрос «Как?» - описывают, как работают отдельные части ИС. Функции обычно определяются входами, процессами, выходами. Особое внимание уделяется взаимодействию частей при выполнении общей задачи.

Колонка сетевой структуры соответствует вопросу «Где?». Архитектурные представления в этой колонке описывают местоположение элементов системы и механизмы их взаимодействия.

Схема Захмана была разработана достаточно давно, поэтому надо помнить, что сегодняшнее ее применение с новыми стандартами может вызвать затруднения.

1.3. Основные пути развития современных ИС организационного управления предприятием

1. *Основные предпосылки развития интегрированных информационных систем организационного управления*

Быстрое развитие вычислительных средств привело к тому, что при создании современных интегрированных информационных систем (ИИС) появилась возможность реализации систем принятия решений в режиме реального времени, позволяющей быстро реагировать на изменения обстоятельств.

В настоящее время для обеспечения конкурентоспособности предприятия больше не могут полагаться на повышение эффективности своей деятельности, основываясь только на внутренние факторы. Они должны отслеживать множество внешних источников информации и быть готовыми быстро реагировать по всей логистической цепочке деятельности предприятия.

Логистика (от греческого *logislike* - искусство вычислять, рассуждать) - наука о планировании, контроле и управлении транспортированием,

складированием и др. материальными и нематериальными операциями, совершаемыми в процессе доведения сырья и материалов до производственного предприятия, внутризаводской переработки сырья, материалов и полуфабрикатов, доведение готовой продукции до потребителя в соответствии с интересами и требованиями последнего, а также передачи, хранения и обработки соответствующей информации.

Для достижения этой цели руководству компании необходимо решить такие сложные задачи, как достижение высокого качества хозяйственной деятельности путем полной автоматизации и оптимизации; расширение предприятия до охвата всех звеньев логистической цепочки; интеграция систем управления предприятием с системами клиентов, поставщиков и партнеров для создания общей информационной базы; инвестиции в переобучение и переквалификацию сотрудников для получения практических знаний, необходимых для оптимизации хозяйственных процессов.

Необходимо, чтобы в ИС была обеспечена интеграция между такими ключевыми видами деятельности, как планирование, оптимизация, выполнение, измерение результатов деятельности. Эти операции должны проходить по всем хозяйственным процессам (рис.1.3).

Решение поставленных задач предполагает проведение ряда организационно-технических мероприятий в таких направлениях, как проектирование, планирование, управление предприятием, управление заказами, нахождение источников снабжения, производство, поставка и сервис, складирование и транспортировка, а также внедрение бюджетирования доходов и расходов, персонификация ответственности за расходы и затраты, пересмотр и ужесточение норм затрат, организация контроля энергоснабжения и др. Эти мероприятия влияют на выполнение бизнес-целей по снижению производственных затрат.

Бюджет (фр. *Budget*) - роспись денежных доходов и расходов на определенный период, утвержденная в определенном порядке. Бюджет является основным инструментом проверки сбалансированности, соответствия прихода и расхода экономических ресурсов.

Реструктуризация социально-бытовой сферы позволить снизить расходы на непроизводственную сферу.

Внедрение стандартов ИСО 9000 обеспечит высокое глобальное качество продукции. ИСО 9000 – стандарт на качество проектирования, разработки, изготовления и послепродажного обслуживания, определяет базовый набор мероприятий по контролю качества и представляет собой

схему функционирования бизнес-процессов предприятия, обеспечивающую высокое качество ее работы. Схема покрывает все этапы деятельности предприятия, включая закупку сырья и материалов, проектирование, обслуживание клиентов, обучение персонала т.п.

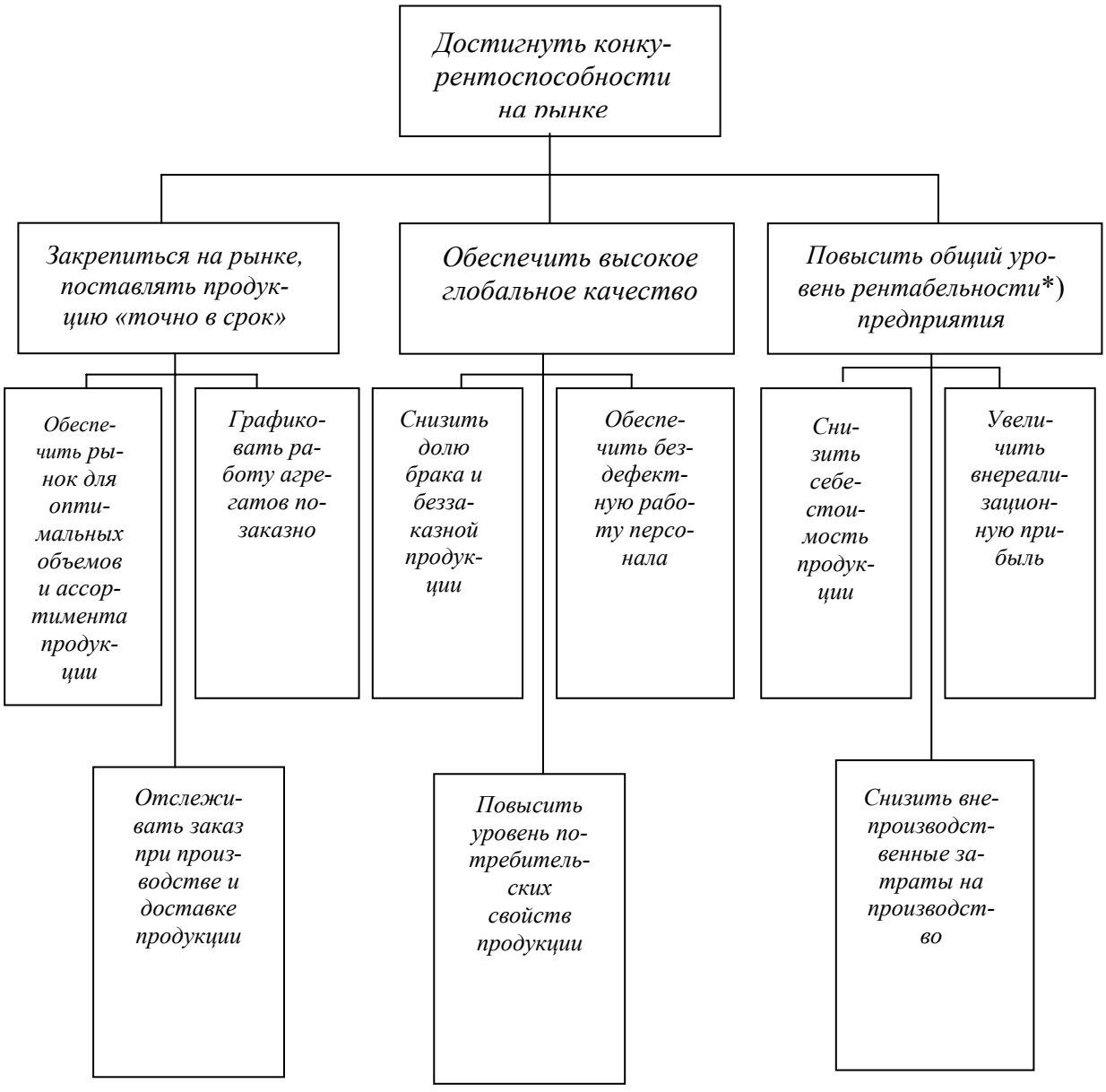


Рис. 1.3. Бизнес-цели предприятия

*) рентабельность (нем. *Rentable* – доходный) – один из стоимостных качественных показателей эффективности производства на предприятии, характеризующий уровень отдачи затрат и степень использования средств в процессе производства и реализации продукции.

Внедрение систем управления предприятием, сертифицированных по стандартам *MRP* (*Manufacturing Resource Planning*) и *ERP* (*Enterprise Resource Planning*) обеспечит руководство предприятием средствами управления производством, отвечающим современным принципам управления предприятием:

1) производственная деятельность описывается как поток взаимосвязанных заказов;

2) при выполнении заказов учитываются ограничения ресурсов;

3) обеспечивается минимизация производственных циклов и запасов;

4) заказы снабжения и производства формируются на основе заказов реализации и производственных графиков;

5) движение заказов увязывается с экономическими показателями;

6) выполнение заказа завершается к тому моменту, когда он необходим;

7) планирование потребностей в материалах на основе данных о составе изделий и складских запасов;

8) формирование производственной программы в масштабе всего предприятия и контроль ее выполнения на уровне подразделений;

9) прогнозирование, планирование и контроль производства по всему циклу, начиная с закупки сырья и заканчивая отгрузкой товара потребителю;

10) планирование потребностей в распределении и ресурсах.

Важнейшим и стратегическим инструментом достижения генеральной цели предприятия является создание ИИС организационного управления предприятием. Управление предприятием предполагает постоянный анализ деятельности компании и сравнение фактических результатов с запланированными, включая мониторинг качества, производительности и финансов, а также управление кадрами и основными средствами предприятия. Поэтому главной задачей ИИС организационного управления должно быть обеспечение информационной поддержки управленческих задач и решений. ИИС должна интегрально объединить в единую логистическую цепочку весь цикл жизнедеятельности предприятия:

Закупка → производство → продажи → бюджет → затраты → финансы → бухучет

Исходя из этих предпосылок создания и развития, рассматриваются актуальные проблемы создания современных ИИС организационного управления предприятием, признанных обеспечить поддержку следующих функций жизнедеятельности предприятия:

1) бизнес-планиерование: стратегическое и текущее планирование; составление бюджета; управление активами и основными средствами;

2) контроллинг: учет затрат по процессам и местам возникновения, анализ рентабельности;

3) финансы и бухгалтерский учет: главная книга, бухгалтерия дебиторов и кредиторов, регулярный и проектный бухгалтерский учет;

4) контроллинг деятельности предприятия: поддержка принятия решений и ведение отчетности;

5) управление основными средствами: учет, история изменений, амортизация.

2. *Анализ существующих автоматизированных систем управления предприятием*

Проведенный анализ состояния существующих АСУ предприятием на ряде крупнейших и средних (металлургических) предприятий показывает, что существующие средства информационной поддержки управленческих решений в корне не отвечают современным задачам управления предприятием [2]. Статистические данные характеристик информации в существующих АСУ показывают, что:

1) отсутствует информация о деятельности предприятия - 31%;

2) запаздывание информации в объеме - 32%;

3) неполная информация в объеме - 33%;

4) способ получения информации за счет компьютеризации - 25%.

Таким образом, базовыми причинами проблем управления предприятием являются:

1) отсутствие полной и целостной информации о деятельности предприятия;

2) поступающая руководству предприятия информация не достоверна, противоречива и не дает полной и точной картины производства, финансового и материального хозяйства предприятия;

3) информация к руководству для принятия решения поступает не оперативно, со значительным опозданием по времени совершения события, когда принимать решения уже поздно;

4) отсутствует единая система управления логистическими цепочками производства.

Проведенный анализ бизнес-процессами управления предприятием (на ряде металлургических предприятий) показал:

- в управлении бюджетом и финансовыми потоками имеются следующие недостатки:

- бюджет не носит всеобъемлющего характера;
- недопустим длительный срок составления бюджета, отсутствует расчет ожидаемых показателей бюджета и финансового плана на начало месяца;
- крайне запоздалое составление отчета об исполнении бюджета, отчет не всегда совпадает с бухгалтерскими данными;
- нет текущего контроля исполнения бюджета, только месячный отчет;
- в процессе планирования не используются нормативы по дебиторской и кредиторской задолженности, текущий анализ динамики изменения структуры задолженности не производится;

- в управлении затратами в цехах основного производства выявлено:

- отсутствует система оперативного учета производственной себестоимости;
- имеются недостатки в методике анализа себестоимости, отсутствует маржинальный[*)] анализ;
- фактическая себестоимость рассчитывается с большим опозданием (до 20 дней после окончания месяца); отсутствует эффективный контроль за совершаемыми затратами, в том числе по конкретным заказам.

Анализ программно-технических средств АСУ показывает, что создание АСУ велось в направлении собственных разработок без достаточной информационной, методологической и организационной поддержки, АСУ создавалась в рамках централизованного (планового) управления предприятием.

Архитектура системы представляла собой разрозненные куски подсистем автоматизации.

[*)] маржа (фр. *Marge*, поле страницы, край) – термин, применяемый в банковской, биржевой, торговой, страховой практике для обозначения разницы между процентными ставками, курсами ценных бумаг, ценами товаров и др. показателями.

Была создана кусочная, островная технология учета, обработки и представления информации для руководства предприятия и принятия решений по управлению предприятием.

В результате существующие АСУ в части программно-технических средств обладают существенными недостатками:

1) информационная база состоит из ряда локальных БД, слабо взаимосвязанных друг с другом;

2) ввод данных в подавляющем большинстве характеризует события «задним числом»;

3) недостаточная поддержка процессов планирования, особенно стоимостных показателей;

4) отсутствие интеграции данных оперетивного и бухгалтерского учета;

5) множество устаревших программ;

6) зависимость от отдельных разработчиков, являющихся носителями «ноу-хау»;

7) серьезные проблемы по администрированию и управлению множеством БД и сетей;

8) проблемы рассогласованности в работе отдельных подразделений по поддержанию и развитию существующих АСУ;

9) разнообразие технических средств, системного и прикладного программного обеспечения.

3. *Основные пути развития современных ИИС* организационного *управления предприятием*

Решение задач по развитию современных ИИС организационного управления требует больших усилий и затрат интеллектуального труда, огромных инвестиций и времени. Всего этого как раз и не хватает предприятиям. Поэтому развитие ИС должно осуществляться осознанно, поэтапно и с большой аргументированностью принимаемых решений. Прежде всего, в основу базисных решений реорганизации предприятия должен быть положен интегрированный подход. В первую очередь руководство предприятия должно определить стратегические задачи развития бизнеса на своем предприятии. На основе стратегии развития строится модель управления, разрабатываются бизнес-процессы, разрабатывается системный проект реорганизации, где определены требования к ИИС, разработаны основные решения и установлены этапы создания инфраструктуры предприятия.

Анализ зарубежных ИИС управления крупнейших западных фирм-разработчиков, таких как *SAP AG* (*R*3) - Германия, *ORACLE* (*ORACLE Application*) - США, *Computer Association* (*MAN/MANX*) - США, *AVALON Software* - США, *IFS Application* (*System* IV) - Швеция, опыт внедрения систем на крупнейших предприятиях *Krupp-Hoesch, Mannesmann* (Германия), *VOEST ALPINE* (Австрия), *DANIEL* (Италия) показывают существенное (коренное) преимущество ИИС, которые выводят предприятие на качественно новый уровень управления деятельностью предприятия [2].

По данным фирмы *VOEST ALPINE* (Австрия) при внедрении ИИС управления резко улучшились показатели рентабельности производства:

- себестоимость продукции понизилась на 10%;
- запасы сырья уменьшились на 50%;
- время производственного цикла сократилось на 30%;
- горячая загрузка увеличилась на 15%.

Мировой опыт и отечественная практика показывают, что внедрение современных интегрированных систем – путь коренной перестройки производства и значительного ускорения развития экономики страны.

Создание ИИС оперативного управления предприятием требует решения множества задач по планированию и оптимизации всей логистической цепочки, что приведет к созданию фундаментальной базы информатизации, охватывающей все области деятельности предприятия. Внедрение ИИС оперативного управления предприятием позволяет всесторонне анализировать информацию и быстро реагировать на конкретные тенденции развития бизнеса, что значительно повысит возможности предприятия по стратегическому планированию и принятию решений.

1.4. Методы оценки эффективности информационных систем

1. *Проблемы оценки экономической эффективности ИС*

Основные понятия.

Эффект - показатель (результат), характеризующий величину выгодности применения системы.

Эффективность - сопоставление эффекта от реализации инвестиций в системы с величиной затрат, необходимых для их внедрения.

Под оценкой эффективности работы ИС подразумевается наличие методики, которая позволяет количественно оценить результат ее внедрения и эксплуатации.

Любой руководитель, получив информацию о затратах на автоматизацию, задумывается над тем, что, собственно, это даст его предприятию и как вообще соотнести между собой затраты на установку системы и пользу от неё.

Каждое предприятие решает этот вопрос самостоятельно, сравнивая с аналогами. В качестве критериев оценки ИС выступают ожидания лиц, принимающих решения, различия реальных показателей и ожидаемых. Кроме того, уже в 1970-е годы высказывалось мнение, что расчёты эффективности автоматизированных систем не так уж и необходимы. Например, все организации имеют телефоны, но никто не рассчитывает экономическую эффективность телефонизации предприятия. В опровержение такого подхода можно привести следующий аргумент: затраты на разработку и внедрение информационной системы, как правило, значительно более высоки, и руководители часто не решаются на них без оценки эффективности.

Разработка, внедрение проектов по автоматизации функций управления связаны с инвестициями, вложениями средств, затратами рабочего времени и труда. Денежные затраты на автоматизацию представляют собой капитальные затраты на разработку и внедрение проекта и эксплуатационные текущие расходы. К капитальным затратам следует также относить:

1) стоимость проектных работ, расходы по постановке и алгоритмизации задач;
2) затраты на приобретение технических средств, оборудования, инвентаря,
3) затраты на монтаж, установку технических средств;
4) затраты на приобретение программных средств;
5) затраты на создание информационной базы (базы данных);
6) стоимость внедрения;
7) затраты на обучение.

Эксплуатационные текущие расходы включают в себя:

1) затраты на электроэнергию;
2) содержание помещений;
3) сопровождение программного обеспечения (поддержание в работоспособном состоянии, обновление, замена версий);

4) сопровождение информационной базы (восстановление целостности, архивирование и резервное копирование, антивирусная защита, управление доступом);

5) затраты на ремонт и обслуживание технических средств;

6) стоимость расходных (бумага, краска и картриджи для принтера) и прочих вспомогательных материалов.

Целесообразно учитывать и затраты живого труда на выполнение перечисленных работ, особенно время работников, выполняющих их без дополнительной оплаты в своё рабочее время.

Любой руководитель, получив полную или (чаще) не совсем полную информацию о затратах на автоматизацию, задаётся вопросом о том, какую отдачу, какой полезный результат они могут дать и как сопоставить затраты и «полезность». Известные экономические показатели, такие как экономия затрат, в том числе на обработку информации, коэффициент окупаемости и коэффициент эффективности, годовой прирост прибыли от внедрения автоматизированной системы на практике рассчитать чаще всего невозможно. Проблема состоит в оценке полезного эффекта от внедрения автоматизированной системы. Ясно, что основная часть исходных данных не может быть абсолютно точной. В некоторых ситуациях экономическую эффективность работ по автоматизации вообще не удаётся рассчитать с желаемой точностью и убедительностью.

Проблема получения исходных данных для расчёта может решаться с помощью различных методов:

1) На основе результатов деятельности предприятия за прошлые годы прогнозируются экономическими методами результаты на интересующий год и сравниваются с фактическими. Но для достаточно точного прогнозирования условия работы предприятия не должны претерпевать существенных изменений, но при рыночной экономике условия работы предприятия постоянно меняются.

2) Нормативные методы оценки ожидаемой экономической эффективности информационной системы возможны при наличии таких нормативов. Но их нет. Причина — разнообразие предприятий, автоматизированных систем, условий их работы. Наиболее подробно разработанные и утверждённые нормативы Минтруда России на работы по бухгалтерскому учёту и финансовой деятельности в условиях автоматизации даже не упоминают, какие, собственно, условия имеются в виду. А время, затрачиваемое на обработку конкретного документа, будет зависеть

и от применяемого программного комплекса, и от технических средств, скорости передачи информации по сети, наличия и качества информации в массивах справочных данных, организации этих массивов и множества других факторов.

3) Экспертные методы. Это довольно тонкая процедура, её результаты могут зависеть от компетентности экспертов, чёткости постановки задачи, информации, предоставляемой экспертам, и т. д.

4) Имитационные методы состоят в моделировании на компьютере самого объекта управления, системы управления, основных возмущений, действующих на объект, и основаны на использовании математических моделей, т. е. достаточно сложного математического аппарата.

На оценку эффективности автоматизированной системы влияет ещё ряд факторов, о которых пойдёт речь ниже.

Даже при самом удачном проектировании информационная система не может долгое время оставаться неизменной, требует постоянной модернизации, особенно в современных условиях.

Использование ИС для принятия решений требует времени на освоение, адаптацию к новым условиям. Эффект проявляется не сразу, а через какое-то время.

Для расчета экономии затрат на обработку информации нужно сравнить затраты на обработку информации или на управление в целом при двух вариантах обработки информации:

$$DC = C1 - C0, \qquad (1.1)$$

где C1 - затраты на обработку информации до внедрения проекта; C0 - затраты на обработку информации после внедрения проекта.

Затраты на обработку информации до внедрения проекта можно оценить по фактическим показателям, после внедрения проекта - с помощью проектных, плановых, нормативных показателей. Нормативов, формул для их расчёта, заслуживающих доверия нет. Таким образом, реально оценить проект можно только с применением метода экспертных оценок.

Формула (1.1) предполагает, что затраты на обработку информации с применением средств автоматизации уменьшатся. Реально могут быть сокращены затраты на заработную плату при условии сокращения рабочих мест. Но практика показывает, что этого чаще всего не происходит. Перечислим возможные причины:

1) Не повышается производительность труда. Хотя ряд рутинных вычислений производит компьютер, но больше, чем прежде, требуется времени - на сбор и регистрацию информации, на поиск и исправление допущенных ошибок. Программно-технический комплекс в комплекте с сетью нередко работает довольно медленно, интерфейс программы неудобный, а из-за всего этого и весь процесс идёт медленно. Недостаточная квалификация работников не позволяет в полной мере использовать возможности средств автоматизации. Часть времени затрачивается на вспомогательные операции, устранение неполадок и сбоев, ожидание помощи ИТ-специалиста.

2) Реально производительность труда повышается за счёт автоматизации громоздких рутинных расчётов, автоматического формирования выходных форм, сокращения времени на поиск информации, подготовку документов. При повышении производительности труда также чаще всего не происходит сокращения работников. Их обязанности изменяются, высвобождаемое время затрачивается на другие, более творческие, аналитические работы. Кроме того, объявление о возможном предстоящем сокращении штатов влечёт за собой саботаж со стороны персонала, что нередко приводит к провалу проекта.

3) При сокращении работников целесообразно повысить заработную плату оставшимся, что сохраняет на предприятии квалифицированные кадры, повышает его престиж. Поэтому экономии также не будет.

Основными показателями эффективности работы предприятия являются объём реализации, прибыль, рентабельность. Но на их формирование воздействует множество факторов, выделить из которых влияние автоматизации возможно только с применением громоздкого математического аппарата, да и то не очень точно. Из имеющегося фактического прироста вычитаются значения, достигнутые за счёт факторов, влияние которых на формирование прибыли поддаётся учёту. Это применение новых технологических процессов, оборудования, рационализации, изменение цен, изменение численности работающих и т.д. Естественно, что учёт как самих этих факторов, так и воздействия их на прибыль является отдельной трудоёмкой задачей и производится с большими погрешностями.

Срок окупаемости рассчитывают делением суммы капитальных затрат на экономию затрат на обработку информации или на годовой прирост прибыли, коэффициент эффективности — обратный показатель.

К примеру, имеющиеся оценки, что внедрение систем управления отношениями с потребителями, *CRM*-систем (*Customer Relationship Management),* увеличивает объём реализации на 10-30%, получены экспертным путём.

При определении годового прироста прибыли целесообразно проанализировать влияние автоматизации на себестоимость продукции, что также может быть произведено с помощью вышеперечисленных методов.

Внедрение информационной системы позволяет сократить запасы сырья и материалов за счёт совершенствования организации материально-технического снабжения, хранения материалов, более экономного их использования; снижать остатки незавершённого производства и готовой продукции в результате улучшения планирования производства, отгрузки, контроля за соблюдением договоров.

Оборотные средства, потребляемые предприятием, определяются величиной потребляемых ценностей и временем их нахождения на складах и в производстве. Потребность в оборотных средствах можно рассчитать по формуле:

$$O = P^* T / 360, \qquad (1.2)$$

где О - оборотные средства; Р - ресурсы, потреблённые предприятием в течение года; Т - время нахождения ресурсов в распоряжении предприятия (в днях).

При уменьшении времени нахождения ресурсов на предприятии можно рассчитать экономию оборотных средств.

Представляет интерес формула определения снижения потерь в выпуске продукции:

$$П = tn\ q, \qquad (1.3)$$

где П - потери в выпуске продукции; *tn* - время реакции системы управления на возмущения; *q* - выпуск продукции за единицу времени.

$$tn = t1 + t2 + t3 + t4 + t5. \qquad (1.4)$$

Время реакции *tn* состоит из времени получения данных о возмущении (*t*1), времени уяснения состояния используемых ресурсов (*t*2), времени выбора варианта решения (*t*3), времени доведения решения до исполнителей (*t*4), времени исполнения (*t*5).

Влияние автоматизации на устранение причин потерь рабочего времени (очереди за инструментом, время выдачи сменных заданий) можно определить с помощью фотохронометража. Также целесообразно оценить экономию времени управленческих работников на выполнение некоторых наиболее трудоёмких операций.

Большинство авторов, исследовавших эту тему, сходятся во мнении, что невозможно выразить полный эффект от внедрения системы автоматизации в деньгах. Предлагается выделить косвенные показатели эффективности, которые отражаются на результатах производственно-хозяйственной деятельности не непосредственно, а за счёт повышения уровня управления, оперативности и действенности принимаемых решений, совершенствования всей информационной системы объекта, т.е. косвенным путём. Эти косвенные показатели практически невозможно оценить в денежных единицах. При расчётах экономической эффективности информационной системы возникает проблема преобразования эффекта такого рода в экономические показатели, для чего нужны специальные методы.

К таким показателям относят:

1) повышение оперативности и актуальности информации;

2) сокращение сроков решения отдельных задач и принятия управленческих решений;

3) повышение качества информации, её точности, детальности, объективности, в том числе за счёт сокращения ошибок. При этом повышаются производительность труда и качество обслуживания клиента;

4) углубление анализа, повышение качества аналитических отчётов, получение принципиально новых аналитических возможностей;

5) снижение совокупного количества времени, затрачиваемого на подготовку документов, скорость выдачи выходных документов, отсутствие очередей. На основе заключённого договора система автоматизации позволяет быстро подготовить акт, накладную на отгрузку и др.;

6) повышение качества обслуживания клиента и процента удержания старых клиентов;

7) повышение эффективности маркетинговых компаний. В процессе общения с клиентом информация об источниках рекламы связывается с маркетинговой компанией;

8) снижение дебиторской задолженности за счёт автоматического отслеживания выставленных счетов, сроков платежей, погашения задолженности;

9) при автоматизации складского учёта - сокращение запасов на складах;

10) усиление контроля, предотвращение хищений, злоупотреблений;

11) эффект снижения влияния кадровой «текучки» на производственные показатели. Например, информация о клиентах остаётся в информационной базе, а не «уходит» вместе с менеджером;

12) повышение коллективизма, улучшение деловой обстановки в результате открытости информации;

13) повышение качества труда за счёт сокращения рутинных операций;

14) повышение квалификации сотрудников при внедрении новых технологий;

15) совершенствование работы аппарата управления;

16) улучшение организации производства.

Даже эстетичность рабочего кабинета (современный компьютер) может оказать влияние на престиж предприятия, доверие к нему клиента, готовность квалифицированных кадров сотрудничать с этой фирмой.

В настоящее время чуть ли не единственно возможный путь определения эффективности инвестиций в информационные технологии для российских предприятий заключается в том, чтобы получить ответ на следующий вопрос: можно ли ценой выделенных на автоматизацию средств достичь заданных целей, которые формулируются как параметры автоматизируемых процессов?

Например: составлять квартальный баланс в течение недели, получать данные о товарах на складе в течение заданного времени и т. д.

Достигли мы поставленных целей или нет, предлагается оценивать в значениях *true/false* (да/нет). Также известен балльный метод оценки мероприятия по совершенствованию управления.

2. *Методы оценки эффективности информационных систем.*

Методика оценивания эффективности функционирования ИС предполагает наличие соответствующих показателей, значения которых могут быть получены и могут быть учтены используемой методикой. Обзор ме-

тодов и средств оценивания эффективности систем позволил выделить следующие методы оценивания эффективности различных типов систем:

1) Затратные методы.
2) Методы оценки прямого результата.
3) Методы оценки идеальности бизнес процесса.
4) Методы комплексной оценки качества бизнеса.

2.1. *Затратные методы*

Методы этой группы оценивают работу системы не на основе измерения результатов, а на основе затраченных ресурсов. При явных недостатках такого подхода он часто использовался при оценке информационной деятельности в советское время и применяется для грубых и/или трудоемких подсчетов в настоящее время. К таким методам также можно отнести и подход пороговых значений применимости разнородных систем, который не требует количественного измерения результата их деятельности, предлагая оценивать системы по соотношению затрат на внедрение и сопровождение системы с размерами предприятий и направлениями бизнеса.

2.2. *Методы оценки прямого результата*

Методики оценивают прямой, измеримый результат, следующий из реализации проекта. Например, повышение коэффициента выхода или снижение трудозатрат, или появление побочного продукта основного производства. Результат оценивается по текущим рыночным ценам, и если его недостаточно для обоснования прямых инвестиций, добавляется один или несколько дополнительных результатов.

2.3. *Методы оценки идеальности бизнес-процесса*

В сравнительных алгоритмах таких методик, как правило, базовым показателем выбирается объем реализации основной продукции оцениваемого (улучшаемого) бизнес-процесса. Тогда «идеальным» считается бизнес-процесс с лучшими для отрасли показателями затрат на единицу выхода. Почти все методы требуют предварительного описания составляющих бизнес-процессов анализируемого предприятия, поэтому их применение весьма трудоемко. Методами данной группы особенно часто пользуются консультанты, специализирующиеся на какой либо определенной отрасли.

2.4. *Методы комплексной оценки качества бизнеса*

Такие методы комплексно рассматривают понятие «качество продукции предприятия» как соответствие стандартам и технологическим требованиям, как основную характеристику продукта и как набор его по-

требительских свойств. При этом на первый план выходят стратегические задачи создания и воплощения политики качества. Для измерения и обработки полученных оценок используется весь аппарат теории измерений: статистические, экспертные и социологические методы.

Использование прямых и затратных методов крайне затруднено, либо вовсе невозможно в силу отсутствия показателей, способных напрямую повлиять на выпускаемую продукцию (объем производства, ассортимент и т.д.).

Использование методов оценки идеальности бизнес-процесса допустимо в случае, когда есть аналогичный бизнес-процесс с известными показателями, либо присутствует тесная связь процессов оцениваемой системы с процессами производства. Поэтому возможность применения таких методов в данной работе не рассматривается ввиду слабой связанности ИС с конечным результатом основного производства.

Методики комплексного оценивания качества бизнеса наиболее тонко позволяют подойти к рассматриваемой задаче. Они позволяют оценивать системы исходя из комплексной системы критериев, ориентированных на ее специфику, ее внутренние бизнес-процессы. Один из таких методов данной группы и предлагается к использованию при оценивании эффективности ИС предприятия.

3. Методика BSC как способ оценивания эффективности ИС

При оценке эффективности автоматизированных информационных систем возможно и целесообразно применение *Balanced Scorecard* - системы сбалансированных показателей, разработанной в 1992 году Капланом и Нортоном. Эта теория применяется в основном для оценки эффективности управления предприятием. Применение системы сбалансированных показателей для оценки эффективности автоматизированных информационных систем, непосредственно оказывающих влияние на эффективность управления, представляет собой интерес. Данная тема рассмотрена в работе Кострова А.В. и Матвеева Д.А. [4].

Сбалансированная система показателей (*Balanced Scorecard, BSC*) [5] – это методика анализа состояния компании. Это система стратегического управления организацией на основе измерения и оценки ее эффективности по набору показателей, подобранному таким образом, чтобы учесть все существенные с точки зрения стратегии аспекты ее деятельности (финансовые, производственные, маркетинговые и т.д.). Целесообразность использования данной методики оправдано следующими причинами:

1. Наличие достаточной теоретической информации по принципам и подходам к построению *BSC* систем.

Подразумевается, что разработчики обладают достаточной информацией и литературными источниками по теоретическим основам построения системы показателей *BSC*. Это дает уверенность в том, что методика *BSC* с учетом особенностей ее применения на предприятии может быть доведена до уровня практической реализации для использования. Это, в свою очередь, в конечном счете, позволит успешно использовать ее и в дальнейшем, при оценивании эффективности функционирования ИС предприятия в случае, если ИС будет претерпевать изменения.

2. Иерархичность и последовательность процесса построения *BSC* систем.

Принципы и подходы, лежащие в основе разработки *BSC* систем, позволяют говорить о том, что факторы и компоненты ИС предприятия могут быть включены в нее с учетом некоторых принципов.

а) Учет направленности и области применения того или иного показателя. Например, часть показателей можно отнести только к внутренним бизнес-процессам системы и предприятия, а часть - к финансовой составляющей системы;

б) Руководствуясь принципом «от простого к сложному», начиная с низших компонентов системы (и их показателей эффективности) и заканчивая стратегическими целями и задачами ИС можно выстроить систему критериев, не противоречащих структуре ИС и структуре самого предприятия.

3. Непротиворечивость нормативным документам как базисным документам, являющимся основным при разработке и сертификации ИС.

Предлагаемая к рассмотрению методика позволяет без особых модификаций успешно применять ее для ИС. При этом будут соблюдены принципы и требования к методикам оценивания ее эффективности, не противоречащим требованиям ГОСТ.

4. Возможность интегрального оценивания эффективности работы ИС с использованием одного показателя, вычисляемого на совокупности других показателей системы, отражающих все аспекты ее деятельности.

Следует отметить, что оценка эффективности функционирования ИС сопряжена с некоторыми трудностями. Основной из них является то, что ИС - это неосновная сфера деятельности предприятии, не приносящая основной доход предприятию. Таким образом, прямое использование систе-

мы экономических показателей, в данном случае, неприменимо. Поэтому при выборе показателей определяют неосновные показатели, учет которых объективно возможен в процессе функционирования предприятия и ИС, как неотъемлемой части предприятия. Наличие возможности систематически получать систему показателей и их значений позволяет впоследствии выстроить из них общий критерий оценки эффективности функционирования ИС. Построенная система показателей позволит включить их в итоговый показатель, использование которого будет оправдано исходя из глобальной цели ИС предприятия.

МЕТОДОЛОГИЯ И ТЕХНОЛОГИЯ РАЗРАБОТКИ ИНФОРМАЦИОННЫХ СИСТЕМ

2.1. Методология *RAD*

1. *Методология RAD и основные ее особенности*

Методология создания ИС заключается в организации процесса построения ИС и обеспечении управления этим процессом для того, чтобы гарантировать выполнение требований, как к самой системе, так и к характеристикам процесса разработки.

Разработка, разработать - тщательно, всесторонне исследовать, подготовить, обработать во всех подробностях.

Основными задачами, решение которых должна обеспечивать методология создания корпоративных ИС, являются:

- обеспечение создания ИС, отвечающих целям и задачам предприятия и соответствующих предъявляемым к ним требованиям по автоматизации деловых процессов;

- гарантия создания систем с заданными параметрами в течение заданного периода времени в рамках оговоренного заранее бюджета;

- простота сопровождения, модификации и расширения системы с целью обеспечения соответствия изменяющимся условиям работы предприятия;

- обеспечение создания корпоративных ИС, отвечающих требованиям открытости, переносимости и масштабируемости;

- возможность использования в создаваемой системе разработанных ранее и применяемых на предприятии средств ИТ.

Методологии, технологии и инструментальные средства проектирования (*CASE*-средства) составляют основу проекта любой ИС.

Проект – разработанный план сооружения, какого-нибудь механизма, устройства.

Методология реализуется через конкретные технологии и поддерживающие их стандарты и инструментальные средства, которые обеспечивают выполнение процессов жизненного цикла (ЖЦ) ИС.

Основное содержание технологии проектирования составляют технологические инструкции, состоящие из описания последовательности

технологических операций, условий, в зависимости от которых выполняется та или иная операция, и описание самих операций.

Технология проектирования может быть представлена как совокупность трех составляющих:

1) заданной последовательностью выполнения технологических операций проектирования;

2) критериев и правил, используемых для оценки результатов выполнения технологических операций;

3) графических и текстовых средств (нотаций), используемых для описания проектируемой системы.

Результаты выполнения операций должны представляться в некотором стандартном виде, обеспечивающем их адекватное восприятие при выполнении следующей технологической операции.

Можно сформулировать ряд требований, которым должна удовлетворять технология проектирования, разработки и сопровождения ИС:

- поддерживать полный ЖЦ ИС;

- обеспечивать гарантированное достижение целей разработки системы с заданным качеством и в установленное время;

- обеспечивать возможность разделения крупных проектов на ряд подсистем, разрабатываемые группами исполнителей ограниченной численностью, с последующей интеграцией составных частей;

- обеспечивать возможность ведения работ по проектированию отдельных подсистем группами 3-7 человек (из принципа управляемости коллектива);

- обеспечивать минимальное время получения работоспособной системы;

- предусматривать возможность управления конфигурацией проекта, ведения версий проекта и его составляющих, возможность автоматического выпуска проектной документации и синхронизацию ее версий с версиями проекта;

- обеспечить независимость проектных решений от средств реализации системы – СУБД, ОС, языка и системы программирования.

На начальных этапах существования ИС их разработка велась на традиционных языках программирования. Сокращение сроков разработки программного обеспечения потребовало создания инструментальных средств для быстрой разработки приложений. Развитие этого направления

привело к появлению на рынке программного обеспечения средств автоматизации практически всех этапов ЖЦ ИС.

Методология разработки ИС, основанная на использовании средств быстрой разработки приложений, получила в последнее время широкое распространение и получила название методологии быстрой разработки приложений – *RAD (Rapid Application Development)*.

Под методологией быстрой разработки приложений понимают процесс разработки ИС, основанный на трех основных элементах:

1) небольшое количество программистов (обычно 2-10 человек);

2) тщательно проработанный производственный график работ, рассчитанный на короткий срок разработки (2-6 месяцев);

3) итерационная модель разработки, основанная на тесном взаимодействии с заказчиком – по мере выполнения проекта разработчики уточняют и реализуют в проекте требования, выдвигаемые заказчиком.

Основные принципы методологии *RAD* сводятся к следующему:

1) используется итерационная модель разработки;

2) полное завершение работ на каждом этапе не обязательно;

3) в процессе разработки ИС необходимо тесное взаимодействие с заказчиком и будущими пользователями;

4) необходимо применение *CASE*-средств и средств быстрой разработки приложений;

5) необходимо применение средств управления конфигурацией, облегчающих внесение изменений в проект и сопровождение готовой системы;

6) необходимо использование прототипов, позволяющее полнее выяснить и реализовать потребности конечного пользователя;

7) тестирование и развитие проекта осуществляются одновременно с разработкой;

8) разработка ведется не многочисленной и хорошо управляемой командой профессионалов;

9) необходимы грамотное руководство разработкой системы, четкое планирование и контроль выполнения работ.

2. *Объектно-ориентированный подход и визуальное программирование*

Средства *RAD* дают возможность реализовать иную по сравнению с традиционной технологию создания приложений: информационные объекты формируются как некие действующие модели (прототипы), функцио-

нирование которых согласовывается с пользователем, а затем разработчик может переходить непосредственно к формированию законченных приложений, не теряя из виду общей картины системы.

Использование объектно-ориентированных методов позволяет создать описание (модель) предметной области в виде совокупности объектов – сущностей, объединяющих данные и методы обработки этих данных (процедуры). Каждый объект обладает собственным поведением и моделирует некоторый объект реального мира. Объект обладает целостностью, он может только менять состояние, управляться или становиться в определенное отношение к другим объектам.

При разработке приложений с помощью инструментов *RAD* используется множество готовых объектов, сохраняемых в общедоступных хранилищах. Могут при этом разрабатываться и новые объекты.

Инструментальные средства *RAD* обладают удобным графическим интерфейсом пользователя и позволяют на основе стандартных объектов формировать простые приложения без написания кода программ, что сокращает разработку интерфейса пользователя. Высокая скорость разработки интерфейсной части приложений позволяет быстро создавать прототипы и упрощает взаимодействие с конечным пользователем.

Визуальные средства разработки оперируют со стандартными интерфейсными объектами: окнами, списками, текстами, которые легко можно связать с данными из БД и отобразить на экране монитора. Другая группа объектов представляет стандартные элементы управления - кнопки, переключатели, флажки, меню и т.п., с помощью которых осуществляется управление отображаемыми данными. Все эти объекты могут описываться средствами языка, а сами описания сохранены для дальнейшего использования.

В настоящее время существует много визуального средств разработки приложений. Их можно разделить на универсальные и специализированные.

Среди универсальных систем визуального программирования наиболее распространены такие, как *Borland Delphi, Visual Basic*. Универсальность заключается в том, что с их помощью можно разрабатывать не только приложения баз данных, но и информационные приложения и др. Они могут взаимодействовать с любыми СУБД.

Специализированные средства разработки ориентированы только на создание приложений БД и привязаны к определенным СУБД. Примерами

таких систем являются *Power Builder* фирмы *Sybase, Visual FoxPro* фирмы *Microsoft.*

Поскольку задачи создания прототипов и разработки пользовательского интерфейса, по существу слились, программист получил непрерывную обратную связь с конечным пользователем, который может активно участвовать и корректировать результаты и свои требования. Это сокращает сроки разработки.

Визуальные инструменты *RAD* позволяют максимально сблизить этапы создания ИС: анализ, проектирование системы, разработка прототипов, окончательное формирование приложений становятся сходными, т.к. на каждом этапе разработки оперируют визуальными объектами.

Логика приложения, построенная с помощью *RAD*, является событийно ориентированной. Каждый объект, входящий в состав приложения, может генерировать события и реагировать на события, генерируемые другими объектами (например, открытие и закрытие окон, нажатие клавиши клавиатуры, изменение данных в БД и др.).

Обработчики событий, связанных с управлением БД (*DELETE, INSERT, UPDATE)*, могут реализоваться в виде триггеров на клиентском или серверном узле. Такие обработчики позволяют обеспечить ссылочную целостность БД при операциях удаления, вставки и обновления, а также автоматическую генерацию первичных ключей.

3.*Фазы жизненного цикла в рамках методологии RAD*

ЖЦ ИС состоит из четырех фаз: анализа и планирования требований, проектирования, построения, внедрения. На начальной фазе определяются функции ИС и их приоритетность, описываются информационные потребности, определяются временные рамки проектирования, определяется возможность реализации проекта: в сроки при имеющемся финансировании, на имеющихся аппаратно-программных средствах. Составляются предварительные функциональные и информационные модели системы.

На фазе проектирования с помощью *CASE* средств создаются прототипы приложений. На основе анализа прототипов пользователями уточняются и дополняются требования к системе. Корректируется функциональная модель системы, детально рассматривается каждый процесс системы. Определяются требования разграничения доступа к данным. Информационная система разбивается на ряд подсистем для организации параллельной разработки группами проектировщиков. Результатами выполнения данной фазы являются: общая информационная модель системы, функ-

циональная модель системы в целом и подсистем, реализуемых отдельными командами разработчиков, интерфейсы между автономно разрабатываемыми подсистемами, построенные прототипы экранов, диалогов и отчетов.

На фазе построения выполняется быстрая разработка приложения. Разработчики производят итеративное построение реальной системы на основе полученных ранее моделей. Приложения разрабатываются с использованием визуальных средств программирования. Формирование программного кода выполняется с помощью автоматических генераторов кода, входящих в состав *CASE* средств. Код генерируется на основе разработанных моделей. Пользователи системы оценивают полученные результаты и, при необходимости, вносят коррективы.

После окончания работ каждой команды разработчиков постепенно производится интеграция данной части системы с остальными. Формируется полный программный код, выполняется тестирование совместной работы данной части приложения с остальными, а также тестирование системы в целом.

Завершается физическое проектирование системы: определяется необходимость распределения данных, производится анализ использования данных, производится физическое проектирование БД, определяются требования к аппаратным ресурсам, определяется способ увеличения производительности, завершается разработка документации проекта.

Фаза внедрения сводится к обучению пользователей разработанной ИС.

Применение *RAD* технологии наиболее эффективно при выполнении сравнительно небольших систем, разрабатываемых для вполне определенного предприятия.

Методология *RAD* не применима для создания типовых ИС, построения сложных расчетных программ, операционных систем или программ управления сложными инженерно-техническими объектами, программ, требующих написания большого объема уникального кода, приложений реального времени, драйверов и служб.

Совершенно неприемлема методология *RAD* для разработки систем, от которых зависит безопасность людей (системы управления транспортом, атомных электростанций и др.). Это обусловлено тем, что итеративный подход, используемый в *RAD*, предполагает, что первые версии сис-

темы не будут полностью работоспособны, что в данном случае может привести к серьезным катастрофам.

2.2. Стандарты и методики разработки информационных систем

Важным условием эффективного использования ИТ является внедрение корпоративных стандартов. Корпоративные стандарты представляют собой единые правила организации технологии или управления. За их основу могут применяться международные, национальные или отраслевые стандарты.

В связи с прогрессом в области программного обеспечения и средств ВТ наблюдается рост размеров и сложности ИС. Существенно меняются требования как к основным функциям, сервисным возможностям систем так и к динамике их изменения. В этих условиях применение классических способов разработки и обеспечения качества ИС становится малоэффективным и не отвечающим требованиям к качеству систем.

Полезны в этом отношении стандарты открытых систем (интерфейсы различных видов, протоколы взаимодействия). Однако разработка систем в новых условиях требует новых методов проектирования и новой организации проектирования ИС. В новых условиях требуется адаптивное планирование разработки, в том числе в динамике процесса ее выполнения. Одним из способов адаптивного проектирования является разработка и применение профилей ЖЦ ИС и программного обеспечения (ПО).

Корпоративные стандарты образуют целостную систему, включающую три вида стандартов:

1) стандарты на продукты и услуги;
2) стандарты на процессы и технологии;
3) стандарты на формы коллективной деятельности, или управленческие стандарты.

Существующие на сегодня стандарты можно разделить на группы по следующим признакам:

- по предмету стандартизации, в их числе функциональные стандарты (на языки программирования, интерфейсы, протоколы) и стандарты на организацию ЖЦ создания и использования ИС и ПО;
- по утверждающей организации - международные, национальные стандарты, международных консорциумов и комитетов по стандартизации (международного консорциума *OMG*), стандарты официально никем

не утвержденные (язык взаимодействия с реляционной БД *SQL*, язык программирования *C*), фирменные стандарты (*Microsoft ODBC*).;

- по методическому источнику, к их числу относятся методические материалы ведущих фирм-разработчиков ПО, фирм-консультантов, научных центров, консорциумов по стандартизации.

Далее рассмотрим стандарты и методики, касающиеся организации ЖЦ ИС и ПО: методики *Oracle CDM*, международный стандарт *ISO/IEC* 12207:1995–08–01, национальные стандарты – отечественный комплекс стандартов ГОСТ 34, Великобритании - *SSADM,* США - *SADT*.

1. *Методика Oracle CDM*

Одним из направлений деятельности фирмы *Oracle* стала разработка методологических основ и производство инструментальных средств автоматизации процесса разработки сложных прикладных систем, ориентированных на интенсивное использование баз данных. Методика *Oracle CDM* применяется в *CASE*-средстве *Designer/2000*.

Основу *CASE*-технологии и инструментальных средств фирмы *Oracle* составляют:

- методология структурного нисходящего проектирования;
- поддержка всех этапов ЖЦ прикладной системы;
- организация на реализацию приложений в архитектуре клиент-сервер с использованием всех особенностей современных серверов БД, включая декларативные ограничения целостности, хранимые процедуры, триггеры баз данных, поддержка в клиентской части всех современных стандартов и требований к графическому интерфейсу конечного пользователя;
- наличие централизованной БД, репозитария для хранения спецификаций проекта прикладной системы на всех этапах ее разработки. Репозитарий представляет БД специальной структуры, работающую под управлением СУБД *Oracle*;
- возможность одновременной работы с репозитарием многих пользователей – всех участников разработки системы;
- автоматизация последовательного перехода от одного этапа разработки к следующему: по спецификации концептуального уровня к спецификации уровня проектирования, а затем к генерации готовых к выполнению программ;
- автоматизация стандартных действий по проектированию и реализации приложения: генерация отчетов по содержимому репозитария,

обеспечивающих полное документирование текущей версии системы на всех этапах разработки, возможность проверки спецификаций на полноту и непротиворечивость.

Методика *Oracle CDM* определяет следующие фазы жизненного цикла ИС:

1) стратегия;
2) анализ (формулирование требований к прикладной системе);
3) проектирование;
4) реализация;
5) внедрение;
6) эксплуатация.

На первом этапе анализируются процессы, описывающие деятельность организации, строится модель существующих процессов, выявляются недостатки и возможности усовершенствования.

В результате анализа разрабатывается детальные концептуальные модели предметной области, описывающие информационные потребности организации. Создаются модели двух типов: информационные, отражающие структуру и общие закономерности предметной области; функциональные, описывающие особенности решаемых задач.

На стадии проектирования вырабатываются технические спецификации будущей прикладной системы - определяется структура и состав БД, специфицируется выбор программных модулей. Первоначальный вариант проектных спецификаций получается автоматически с помощью утилит на основании данных концептуальной модели.

На этапе реализации создаются программы, отвечающие требованиям проектных спецификаций. Использование генераторов приложений *Designer*/2000 позволяет автоматизировать этап реализации. Существенно сокращаются сроки разработки системы, повышается ее качество и надежность.

На фазе эксплуатации осуществляется поддержка приложений и слежение за ними, планируются будущие функциональные расширения.

Степень адаптивности *CDM* ограничивается тремя моделями ЖЦ:

1) классическая - предусматривает все этапы;
2) быстрая разработка – ориентирована на использование инструментов моделирования и программирования *Oracle;*
3) облегченный подход – рекомендуется в случае малых проектов и возможности быстрого прототипирования приложений.

Особенностями методологии *Oracle CDM* являются:

- не предусматривает включение дополнительных задач, не оговоренных в *CDM*, и их привязку к остальным. Исключено удаление задач и порождаемых ею документов;
- все модели ЖЦ являются каскадными, сохраняют общий последовательный и детерминированный порядок выполнения задач;
- методика считается фирменным стандартом и не является обязательной. Степень обязательности соответствует ограничениям возможности адаптации;
- прикладная система рассматривается как программно-техническая система. Отсутствует возможность выполнения организационно-структурных преобразований, которые присутствуют при создании новых ИС;
- *CDM* опирается на использование только инструментария *Oracle*;
- методика *Oracle CDM* представляет собой вполне конкретный материал, детализированный до уровня заготовок проектных документов, рассчитанных на использование в проектах ИС с опорой на инструментальные средства и СУБД фирмы *Oracle.*

2. *Международный стандарт ISO/ES* 12207:1995-08-01

ISO/ES 12207 представляет собой базовый стандарт, ориентированный на ЖЦ ПО, ориентирован на различные виды ПО и типы проектов автоматизированных систем. Стандарт определяет стратегию и общий порядок к созданию и эксплуатации ПО. Процессы, используемые во время ЖЦ ПО, должны совмещаться с процессами ЖЦ автоматизированной системы. В отличие от *Oracle CDM* стандарт *ISO/ES* ориентирован на организацию действий разработчика и пользователя ИС. Стандарт не определяет этапы ЖЦ, он определяет лишь ряд крупных, обобщенных процессов: приобретение, поставка, разработка и т.п. Каждый процесс, действие или задача инициируются и выполняются другим процессом по мере необходимости, причем нет заранее определенной последовательности как в *CDM*. При этом сохранятся логика связей по исходным сведениям задач.

В стандарте *ISO/ES* 12207 описаны пять основных процессов ЖЦ ПО:

1) процесс приобретения ИС, ПО или службу программного обеспечения;

2) процесс поставки определяет действия предприятия – поставщика, который снабжает покупателя ИС, ПО или службой программного обеспечения;

3) процесс разработки определяет действия предприятия-разработчика ПО;

4) процесс функционирования определяет действия по обслуживанию системы в целом в процессе ее функционирования в интересах пользователя;

5) процесс сопровождения определяет действия персонала, обеспечивающего сопровождение ПО, модификацию, установку ПО на вычислительной системе и его удаление.

Кроме основных *ISO/ES* 12207 оговаривает 8 вспомогательных процессов, обеспечивающие качество проекта ПО и четыре организационных процесса - управления, создания инфраструктуры, усовершенствования, обучения.

В стандарте определен особый процесс – адаптация, который определяет основные действия, необходимые для адаптации этого стандарта к условиям конкретного проекта.

Особенностями стандарта *ISO/ES* 12207 являются:

- динамический характер способа определения последовательности выполнения процессов и задач;
- максимальная степень адаптивности множества процессов и задач к конкретным проектам ИС;
- стандарт не содержит конкретные методы действий, заготовок решений и документов. Описывает архитектуру процессов ЖЦ ПО;
- стандарт содержит мало описаний, направленных на проектирование БД, т.к. разные системы могут использовать специфические типы БД или вообще не использовать БД и др.

Стандарт содержит наборы задач, характеристики качества, критерии оценки и т.п., дающие всесторонний охват проектных ситуаций. Для анализа требований к ПО предусмотрено 11 классов характеристик качества: функциональные возможности, внешние связи (интерфейсы), требования квалификации, спецификации надежности, защищенности, человеческие факторы, определение данных и требования к БД, установочные и приемочные требования, документация пользователя, работа пользователя, требования сервиса пользователя.

3. *Стандарты комплекса* ГОСТ 34

Объектами стандартизации являются автоматизированные системы различных видов и все виды их компонентов, не только ПО и БД. Комплекс рассчитан на взаимодействие заказчика и разработчика, уделяет внимание содержанию проектных документов и распределению действий между сторонами.

Наиболее популярными в группе ГОСТ 34 являются:

- ГОСТ 34.601-90 Стадии создания автоматизированной системы;
- ГОСТ 34.602-89 Техническое задание на создание автоматизированной системы;
- Методические указания РД 50-34.698-90 требования к содержанию документов.

Стандарты предусматривают стадии и этапы выполнения работ по созданию автоматизированной системы, но не предусматривают сквозных процессов в явном виде.

ГОСТ 34 определяет следующие этапы разработки ИС:

Таблица 2.1 Этапы проектирования ИС по ГОСТ 34.601–90

№ этапа	Наименование этапа
1	Формирование требований к автоматизированной системе.
2	Разработка концепции автоматизированной системы.
3	Разработка технического задания (ТЗ).
4	Эскизный проект.
5	Техническая документация
6	Рабочая документация.
7	Ввод в действие.
8	Сопровождение

Основными особенностями комплекса стандартов ГОСТ34 является следующее:

1) основной целью разработки ГОСТ 34 было разрешение противоречий, возникающих при интеграции систем вследствие несогласованности нормативно-технической документации. В 80-х годах действовали комплексы стандартов для АСУ, ОАСУ, АСУП, АСУТП и др. организационно-экономических систем (24-я система); САПР (23501); АСТПП (14-я система стандартов). Стандарты были несогласованны между собой;

2) адаптивность стандартов ГОСТ 34 обеспечивается тем, что участники работ по созданию автоматизированных систем определяют стадии и этапы работ, которые определяются в договорах и ТЗ;

3) полная обязательность выполнения требований ГОСТ 34 отсутствует. Материалы ГОСТ являются методической поддержкой, в значительной степени ориентированной на заказчика;

4) ТЗ является основным исходным документом для создания системы и ее приемки, определяет точки взаимодействия заказчика и разработчика.

5) ГОСТ 34 определяет автоматизированную систему следующим образом: «система, состоящая из персонала и комплекса средств автоматизации его деятельности, реализующая информационную технологию выполнения установленных функций».

Автоматизированная система рассматривается в первую очередь как персонал, принимающий решения и выполняющий другие управляющие действия, поддержанный организационно-техническими средствами.

4. *Стандарт Великобритании SSADM*

Стандарт *SSADM – Structured System Analysis and Design Method* (метод анализа и проектирования структуры системы), начиная с 70-х годов, стал использоваться при создании промышленных информационных технологий.

В 1981 году данный стандарт *SSADM* был объявлен открытым отраслевым стандартом, обязательным к использованию во всех проектах автоматизированных систем, финансируемых из государственного бюджета.

Таблица 2.2 Этапы проектирования по *SSADM*

№ этапа	Наименование этапа
0	Оценивание реализуемости системы
1	Предпроектное обследование
2	Выбор варианта автоматизации
3	Разработка ТЗ
4	Выбор варианта технической реализации
5	Разработка логического проекта
6	Физическое проектирование

Создатели технологии *SSADM* руководствовались рядом основополагающих принципов:

1) Постоянное вовлечение представителей будущих пользователей в процесс выработки решений на протяжении всего проектирования автоматизированной системы.

2) Четкая структуризация технологического процесса, увязка всех стадий проектирования, этапов и проектных процедур и регламентация ролей всех участников разработки.

3) Эффективный контроль за ходом разработки со стороны руководителя проекта, возможность использования существующих технологий для автоматизированного управления разработкой.

4) Стыковка с технологиями, реализованными в существующих системах программирования и управления базами данных.

5) Формализация процесса разработки, обеспечивающая широкое применение средств автоматизации проектирования.

Продуктом, издаваемым по технологии *SSADM*, является комплект документов, на основе которых может быть реализована разрабатываемая автоматизированная система на вычислительной установке с использованием систем программирования и СУБД.

5. *Стандарт* США *SADT*

Стандарт *SADT – Structured Analysis and Design Technique* (методология структурного анализа и проектирования) введен в США в 1975 году.

Определяет следующие этапы разработки ИС:

Таблица 2.3 Этапы работ, которые включает *SADT*

№ этапа	Наименование этапа
1	Предпроектное обследование
2	Документирование полученных знаний и создание модели первого уровня приложения
3	Корректировка модели, модель второго уровня
4	Разработка логического проекта *IDEF*
5	Динамическая модель – действующий прототип
6	Диаграммы, рекомендуемые к публикации
7	Поэтапное сравнение проекта и технической реализации
8	Сопровождение проекта автоматизированной системы

Стандарт *SADT,* в отличие от отечественного, поддержан целым рядом САПР-ов, построенных на стандартах *IDEF*0*, IDEF*1*, IDEF*1*X*, *IDEF/CPN*, которые являются подстандартами *SADT,* который является языком этих стандартов.

В стандарте *IDEF* первоначально обеспечивается единая информационная среда, а система формализованных правил увязывает в единый комплекс следующие модели:

- семантическую;
- функциональную;
- информационную;
- динамическую.

Предполагается, что проект полностью выполняется в электронном виде.

Сравнивая стандарты можно сделать следующие выводы [6]:

- Ни один из рассмотренных стандартов не является универсальным, описывающим все виды действий и задач, возникающих в конкретных проектах.
- Наиболее широкий набор процессов, действий и задач, охватывающий большинство возможных ситуаций при максимальной адаптируемости, содержится в стандарте *ISO* 12207.
- Перспективной являются схема организации работ и применения инструментария, определяемых стандартом *IDEF*.

ГОСТ 34 достаточно полно и фундаментально определяет:

- систему как объект создания и развития;
- аналитические и исследовательские работы по обоснованию концепции автоматизированной системы;
- виды обеспечения системы, которые согласуются с требованиями *ISO* 12207 к системе и программному обеспечению;

ГОСТ 34 и *CDM* в первую очередь ориентированы на действия по созданию и поддержке системы, а *ISO* 12207 – на приобретение и эксплуатацию систем (разработка логически вытекает из приобретения).

2.3. Профили открытых информационных систем

Создание, сопряжение и развитие современных сложных ИС базируется на методологии построения таких систем, как открытых. Развитие и использование открытых ИС неразрывно связано с применением стандартов на основе методологии функциональной стандартизации ИТ.

При создании и развитии сложных, распределенных, тиражируемых ИС требуется гибкое формирование и применение гармонизированных совокупностей базовых стандартов и нормативных документов разного уровня, выделение в них требований и рекомендаций, необходимых для реализации заданных функций системы. Для унификации и регламентирования такие базовые стандарты должны адаптироваться и конкретизироваться применительно к определенным классам проектов, функций, процессов и компонентов системы. В связи с этим выделилось и сформировалось понятие профиля ИС как основного инструмента функциональной стандартизации.

Профиль – это совокупность нескольких (или подмножество одного) базовых стандартов с четко определенными и гармонизированными подмножествами обязательных и факультативных возможностей, предназначенная для реализации заданной функции или группы функций.

Профиль формируется исходя из функциональных характеристик объекта стандартизации. В профиле выделяются и устанавливаются допустимые возможности и значения параметров каждого базового стандарта и/или нормативного документа, входящего в профиль.

Профиль не должен противоречить использованным в нем базовым стандартам и нормативным документам. Он должен применять выбранные из альтернативных вариантов необязательные возможности и значения параметров в пределах допустимых.

На базе одной совокупности стандартов могут формироваться и утверждаться различные профили для разных проектов ИС. Базовые документы профиля должны обеспечивать качество, совместимость и корректное взаимодействие отдельных компонентов системы, соответствующих профилю в заданной области его применения.

Базовые стандарты и профили могут использоваться как непосредственные директивы, руководящие и рекомендательные документы, а также как нормативная база, необходимая при выборе или разработке средств автоматизации технологических этапов или процессов создания, сопровождения и разработки ИС.

Обычно рассматриваются две группы профилей:

1) регламентирующие архитектуру и структуру ИС;

2) регламентирующие процессы проектирования, разработки, применения, сопровождения и развития системы.

В зависимости от области применения профили могут иметь разные категории и разные статусы утверждения:

- профили конкретных ИС, определяющие стандартизованные проектные решения в пределах данного проекта;

- профили ИС, предназначенные для решения некоторого класса прикладных задач.

Профили ИС унифицируют и регламентируют только часть требований, характеристик, показателей качества и процессов, выделенных и формализованных на базе стандартов и нормативных документов. Другая часть функциональных и технических характеристик системы определяется заказчиками и разработчиками творчески, без учета положений нормативных документов.

1. *Принципы формирования профиля ИС*

Использование профиля ИС призвано решать следующие задачи:

1) снижение трудоемкости проектов;

2) повышение качества компонентов ИС;

3) обеспечение расширяемости и мсштабируемости разрабатываемых систем;

4) обеспечение возможности функциональной интеграции в ИС задач, которые раньше решались раздельно;

5) обеспечение переносимости прикладного ПО.

Актуальность использования профилей ИС обусловлена современным состоянием стандартов ИТ, которые характеризуются следующими особенностями:

- существует множество международных и национальных стандартов, которые не полностью удовлетворяют потребности в стандартизации объектов и процессов создания и применения сложных ИС;

- длительные сроки разработки, согласования и утверждения международных и национальных стандартов приводят к консерватизму и отставанию от современных ИТ;

- функциональными стандартами поддержаны и регламентированы только самые простые объекты и рутинные, массовые процессы: телекоммуникации, программирование, документирование программ и данных. Наиболее сложные и творческие процессы, как системный анализ и проектирование, интеграция компонентов и системы, испытания и сертификация почти не поддержаны требованиями и рекомендациями стандартов из-за трудности их формализации и унификации;

Совершенствование и согласование нормативных и методических документов позволяет создавать на их основе национальные и международные стандарты.

Подходы к формированию профилей ИС могут быть различными. В международной функциональной стандартизации ИТ принято жесткое понятие профиля. Основой профиля могут быть только международные и национальные утвержденные стандарты. Это затрудняет унификацию, регламентирование и параметризацию множества конкретных функций и характеристик сложных объектов, архитектуры и структуры современных ИС.

Другой подход состоит в использовании совокупности адаптированных и параметризованных базовых международных и национальных стандартов и открытых спецификаций, отвечающих стандартам де-факто и рекомендациям международных консорциумов.

Эталонная модель среды открытых систем *OSE/RM* определяет разделение любой ИС на две составляющие: приложения (прикладные программы и программные комплексы) и среду, в которой эти приложения функционируют. Между приложениями и средой определяются стандартизированные интерфейсы – *Application Program Interface (API)*, которые являются необходимой частью профилей любой открытой системы. Кроме того, в профиле могут быть определены унифицированные интерфейсы взаимодействия функциональных частей друг с другом и интерфейсы взаимодействия между компонентами среды системы.

Таким образом, профили ИС с иерархической структурой могут включать в себя:

- стандартизованные описания функций;
- функции взаимодействия системы с внешней для нее средой;
- стандартизованные интерфейсы между приложениями и средой ИС;
- профили отдельных функциональных компонентов, входящих в систему.

Для эффективного использования конкретного профиля необходимо:

1) выделить объединенные логической связью проблемно-ориентированные области функционирования, где могут применяться стандарты, общие для одной организации или группы организаций;

2) идентифицировать стандарты и нормативные документы, варианты их использования и параметры, которые необходимо включить в профиль;

3) документально зафиксировать участки конкретного профиля, где требуется создание новых стандартов и нормативных документов, и идентифицировать характеристики, которые могут оказаться важными для разработки недостающих стандартов и нормативных документов этого профиля;

4) формализовать профиль в соответствии с его категорией, включая стандарты, различные варианты нормативных документов и дополнительные параметры, которые непосредственно связаны с профилем;

5) опубликовать профиль и/или продвигать его по формальным инстанциям для дальнейшего распространения.

При использовании профилей важное значение имеет обеспечение проверки корректности их применения путем тестирования, испытаний и сертификации. Технология проверки и тестирования должна поддерживаться совокупностью методик, инструментальных средств, составом и содержанием оформляемых документов на каждом этапе выполнения проекта.

Использование профилей способствует унификации при разработке тестов, проверяющих качество и взаимодействие компонентов проектируемой ИС. Профили должны определяться таким образом, чтобы тестирование их реализации можно было проводить по возможности наиболее полно по стандартизованной методике.

2. *Структура профилей ИС*

Разработка и применение профилей должны стать частью процессов проектирования, разработки и сопровождения ИС.

Стандарты, важные с точки зрения заказчика, должны задаваться в ТЗ на проектирование системы и составлять ее первичный профиль. То, что не задано в ТЗ, первоначально остается на усмотрение разработчика системы, который в соответствии с требованиями ТЗ может дополнять и развивать профили системы и впоследствии согласовать с заказчиком.

В профиль конкретной системы включаются спецификации компонент, разработанных в составе данного проекта, спецификации использованных готовых программных и аппаратных средств, если эти средства не специфицированы соответствующими стандартами. После завершения проектирования и испытаний системы, в ходе которых проверяется ее со-

ответствие профилю, профиль применяется как основной инструмент сопровождения системы при эксплуатации, модернизации и развития.

Формирование структуры профиля ИС осуществляется на основе использования международных и национальных стандартов, ведомственных нормативных документов, а также стандартов де-факто при условии доступности соответствующих им спецификаций. Для корректного применения профилей их описания должны содержать:

- определение целей использования данного профиля;
- точное перечисление функций объекта или процесса стандартизации, определенного данным профилем;
- формализованные сценарии применения базовых стандартов и спецификаций, включенных в данный профиль;
- сводку требований к ИС или ее компонентов, определяющих соответствие профилю, и требований к методам тестирования соответствия;
- нормативные ссылки на конкретный набор стандартов и др. нормативных документов;
- информационные ссылки на все исходные документы.

На стадии ЖЦ ИС выбираются и затем применяются основные функциональные профили:

- профиль прикладного ПО;
- профиль среды ИС;
- профиль защиты информации в ИС;
- профиль инструментальных средств, встроенных в ИС.

2.1. *Профили прикладного ПО*

Прикладное ПО всегда является проблемно-ориентировнным и определяет основные функции ИС. Функциональные профили системы должны включать в себя согласованные базовые стандарты. Профили должны согласоваться между собой. Необходимость такого согласования возникает при использовании стандартизованных *API*, в том числе интерфейсов приложений со средой их функционирования и со средствами защиты информации. При согласовании функциональных профилей возможны также уточнения профиля среды системы и профиля встраиваемых инструментальных средств создания, сопровождения и развития прикладного ПО.

2.2. *Профиль среды ИС*

Профиль среды ИС должен определять ее архитектуру в соответствии с выбранной моделью обработки данных. Стандарты интерфейсов

приложений со средой (*API*) должны быть определены по функциональным областям профилей ИС. Декомпозиция структуры среды функционирования системы на составные части, выполняемая на стадии эскизного проектирования, позволяет детализировать профиль среды ИС по функциональным областям эталонной модели *OSE/RM:*

- области графического пользовательского интерфейса;
- область реляционной или объектно-ориентированных СУБД;
- область операционных систем с учетом сетевых функций, выполняемых на уровне ОС;
- область телекоммуникационной среды в части услуг и служб прикладного уровня: электронной почты, доступа к удаленным БД, передачи файлов, доступа к файлам и управления файлами.

Профиль среды распределенной ИС должен включать стандарты протоколов транспортного уровня, стандарты локальных сетей (например, *Ethernet IEEE* 802.3, *Fast Ethernet IEEE* 802.3*u*), а также стандарты средств сопряжения проектируемой ИС с сетями передачи данных общего назначения.

Выбор аппаратных платформ ИС связан с определением их параметров: вычислительной мощности серверов и рабочих станций в соответствии с проектными решениями по разделению функций между клиентами и серверами; степени масштабируемости аппаратных платформ; надежности. Профиль среды должен содержать стандарты, определяющие параметры технических средств и способы их измерения (например, стандартные тесты измерения производительности).

2.3. *Профиль защиты информации*

Этот профиль должен обеспечивать реализацию политики информационной безопасности, разрабатываемой в соответствии с требуемой категорией безопасности и критериями безопасности, заданными в ТЗ на систему. Построение профиля защиты информации в распределенных системах клиент-сервер методически связано с точным определением компонентов системы, ответственных за те или иные функции, службы и услуги, и средств защиты информации, встроенных в эти компоненты. Функциональная область защиты информации включает следующие функции защиты, реализуемые разными компонентами системы:

- функции, реализуемые ОС;
- функции защиты от несанкционированного доступа, реализуемые на уровне программного обеспечения промежуточного слоя;

- функции управления данными, реализуемые СУБД;

- функции защиты программных средств, включая средства защиты от вирусов;

- функции защиты информации при обмене данными в распределенных системах, включая криптографические функции;

- функции администрирования средств безопасности.

Профиль защиты информации должен включать указания по методам и средствам обнаружения в применяемых аппаратных и программных средствах недекларированных возможностей. Профиль должен также включать указания на методы и средства резервного копирования информации и восстановления информации при отказах и сбоях аппаратных систем.

2.4. *Профиль инструментальных средств*

Профиль инструментальных средств, встроенных в информационную систему, должен отражать решения по выбору методологии и технологии создания, сопровождения и развития ИС. В этом профиле должны содержаться ссылки на описания выбранных методологии и технологии, выполненное на стадии эскизного проектирования системы.

Состав инструментальных средств определяется на основании решений и нормативных документов об организации сопровождения и развития ИС. При этом должны быть учтены правила и порядок, регламентирующие внесение изменений в действующие системы. Функциональная область профиля инструментальных средств, встроенных в систему, охватывает функции централизованного управления и администрирования, связанные с:

- контролем производительности и корректности функционирования системы в целом;

- управление конфигурацией прикладного ПО, тиражированием версий;

- управление доступом пользователей к ресурсам системы и конфигурацией ресурсов;

- перенастройкой приложений в связи с изменениями прикладных функций ИС;

- настройкой пользовательских интерфейсов (генерация экранных форм и отчетов);

- ведением баз данных системы;

- восстановлением работоспособности системы после сбоев и аварий.

Дополнительные ресурсы, необходимые для функционирования встроенных инструментальных средств, также как минимальный и рекомендуемый объем оперативной памяти, размеры требуемого дискового пространства и т.п. должны быть учтены в разделе проекта, относящемся к среде ИС.

Выбор инструментальных средств, встроенных в систему, должен производиться в соответствии с требованиями профиля среды. Ссылки на соответствующие стандарты, входящие в профиль среды, должны содержаться в профиле инструментальных средств.

В этом профиле должны также содержаться ссылки на требования к средствам тестирования, которые необходимы для процессов сопровождения и развития системы и должны быть в нем встроены. В число встроенных в ИС средств тестирования должны входить средства функционального тестирования приложений, тестирования интерфейсов, системного тестирования и тестирования серверов/клиентов при максимальной нагрузке.

СИНТЕЗ СТРУКТУРЫ ИНФОРМАЦИОННОЙ СИСТЕМЫ УПРАВЛЕНИЯ

3.1. Основные характеристики структуры системы управления

Создание информационной системы управления (ИСУ) отождествляется не только с автоматизацией функций управления, но и с критическим анализом и выбором принципов управления, структуры и функций системы.

Структура – (от латинского *structure*, означает строение, расположение, порядок) отражает определенные взаимосвязи, взаиморасположение составных частей системы, ее устройство (строение).

Структура системы управления отражает строение и внутреннюю форму организации, прочные и относительно устойчивые взаимоотношения элементов системы.

Структурные связи обладают относительной независимостью от элементов и могут выступать как инвариант при переходе от одной системы к другой, перенося закономерности, выявленные и отраженные в структуре одной из них, на другие. При этом системы могут иметь различную физическую природу [7].

Одна и та же система может быть представлена разными структурами в зависимости от стадии познания объектов или процессов, от аспекта их рассмотрения, цели создания. При этом по мере развития исследований и в процессе проектирования структура системы может изменяться. Структуры могут быть представлены в матричной форме, в форме теоретико-множественных описаний, с помощью языка топологии, алгебры и других средств моделирования систем

Различают следующие виды структур: сетевые, иерархические, матричные, смешанные иерархические структуры, с произвольными связями. От вида структур зависит важная характеристика любой системы - степень ее целостности и устойчивости.

Для сравнительного анализа структур используются информационные оценки степени целостности α и коэффициента использования компонентов системы β, которые могут интерпретироваться как оценки устойчи-

вости оргструктуры или предоставлении свободы элементов или как оценки степени централизации-децентрализации управления в системе.

Эти оценки получены из соотношения, определяющего взаимосвязь системной C_c, собственной C_o и взаимной $C_в$ сложности системы:

$$C_c = C_o + C_в. \qquad (3.1)$$

Собственная сложность C_o представляет собой суммарную сложность элементов системы вне связи их между собой. Системная сложность C_c представляет содержание системы как целого. Взаимная сложность $C_в$ характеризует степень взаимосвязи элементов в системе, т.е. сложность ее устройства, схемы, структуры.

Разделив (3.1) на C_o получаем две важные сопряженные оценки:

$$\alpha = - C_в/ C_o,$$
$$\beta = C_c/ C_o, \qquad (3.2)$$

причем $\beta = 1 - \alpha$.

Первая из них характеризует степень целостности, связности, взаимозависимости элементов системы. Для организационных систем α может быть интерпретирована как характеристика устойчивости, управляемости, степени централизации управления.

Вторая β – как самостоятельность, автономность частей целого, степень использования возможностей элементов. Для организационных систем β удобно называть коэффициентом использования элементов в системе.

В случае если такие оценки не удается получить, либо реальные процессы необходимо представить иерархическими структурами типа «страт» или «эшелонов», либо большое число и разнообразие связей между компонентами системы приводит к «проклятию размерности», следует использовать полевое описание системы в пространстве ее структуры [7].

Тогда обозначив через N – «мощность» объекта управления (способность производить любую продукцию, включая информацию, в соответствии со своим назначением), и через r плотность N в каждой точке соответствующего пространства, потребуем, чтобы с учетом ограничений на пропускную способность системы управления потенциал H в каждой точке был максимален:

$$H = \frac{1}{4\pi}\int \frac{R\rho}{r} = \frac{1}{4\pi}\int \frac{RdN}{r} \to \max$$

где r – число инстанций между данной точкой и каждой остальной в пространстве управления; R – доля общего числа функций объекта, участвующих во взаимодействии с каждой точкой.

Это обеспечивает максимальную управляемость и связность (целостность) системы, а тем самым и выбор наилучшего варианта структуры системы управления.

Каждая ИС, как большая и сложная система, обладает большим числом элементов, свойств, связей между элементами, и охватить их одновременно в рамках одного понятия структуры не представляется возможным.

1. *Основные характеристики структур ИС*

Различают две группы характеристик структур ИС:

1) характеристики связаны с иерархичностью системы;

2) характеристики, оценивающие качество функционирования ИС заданной структуры.

К первой группе относятся: число уровней иерархии, число подсистем на каждом уровне иерархии, степень централизации, норма управляемости, мера равномерности связей, степень специализации подсистем, характер взаимосвязи между подсистемами и уровнями иерархии и т.д.

Ко второй группе характеристик относятся: эффективность, надежность, живучесть, гибкость структуры, быстродействие, достоверность обработки данных, загрузка технических средств и узлов и т.д.

Ряд из перечисленных характеристик имеет количественную меру, другие - качественную меру. Число уровней иерархии и число подсистем на каждом уровне характеризуют соответственно «высоту» организационной структуры и «ширину» каждого ее уровня.

Одной из важнейших характеристик организационной структуры системы является степень централизации и норма управляемости (размах контроля). Степень централизации служит мерой разделения полномочий между уровнями системы.

Для каждой пары смежных уровней (n-1), n ; n=2, 3,.. N степень централизации вычисляется по формуле:

$$\alpha_n = \frac{W_n}{W_{n-1}},$$

где W_n, W_{n-1} - число задач, решаемых на уровнях.

Объем решаемых задач может быть оценен через количество перерабатываемой информации на уровне *n*.

Тогда степень централизации всей ИС рассчитывается по формуле:

$$\alpha = \sum_{i=2}^{N} \beta_n \cdot \alpha_n \ , \qquad (3.3)$$

где β_n – весовые коэффициенты, учитывающие специфику системы.

Чем больше значение α, тем выше степень централизации системы. Смещение основной массы решений в сторону высшего уровня иерархии отражает повышение степени централизации. Это повышение отождествляется с повышением управляемости подсистем и увеличением переработки информации на верхних уровнях. Повышение степени децентрализации соответствует увеличению самостоятельности подсистемы и уменьшению информации, перерабатываемой на верхних уровнях.

Степень централизации тесно связана с нормой управляемости, характеризующей объем задач, решением которых эффективно управляет руководитель. На практике норма управляемости определяется количеством работников, подчиненных одному руководителю и для различных уровней и подсистем ИС неодинакова.

Мера равномерности вертикальных связей характеризует степень отклонения связей в анализируемой системе по сравнению с равномерной линейной иерархической структурой. В равномерной структуре каждый элемент *n*-го уровня имеет одинаковое число вертикальных связей или подчиненных ему элементов *(n – 1)* уровня.

Равномерность для *n*-го уровня R_n определяется по формуле:

$$R_n = (\prod_i a_i) / a_{cp}^{k},$$

где a_i – число связей i элемента n уровня с элементами $(n-1)$ уровня;

a_{cp} – среднее число связей; k – число элементов на n уровне.

Для многоуровневой системы мера равномерности вертикальных связей вычисляется по формуле:

$$R = \frac{1}{m-1}\sum_{n=2}^{m} R_n , \qquad (3.4)$$

где m – число уровней системы.

Для равномерных вертикальных связей R=1.

Степень специализации управляющих подсистем можно выразить отношением числа однородных задач, выполняемых подсистемой, к общему числу задач этого же типа, решаемых всей системой управления. Степень специализации i- подсистемы по управлению l-го вида оценивается по формуле:

$$\lambda_i^{(l)} = \frac{V_i^{(l)}}{\sum_i V_i^{(l)}} ,$$

где $V_i^{(l)}$- число однородных задач типа l, выполняемых подсистемой i=1,2,..k (могут быть трудозатраты на решение).

Неравномерность функциональной специализации можно определить следующим образом. Пусть имеется k структурных элементов, которые должны выполнить L функций или управленческих задач. Для выполнения l-й функции (l=1,2,..L) исходя из ее трудоемкости или других соображений, назначается k_l элементов, причем

$$\sum k_l = k \text{ и } k_l \rangle 0.$$

При ограниченном числе структурных элементов, выполняющих каждую функцию, число возможных вариантов распределения k элементов по L функциям может быть рассчитано по формуле:

$$D = \frac{k!}{k_1!k_2!\ldots \cdot k_L!} .$$

При большом k выше записанную формулу можно аппроксимировать зависимостью:

$$D = -\sum_{l=1}^{L} \frac{k_l}{k} \ln \frac{k_l}{k} .$$

Значение D максимально в случае равномерной специализации, когда каждую функцию выполняет одинаковое число структурных элементов, т.е.

$$k_1 = k_2 = \ldots k_L = k/L; \text{ при этом } D_{max} = ln\ L.$$

Неравномерность функциональной специализации F можно характеризовать отношением числа возможных вариантов распределения структурных элементов по функциям управления в исследуемой структуре (для принятых значений k_l) к D_{max}:

$$F = \frac{D}{D_{МАКС}} = -\frac{1}{\ln L}\sum_{l=1}^{L}\frac{k_l}{k}\ln\frac{k_l}{k}. \qquad (3.5)$$

Эффективность ИС во многом зависит от степени совершенства ее организации и функциональной структуры. Тогда можно говорить о некоторой эффективности структуры, обеспечивающей максимальную эффективность системы в целом. Однако связь между структурой и эффективностью системы управления чрезвычайно трудно выявить и тем более формализовать.

При синтезе структур оптимизируется одна или несколько характеристик. В качестве критериев оптимизации принимаются: минимизация затрат на создание системы, максимизация общей эффективности системы, минимизация трудоемкости управления и т.д.

3.2. Синтез организационной структуры. Методы синтеза

Различают эвристические и формализованные методы синтеза оргструктур.

К эвристическим методам синтеза относятся: методы аналогий, методы экспертных оценок, организационное моделирование и др. Эвристические методы базируются на опыте и интуиции проектировщика. Анализируя функции и структуры лучших отечественных и зарубежных аналогичных систем управления, выявляют прогрессивные решения и образцы. Их используют в синтезируемых структурах.

Организационное моделирование заключается в разработке графических и макетно-стендовых схем, схем информационных потоков, матриц распределения ответственности, обобщенных структурно-информационно-временных схем, позволяющих обобщить и наглядно представить собранную информацию о функционировании системы управления. Использование организационного моделирования позволяет установить специализацию подразделений и должностных лиц по каждой функции и отдельным управленческим задачам, соподчиненность структурных элементов системы управления и связи между ними, упорядочить информационные потоки, устранить дублирование, выявить «узкие» места и т.д.

Формализованные методы синтеза структур базируются на математических моделях и методах, используют количественные зависимости между параметрами структуры и характеристиками системы.

При построении формализованных моделей синтеза структуры ИСУ широко используются статистические методы, методы математического программирования, теория массового обслуживания, теория графов, имитационное моделирование и др.

Остановимся подробнее на формализованных методах синтеза.

На практике для синтеза организационной структуры получили распространение следующие методы:

1) Нормативный метод;
2) Использование графовых моделей;
3) Метод центральной планирующей организации;
4) Методы теории массового обслуживания и др.

1. *Нормативный метод синтеза организационных структур*

Нормативный метод основан на выявлении статистических зависимостей между параметрами характеристик структуры и факторами, влияющими на эти характеристики.

Статистические зависимости устанавливаются в результате исследования однородной группы лучших систем управления: собираются данные о численных значениях структурных параметров и факторов; с помощью корреляционного анализа выявляется степень влияния каждого фактора на структурные параметры, и отбираются наиболее существенные факторы; выводятся нормативные формулы для расчета параметров структуры. Для данной группы ИСУ полученные зависимости считаются наилучшими, их используют при проектировании организационных структур ИСУ аналогичного типа. С помощью нормативного метода разрабатываются нормативы численности по каждой из выделенных функций управления, коэффициенты распределения работников аппарата управления между управляющими и производственными подразделениями.

На основе статистического и логического анализа выделяются производственные и управленческие факторы, наиболее существенно влияющие на объем работ. Исходя из этих факторов строится зависимость вида:

$$H_j = c \prod_i x_i^{aij},$$

где H_j – параметр организационной структуры ИСУ; c – коэффициент регрессии, отражающий средний уровень значения параметра в исследуемой группе; x_i- численное значение i-го фактора;αij- показатель степени, выражающий влияние i-го фактора на j-й параметр структуры.

Формирование нормативов проходит в два этапа. На первом этапе принимают во внимание производственные факторы, на втором – определяют границы допустимых отклонений от найденных нормативов и корректируют коэффициенты c, исходя из другой группы фактов, характеризующих уровень организации управленческого труда, наличия средств оргтехники, системы документооборота и др. При этом предполагается, что производственные факторы обуславливают объем управленческих работ, а вторая группа факторов влияет на производительность управленческого труда.

2. *Синтез организационной структуры на графовых моделях*

Рассмотрим особенности использования графовых моделей для синтеза организационной структуры ИС [3]. Организационную структуру системы удобно представить в виде графа $G(E,V)$, где E – множество вершин, представляющих собой элементы структуры; V – множество дуг, указывающих связи между элементами.

Графовые модели могут обладать различной информативностью. На этих моделях в простых случаях можно ограничиться указанием структурных подразделений и связей между ними. Можно на графовых моделях указать более подробную информацию. Так, например, вершинам можно приписывать веса, указывающие численность подразделений, а дугам – мощность, характеризующую количество передаваемой информации.

Синтез организационной структуры на графовых моделях основан на принципе агрегирования, т.е. объединении в одну подсистему наиболее близких задач, или в один управляющий узел наиболее тесно взаимодействующих подразделений и исполнителей. Этот принцип базируется на интуитивно ясном и проверенном на практике соображении, что при таком объединении уменьшается объем циркулирующей информации между подразделениями; время, затрачиваемое на передачу информации и согласование плановых решений; дублирование функций и т.д., хотя в явном виде все эти характеристики не учитываются. Так, например, множество E графа G можно интерпретировать множеством различных подразделений и исполнителей управляющего органа, а множество V множеством

взаимосвязей между ними. Тогда в результате решения задачи оптимального распределения G получим множество моделей организационной структуры или подсистем, которые могут являться элементами структуры в дальнейших задачах синтеза.

Задача синтеза организационной структуры в терминах теории графов формулируется как разбиение графа G на подграфы $G_1, G_2, ..., G_N$. При этом каждый из графов $Gi \in G$, $i = 1, 2, ..., N$, подграфы не должны пересекаться $G_i \cap G_j = 0$, для $i, j = 1, 2, ..., i \neq j$, объединение должно дать исходный граф: $\bigcup_i G_i = G$.

Полученное разбиение должно минимизировать функцию:

$$\sum_{i=1}^{N} C_i(G_i) \rightarrow \min, \qquad (3.6)$$

где $C_i(G_i)$ - некоторая функция, определенная на множестве разбиений G_i, $i = 1, 2, ...N$.

В зависимости от вида целевой функции графовые задачи синтеза организационной структуры формализуются следующим образом:

1) Найти разбиение графа $G(E,V)$ на подграфы G_1 , $G_{2, ...}$ G_N при $r(E_i) \leq r$, которое минимизирует некоторую функцию от величины внешних связей между подграфами (например, сумму внешних связей, величину связей между отдельными подграфами и т.д.). Ограничение $r(E_i) \leq r$ может означать, например, допустимое количество вершин (сотрудников) в подграфе (в подразделении).

2) Найти разбиение графа $G(E,V)$ на сильно связанные подграфы, т.е. на подграфы, у которых связи между элементами внутри подграфа больше, чем с другими элементами графа $G_1, G_2, ..., G_N$.

3) Найти разбиение графа $G(E,V)$, чтобы $a(U)=0{,}5\sum c(E_i) \rightarrow min$ при $max\ l(E_i) \leq a(U)$, где $a(U)$ – суммарная внешняя связанность подграфов минимизируется, при котором максимальная внутренняя связь подграфа не должна превышать суммарной внешней связи. Обозначения: $c(E_i)$ - число связей всех вершин E_i с другими вершинами $V \backslash V_i$; $l(E_i)$ – число связей вершин графа G_i между собой.

4) Найти разбиение графа $G(E,V)$ на N непересекающихся подмножеств $\bigcup_i G_i = G$, $G_s \cap G_t = 0$ для $s, t = 1, 2, \ldots N, s \neq t$ таким образом, чтобы

$$\sum_{s,t=1}^{N} \sum_{i,j=1}^{k} d_{ij} x_{is} x_{jt} \rightarrow \min,$$

где d_{ij} – показатель степени связи между вершинами графа G;

$$x_{is} = \begin{cases} 1 - если\ E_i \in G_s \\ 0 - в \quad противном \quad случае \end{cases}.$$

3. *Синтеза организационной структуры методом центральной планирующей организации*

В основу метода центральной планирующей организации положен принцип максимальной связности задач, решаемых в каждом подразделении [8]. Задача синтеза формализуется аналогично предыдущей задаче, но для получения многоуровневой структуры поиск автономных подсистем ведется не только по горизонтали (в пределах одного иерархического уровня), но и по вертикали.

Вначале по заданному ограничению $max\ b(E_i) \leq B$ для графа $G(E,V)$ решается задача поиска оптимального разбиения u_1 для нижнего иерархического уровня с целевой функцией $a(E_i) \rightarrow min$ (из графа G выделяется подграф первого уровня по критерию минимума внешних связей и ограничении числа внутренних связей B. Для графа $G \setminus G_1 = G_{u1}$ решение повторяется и т.д., пока число внутренних связей не будет превышать B).

Затем та же задача решается вновь, но уже для графа G_{u1}, что позволяет найти такое разбиение u_2 для второго уровня, у которого $a(u_2) \rightarrow min$. И так до тех пор, пока на некотором уровне β значение $min\ a(u_\beta)$ не будет превосходить константу B. Тем самым проблема синтеза структуры сводиться к определению частных (субоптимальных) разбиений u_1, u_2, ... u_β.

4. *Использование методов теории массового обслуживания для синтеза организационной структуры*

При оптимизации структуры иерархической системы оперативного управления каждый из узлов системы рассматривается как система массового обслуживания (СМО), имеющая m входов (входящих потоков требований на обслуживание) и l выходов [3]. На вход любого узла системы в

некоторые случайные моменты времени в соответствии с заданным законом распределения поступает m потоков. Потоки могут быть либо неограниченными, либо состоять из конечного числа требований. Выходящий поток образуется из последовательности обслуженных требований различных входящих потоков и из требований, покидающих систему или очередь до окончания обслуживания.

Оптимизация проводится для однородных иерархических структур, т.е. характеристики узлов одной ступени одинаковы, и к каждому узлу подключено одинаковое для данной ступени число узлов предыдущей. В качестве общего критерия функционирования системы, характеризующего суммарные потери, принимается

$$W(m) = n_{m-1} W(m-1) + W_m \quad \textit{или}$$
$$W(m) = \sum_{i=1}^{m} W_i \prod_{j=i}^{m} n_j, \qquad (3.7)$$

где $W(m)$- величина критерия для m-ступенчатой системы; $W(m\text{-}1)$- соответствующие потери для составляющих ее подсистем $(m\text{-}1)$-го порядка (всего таких подсистем $n_{m\text{-}1}$); W_m – потери в системе обслуживания последней ступени; W_i - потери в системе обслуживания одного узла i-й ступени; n_j – число узлов j-й ступени, подключаемых к одному узлу $(j+1)$-й ступени.

Оптимизация заключается в нахождении таких значений n_i^*, i=1, 2, ..$(m\text{-}1)$, n_m=1 , при которых $W(m)$ минимально; где m - число ступеней.

В критерии первая формула выражает суммарные потери через потери в однородных подсистемах, их число $n_{m\text{-}1}$, потери в каждой подсистеме $W(m\text{-}1)$. Во второй формуле потери вычисляются через потери в узлах W_i , количество узлов n_j, подключаемых к одному узлу $(j+1)$-й ступени.

3.3. Синтез функциональной структуры

Синтез функциональной структуры ИС включает в себя распределение решаемых задач (операций управления) по подсистемам и уровням организационной структуры. Одним из наиболее распространенных количественных критериев объединения в подсистемы решаемых организацией задач связан с понятием «близости» решаемых задач и выполняемых операций. «Близость» может выражаться в том, что решение одной из них невозможно без решения другой. Задачи могут считаться близкими вследст-

вие принадлежности к одной и той же теме или в связи с использованием при выполнении одних и тех же ресурсов. Близость может оцениваться также по величине потока информации, которой обмениваются подразделения или по степени изменения взаимосвязей по времени.

Объединение в одну подсистему наиболее близких задач облегчает управление ходом решения задач внутри самих подсистем, так и координацию деятельности подсистем в целом. Если в подсистему объединяются наиболее связанные задачи, то объем информации, которыми обмениваются подсистемы, значительно сокращается. Благодаря этому уменьшается время, затрачиваемое на обмен информацией, упрощается координация деятельности подсистем.

При анализе процессов управления в организации важно понятие элементарной операции. В зависимости от уровня рассматривания элементарными операциями могут быть в ИСУ операции производства, сбыта, финансирования, обработки информации, принятия решений и т.п.

В общем случае элементарную операцию можно записать как преобразование входного вектора $X_i = (x_{1i}, x_{2i}, \ldots, x_{mi})$ в выходной вектор $Y_i = (y_{1i}, y_{2i}, \ldots, y_{ni})$. В общем случае:

$$Y_i = f_i(x_i), \tag{3.8}$$

где f_i – функция преобразования.

Если преобразование является линейным, то (3.8) можно записать в матричном виде:

$$Y_i = \boldsymbol{A_i x_i}\ , \tag{3.9}$$

где A_i – матрица преобразований.

Элементарные операции связаны между собой. Связь операций удобно представить графом. $\Gamma^1(E, H^1)$ без петель, множество вершин которого соответствует операциям $e_1, e_2, \ldots e_n$, а каждая дуга $h'_{ij} \rightarrow H^1$ указывает на то, что выход операции e_i является входом операции e_j. Как правило, преобразование (3.8) связано с затратами ресурсов (вычислительных ресурсов, денежных средств, сырья и т.п.). Тогда кроме логических связей между операциями e_i необходимо учесть связи, обусловленные наличием ограничений типа:

$$\varphi_k(f_i(e_i)) \le u_k;\ e_i \in E_k;\ k = 1,2,..m\ , \tag{3.10}$$

где u_k – количество ресурсов k-го типа; $E_k \in E$ - подмножество операций, занятых выполнением k-й функции или потребляющих ресурсы k-го типа; φ_k – функция затрат k-го вида ресурса.

Связи между операциями, возникающие при наличии ограничений (3.10) называются ресурсными или функциональными. Эти связи можно изобразить графически, построив ресурсный граф $\Gamma^2 = (E \cup V, H^2)$, в котором множество вершин $V=V_1,V_2,...V_m$ представляет собой источники ресурсов (могут быть фиктивными). Каждая дуга h^2_{kj} показывает, что для операции e_j требуются ресурсы k-го типа V_k .

Чтобы получить полную картину взаимосвязи операций, необходимо построить объединенный граф $\Gamma = \Gamma^1 \cup \Gamma^2 = (E \cup V, H^1 \cup H^2)$, полученный из графа Γ^1 добавлением ресурсных вершин и дуг графа Γ^2, который назовем графом взаимосвязей операций.

Задача синтеза функциональной структуры состоит в разбиении множества операций E на N независимых подмножеств:

$$E_1, E_2, \ldots, E_N, \tag{3.11}$$

где $E_i \in E$ и $E_i \cap E_j = 0$ для $i, j = 1, 2,\ldots N, \quad i \neq j, \quad u \quad \bigcup_{i=1}^{i=N} E_i = E$.

Задача состоит в том, чтобы получить разбиение, которое минимизирует функцию:

$$\sum_{i=1}^{N} C(E_i) \to \min, \tag{3.12}$$

где $C(E_i)$ – функция, определенная на множестве разбиений E_i , $i = 1, 2,..N$

Задача синтеза структуры системы состоит в разбиении E на подмножества $E_1, E_2,...,E_N$ при различных критериях разбиения системы на подсистемы. При этом близость операции e_i к e_j будем характеризовать величиной m_{ij}, где m_{ij} – значение потока по дуге h_{ij} графа Γ, измеряемое объемом информации или количеством связей.

Можно выделить два принципа декомпозиции:

1) разложение переменных;

2) разложение ограничений.

Если дуга (e_i, V_k) разрывается и $e_i \in E_i, V_k \in E_k, E_i \cap E_k = \text{O}$, то в первом случае (разложение переменных) вместо вершины e_i вводятся две несвязанными между собой: вершина e_i' и e_i''.

При этом вершина e_i' остается связанной только с вершинами первой подсистемы, а вершина e_i'' – с вершинами второй подсистемой. Это преобразование соответствует тому, что в целевой функции и ограничениях первой подсистемы x_i заменяется на x_i', а во второй на x_i''. Благодаря этому модели подсистем становятся формально независимыми и необходимым (а иногда достаточным) условием согласования является выполнение равенства:

$$x' = x''.$$

Разложение систем на несколько относительно автономных подсистем приводит к необходимости создания высшего координирующего органа. Формальными способами воздействия координирующего органа могут служить: плата за взаимодействие, фиксирование взаимодействий, оценки и предсказания взаимодействий.

В методе разложения ограничений вместо вершины V_k вводится две вершины: V_k' и V_k'', первая из которых связана только с первой подсистемой, а вторая – только со второй подсистемой. Это соответствует тому, что вместо одного ограничения

$$\varphi_{k1}(x_i) + \varphi_{k2}(x_i) \le U_k$$

на ресурсы типа k вводят два несвязанных ограничения

$$\varphi_{k1}(x_i) \le U_k' \quad и \quad \varphi_{л2}(x_i) \le U_k'',$$

где U_k, U_k', U_k'' – количество ресурсов соответственно V_k, V_k' и V_k''.

Здесь необходимо выполнение условия $U_k = U_k' + U_k''$.

Формальным способом воздействия координирующего органа на подсистемы в этом случае может служить распределение ресурсов и плата за ресурсы.

Таким образом, после решения задачи разбиения системы на подсистемы с помощью двух рассмотренных принципов декомпозиции проводится разрыв связей и выбор способов координации.

Не уменьшая общности дальнейших рассуждений, рассмотрим матричную модель системы, полученную из Γ^2, и описываемую системой линейных ограничений:

$$\boldsymbol{A}x \leq u; \quad x \geq 0, \tag{3.13}$$

где $\boldsymbol{A}$ – матрица; x и u векторы столбцы с критерием

$$cx \rightarrow max; \tag{3.14}$$

здесь c – вектор строка.

Рассмотрим квазиблочную матрицу:

$$\begin{vmatrix} A_1 A_i \\ a_{ki} A_k \\ A_2 \end{vmatrix} * \begin{vmatrix} x_1 \\ x_i \\ x_{2_1} \end{vmatrix} \leq \begin{vmatrix} u_1 \\ u_k \\ u_2 \end{vmatrix}, \tag{3.15}$$

где $\boldsymbol{A}_1$ и $\boldsymbol{A}_2$ подматрицы матрицы $\boldsymbol{A}$; $\boldsymbol{x_1}$, $\boldsymbol{x_2}$, u_1, u_2 – вектор столбцы; $\boldsymbol{A_к}$ – вектор строка.

Для того, чтобы матрица $\boldsymbol{A}$ распалась на две независимые подматрицы (подсистемы), можно воспользоваться одним из методов декомпозиции, рассмотренных выше. Если целесообразно объединить подразделение минимизируемой структуры в соответствии с типом используемых им ресурсов (т.е. по функциональному признаку), то применяется разложение переменных x_i на $x_i^{/}$ и $x_i^{//}$. Получаются две независимые подсистемы: первая оптимизирует x_1 и $x_i^{/}$ при ограничениях:

$$\left|A_1 A_i\right| * \begin{vmatrix} x_1 \\ x_i^{/} \end{vmatrix} \leq u_1, \tag{3.16}$$

вторая – x_2 и $x_i^{//}$ при ограничениях:

$$\begin{vmatrix} a_k A_k \\ 0 A_2 \end{vmatrix} * \begin{vmatrix} x_i^{//} \\ x_2 \end{vmatrix} \leq \begin{vmatrix} u_k \\ u_2 \end{vmatrix}. \tag{3.17}$$

Координирующий орган воздействует на подсистемы таким образом, чтобы

$$x_i^{/} - x_i^{//} \rightarrow 0. \tag{3.18}$$

В общем случае производится разложение графа Γ по переменным u_1 и u_2.

Возможно другое разделение матрицы, при котором подразделения объединяют в соответствии с содержанием работ (по тематическому признаку). В этом случае образуются независимые подсистемы: первая оптимизирует x_1, x_i при ограничениях

$$\begin{vmatrix} A_1 A_i \\ 0 \; a_{ki} \end{vmatrix} * \begin{vmatrix} x_1 \\ x_i \end{vmatrix} \le \begin{vmatrix} u_1 \\ u_k^{/} \end{vmatrix} . \qquad (3.19)$$

Вторая – x_2 при ограничениях

$$\begin{vmatrix} A_k \\ A_2 \end{vmatrix} x_2 \le \begin{vmatrix} u_k^{//} \\ u_2 \end{vmatrix} . \qquad (3.20)$$

Координирующий орган регулирует значения $u_k^{/}$, $u_k^{//}$ таким образом, чтобы решения, полученные подсистемами, были оптимальны для организации в целом, и выполнялось условие:

$$u_k^{/} + u_k^{//} = u_k.$$

В общем случае проводится разложение графа Γ по ресурсным вершинам.

Иллюстрацией модели системы (3.13), (3.14) может служить управляющий орган, выполняющий функцию планирования, формализуемую в виде задачи линейного программирования, имеющую матрицу условий рассматриваемого вида, например задач планирования выпуска продукции предприятием.

При этом обозначения интерпретируются следующим образом: вектор x – выпуск продукции (x_i – выпуск продукции i-го вида); $\boldsymbol{A}$ - матрица затрат сырья (a_{ki} – затраты сырья k-го вида на выпуск единицы продукции i-го вида); c - вектор стоимостей или прибыли; u – вектор наличия сырья.

Особенность матрицы (два блока с зацеплением по i-му виду продукции) позволяют провести декомпозицию задачи на две задачи линейного программирования, которые соответствуют двум элементам организационной структуры (плановые отделы) по двум группам продукции и координирующего органа. После разбиения множества функций ИСУ на подсистемы и задачи возникает необходимость распределения задач по узлам и уровням организационной структуры ИСУ.

При определении оптимального распределения функций по узлам ИСУ исходными являются:

1) Выполняемые системой функции, формализованные в виде множества решаемых задач, каждая из которых состоит из ряда этапов.

2) Связи между задачами и их этапами.

3) Множество возможных узлов ИСУ и связей между ними.

4) Виды и характеристики технических средств, применение которых возможно в проектируемой системе.

5) Внешние для системы источники и потребности информации по всем задачам и их этапам.

Задача выбора оптимальной структуры ставится как нелинейная задача математического программирования [8]:

$$F_0(x_{ijkl}) \rightarrow extr;$$

$$F_n(x_{ijkl}) \leq B_n \quad n=1, 2, ..m;$$

$$\sum_{jkl} x_{ijkl} = 1 \quad (i=1, 2, ..n). \qquad (3.21)$$

x_{ijkl} – булева переменная, равная единице, если функциональная задача i решается в j-м узле системы с помощью алгоритма k на l-м наборе технических средств, и равна нулю в противном случае.

Эффективность выбранной структуры должна базироваться на обоснованном соотношении величины выгод (затрат) и сроков их получения. Эффект от внедрения ИСУ складывается из различных факторов, одни из которых представляют интерес для организации, непосредственно внедряющей систему, другие имеют общее народнохозяйственное значение (для региона, государства), и результаты их реализации на деятельность организации отражаются косвенным образом.

Модель позволяет учитывать такие характеристики эффективности вариантов структуры системы, как стоимость, оперативность, надежность и др.

Разработанные методы дают решение задачи синтеза структуры ИСУ в случае аддитивных мультипликативных функций. Оптимальную структуру определяют при ограничениях на ресурсы, загрузку технических средств (ТС), на своевременное решение задач и их этапов и т.д.

3.4. Синтез структуры с учетом затрат на обмен информацией

Рассмотрим постановку задачи синтеза структуры с учетом затрат на обмен информацией между задачами, решаемыми на различных уровнях, и затрат на эксплуатацию системы.

Задано множество возможных алгоритмов $M_i=(k(k=1,2,..k_i))$ решения i-й задачи ($i=1,2,..I$) в ИСУ, включая решение задач без применения ЭВМ; матрица связи между задачами $\|a_{ii*}\|$. Задачи i и i^* считаются связанными, если для решения i^*-й задачи используется информация, являющаяся выходной для i-й задачи, при этом элемент матрицы a_{ii*} имеет смысл среднего потока информации от i-й задачи к i^*- й. Если задачи не связаны, то $a_{ii*}=0$; $Q=(j(j=1,2,..J))$ – множество узлов в ИСУ; $\|\gamma_{jj*}\|$- матрица удельных затрат на передачу информации из j-го в j*-й узел. Для несвязанных узлов $\gamma_{jj*}=\infty$ и $\gamma_{jj}=0$; $F=(l(l=1,2,..L))$ – множество технических средств; m_l – величина, характеризующая ресурсы l-го технического средства; a_{ikj} – эффективность решения i–й задачи, решаемой k–м способом в j-м узле; m_{ik} – потребность в ресурсах технических средств i-й задачи, решаемой k-м способом (например, в машинном времени, объеме памяти и др.); c_{lj} – затраты на эксплуатацию l–го технического средства в j-м узле; k_l – капитальные затраты на технические средства; k_{ik} – затраты на разработку и внедрение i-й задачи в k-м варианте.

Задача реализуется следующим образом:

$$\max_{x_{ijk},x_{lj}}\left[\sum_{ikj,i^*j^*k^*} b_{ikj,i^*k^*j^*}x_{ikj}x_{i^*k^*j^*}-\sum_{l,j}c_{lj}x_{lj}\right],$$

$$\text{где}\quad b_{ikj,i^*k^*j^*}=\begin{cases} a_{ikj}, & \text{если}\quad ikj=i^*k^*j^*; \\ a_{iki^*k^*}\gamma_{jj^*}, & \text{если}\quad ikj\neq i^*k^*j^*; \end{cases} \qquad (3.22)$$

$x_{ikj}=1$ - если i-я задача решается в j-м узле k-м способом и равно 0 в противном случае;

$x_{lj}=1$ – если j-й узел оборудуется l-м техническим средством, и равно 0 в противном случае;

при ограничениях:

$$\sum_{k,\,j} x_{ikj} = 1 \quad (i = 1, 2, \ldots J);$$

$$\sum_{l,\,j} k_l x_{lj} + \sum_{i,k,\,j} k_{ik} x_{ikj} \prec k; \tag{3.23}$$

$$\sum_{i,k} m_{ik} x_{ikj} \prec \sum_{l} m_l x_{lj} \quad (j = 1,2,..J). \tag{3. 24}$$

Величина критерия (3.22) имеет смысл общей эффективности разрабатываемой ИСУ, равной эффективности решения задач управления в узлах системы за вычетом эксплуатационных затрат на функционирование системы. В формуле (3.22) величина $b_{ikji*k*j*}$ равна эффективности решения k-го варианта i-й задачи в j-м узле (a_{ikj}), если $ikj=i*k*j*$, и равна затратам на передачу информации из j-го узла в j*-й узел в противном случае.

Первое ограничение (3.23) не допускает решение i-й задачи в различных узлах. Второе ограничение учитывает то факт, что ресурсы на разработку не должны превышать заданной величины k. Третье ограничение учитывает потребность в ресурсах технических средств, которые не должны превышать ресурсные возможности выбираемых технических средств.

Приведенная формализация задачи синтеза позволила создать общую модель определения оптимальной структуры ИС. Нахождение оптимальной структуры является сложной задачей, поэтому в практических случаях ограничиваются нахождением рациональной структуры системы.

Методика определения рациональной структуры основана на поэтапном решении частных задач. Для решения задачи синтеза структуры управляющих систем реального времени используется итерационная процедура. На первом этапе определяются перечень возлагаемых на технические средства ИСУ задач и их распределение по узлам системы с учетом ограничений на ресурсы. На следующем этапе выбирают технические средства. Находят характеристики динамики функционирования узлов. Если необходимо, то осуществляют корректировку временных ограничений первого этапа и цикл повторяется. Эту задачу решают на этапе эскизного проектирования, а на этапе технического проектирования это решение уточняется.

Рационально построенная ИС имеет иерархическую структуру. Узлы системы одного уровня можно разбить на группы, в которых они идентичны. Это позволяет рассматривать работу лишь одного «типового» узла для каждой из групп одного уровня [8].

Критерий качества функционирования для таких систем обычно аддитивен и имеет вид:

$$W = \sum_{r_m=1}^{R_m} \sum_{m=1}^{M} \sum_{j=1}^{n} W_{jr,m} \ , \qquad (3.25)$$

где $W_{jr,m}$ – величина критерия для j–го узла m–го уровня r–й группы;
r_m=1,2,…R_m – число групп подсистем m-го уровня системы;
m=(1,2,…M) – число уровней системы.

Если узлы системы одного уровня иерархии идентичны, то критерий качества функционирования имеет вид:

$$W(m) = n_{m-1}W(m\text{-}1)+W_m, \qquad (3.26)$$

где $W(m)$ – величина критерия для всех m уровней системы; $W(m\text{-}1)$ – величина соответствующего критерия для подсистемы (m-1)-го уровня (всего таких подсистем n_{m-1}); W_m – величина критерия для верхней ступени.

Решение задачи в этом случае существенно упрощается, причем для некоторых систем коэффициенты ветвления могут быть искомыми величинами.

В отдельных случаях решают частные задачи синтеза оптимальной структуры, такие, как определение оптимального распределения возлагаемых на ИСУ функций по заданным уровням и узлам системы, определение оптимальных вариантов реализации функций в ИСУ, выбор комплекса технических средств (КТС), обеспечивающего качественную реализацию функций и т.д. В частности, если каждая из функций может быть реализована лишь в одном узле, то для оптимального их распределения можно использовать метод «ветвей и границ».

Если для каждой из функций ИСУ задана совокупность различных вариантов их реализации и соответственно различные комплексы технических средств, то для выбора минимального по стоимости варианта реализации функций и КТС используют алгоритм, основанный на построении

графа на множестве частных графов для отдельных функций с последующим отысканием в этом графе пути минимальной длины.

Когда задано множество решаемых задач, множество узлов системы и связей между ними и распределение задач по узлам системы, возникает задача выбора КТС, обеспечивающего функционирование ИСУ по заданным критериям качества. Оценку и выбор КТС ИСУ осуществляют таким образом: определяют потенциальные возможности средств, описывают их загрузку, формулируют критерии и ограничения для оценки и выбора оборудования, проводят описание и анализ работы ЭВМ в реальных условиях. На основании данной информации проводят выбор КТС.

Для определения потенциальных возможностей ЭВМ используют оценки, основанные на сопоставлении технических параметров, позволяющие получить общее представление об ЭВМ; оценки на основе системы команд; оценки ЭВМ с использованием типовых работ, которые формируются на основе анализа конкретных работ, возлагаемых на ЭВМ в ИСУ, и, наконец, оценки, основанные на моделировании работ ЭВМ. Такой подход является наилучшим, однако он затруднителен из-за сложности системы и недостаточности исходной информации.

Выбор технических средств для больших ИС (акционерных обществ, крупных корпораций, холдингов и др.) проводят на основе анализа временной загрузочной диаграммы, причем отдельные технические средства могут интерпретироваться как различные системы массового обслуживания.

Основными исходными данными при применении этой методики, кроме перечисленных выше, являются перечень и характеристики средств, применение которых возможно в ИСУ, основные характеристики задач и ограничения на время их решения. Последовательность выбора состоит в качественном отборе технических средств, исследовании входящих потоков задач, определении временных характеристик их решения, построении временной загрузочной диаграммы для каждого устройства ИСУ, определении затрат на технические средства. По результатам этих работ выбирают комплекс аппаратуры, удовлетворяющий ограничениям и обеспечивающий минимум затрат.

Для анализа динамики функционирования распределенных (многоступенчатых) систем и оценки характеристик, таких, как загрузка, задержки в управлении, влияние ненадежности технических устройств и алгоритм функционирования систем, удобно формализовать системы в виде сетей

массового обслуживания соответствующей структуры и использовать аналитические методы для определения оценок или метод статистических испытаний в более сложных случаях. При этом определяют такие параметры ИСУ, как надежность, число уровней иерархии системы и взаимосвязи между ними, методы оптимизации работы вычислительных систем узла, вопросы гибкости, т.е. способности к перестройке, живучесть и др. характеристики.

МЕТОДОЛОГИЯ РАСЧЕТА ПОТРЕБНОСТЕЙ ПРОГРАММНО-ТЕХНИЧЕСКОГО ОБЕСПЕЧЕНИЯ ИНТЕГРИРОВАННЫХ ИНФОРМАЦИОННЫХ СИСТЕМ

4.1. Структура программно-технических средств интегрированной информационной системы организационного управления предприятия

1. *Количественные методы обоснования выбора программно-технического обеспечения ИИС*

В основу разработки методологии обоснования выбора программно-технического обеспечения (ПТО) заложены три метода [2]:

- метод динамического функционального анализа на основе сетей Петри различного вида;
- метод функционального стоимостного анализа *ABC*;
- метод транзакционного анализа конфигурационных карт.

Каждый из этих методов требует вычисления оценок:

- построение статической функциональной модели с использованием *SADT*;
- расширение статической модели соответственно поведенческими или стоимостными характеристиками его объектов;
- сбор и ввод в модель необходимой фактической информации;
- использование модели и получение соответствующих оценок.

Сети Петри представляют собой ориентированный граф с вершинами двух типов (позиция и переход), в котором дугами могут соединяться только вершины различных типов. Переход срабатывает, если условие выполняется. Фактически сеть Петри декомпозирует систему на активные (переходы) и пассивные (позиции) элементы.

Динамическое моделирование с использованием сети Петри осуществляется на основании статистической функциональной и информационной модели. Соответствующие инструментальные средства, например, *Design/CPN* для *SADT*, осуществляют автоматическое преобразование моделей в прообразы сетей Петри, которые затем дорабатываются вручную.

С использованием динамической модели описываются и анализируются:

- механизм взаимодействия процессов (последовательность, параллелизм, альтернативы);

- временные отношения между выполнением процессов (одновременность, наложение, поглощение, одинаковое время запуска/завершения и др.);

- абсолютные времена (длительность процесса, время запуска, зависимости от времени выполнения процесса и др.);

- управление исключительными ситуациями.

Построение динамической модели позволяет осуществить следующие операции:

- статический анализ системы (компоненты сети, иерархия сети, соответствие типов);

- динамический анализ системы;

- имитационное моделирование системы с построением графиков в системном времени, определяемым моментом срабатывания переходов, и в реальном времени путем задания задержек, отображающих продолжительность реальных процессов.

Метод функционально-стоимостного анализа ABC (*Activity Based Costing*) был разработан как операционно-ориентированная альтернатива традиционным подходам, основанным на использовании прямых затрат стоимости материалов (ресурсов). *ABC*-метод рассматривает любую деятельность как последовательность выполняемых процессов и функций.

Определение стоимости производится в два этапа:

- определение затрат на выполнение функций (процессов);

- определение затрат на стоимостные объекты (ресурсы).

Фактическая *ABC*-модель содержит:

- модуль ресурсов;

- модуль функций (процессов);

- модуль стоимостных объектов.

Разработка *ABC*-модели выполняется этапами:

- выявление требуемых ресурсов;

- выявление стоимостных объектов;

- определение функций;

- определение факторов ресурсов;

- определение стоимости функций.

Метод транзакционного анализа и конфигурационных карт обеспечивает переход от модели требований к модели реализации.

Для реализации этого подхода необходимо:

- разбиение иерархии диаграмм потоков способами транзакционного анализа;

- конвертирование каждой единицы в конфигурационную карту также средствами транзакционного анализа;

- соединение (суммирование) разделенных единиц в полную модель реализации.

Транзакционные типы могут быть идентифицированы как объекты:

- функциональный процесс;

- системный (рабочий) процесс;

- диалог-шаг в системе.

Транзакционный анализ является методом идентификации типов транзакций системы для их дальнейшего преобразования в качестве единицы проектирования. Транзакционный анализ позволяет также преобразовать объекты и системы в структурные карты, в нашем случае конфигурационные карты.

Вышеописанные методы и средства положены в основу методологии расчета потребности средств программно-технического обеспечения (ПТО) ИИС.

2. *Структура программно-технических средств (ПТС) ИИС*

Структура ПТС определяется при создании концептуальной (целевой) модели предприятия. На этапе расчета потребности ПТС уже определены:

- перечень функций и объемы функциональных программ;

- количество и распределение пользователей по выполняемым функциям;

- топология распределения рабочих мест и серверной группы;

- соотношение активных пользователей и общего числа пользователей;

- среднее время реакции системы (необходимое);

- относительная нагрузка фоновых задач на систему.

Эти данные являются исходными для расчета потребности в ресурсах программно-технического обеспечения. Результатом расчетов является:

- структура, объемные и временные характеристики серверной группы;

- пропускная способность и типы сетевого оборудования;

- мощности персональных компьютеров рабочих мест системы;
- среднее время реакции системы (расчетное);
- стоимость в условных единицах.

Современные ИИС ориентированы на трехуровневую архитектуру, в которой концепция клиент/сервер является одним из центральных элементов построения системы.

В рамках архитектуры клиент/сервер важная роль отводится компьютерам, выполняющим функции серверов.

На крупных предприятиях и корпорациях все больше и больше применяется пакет *R*3 фирмы *SAP* (Германия). Поэтому в качестве исходного пакета для расчета потребностей ПТО будем рассматривать пакет *R*3.

На рисунке 4.1 показана структурная схема корпоративной ИСУ. На верхнем уровне расположены серверы баз данных, далее - сервера приложений и замыкают рабочие станции пользователей.

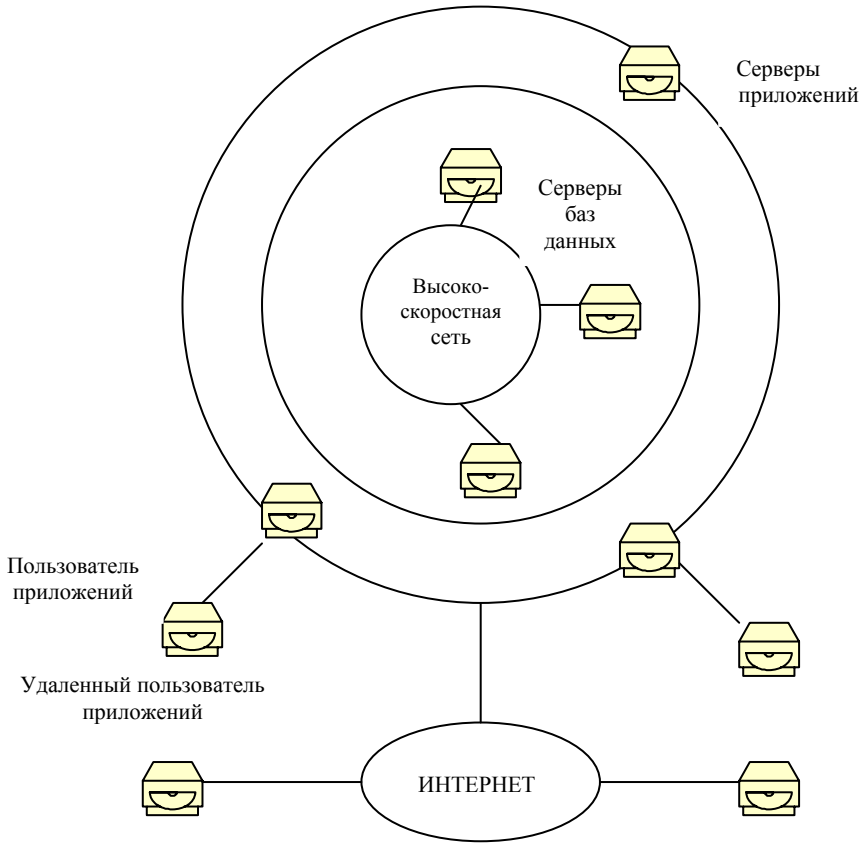


Рис. 4.1. Многоуровневая корпоративная информационная система

Каждую информационную систему можно разложить на три сферы задач: представление данных, логику приложений и хранение данных.

При использовании возможностей распределенной обработки в системе *R*3 согласно архитектуре клиент/сервер, компоненты системы размещаются на трех уровнях иерархии вычислительной системы (рис.4.2).

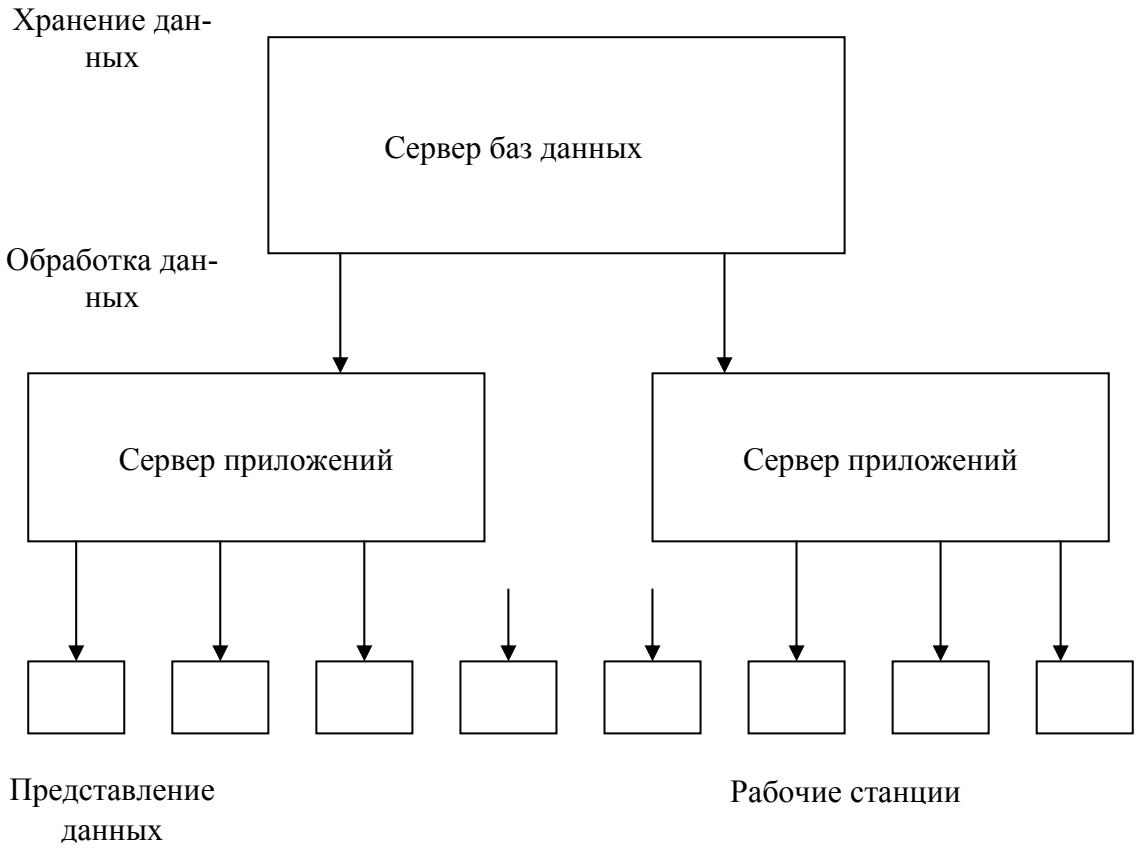


Рис.4.2. Три уровня информационных задач

Сервер баз данных выполняет процессы, представляющие услуги БД. Дополнительно на этом же сервере базируются услуги по обновлению данных, т.е. сервер отвечает также за проведенные изменения в базе данных. К серверу баз данных могут быть подключены несколько серверов приложений. К каждому серверу приложений может быть подключено множество рабочих станций, на которых работают конечные пользователи.

Инсталляция дополнительных серверов при возникновении узких мест, параллельно работающие серверы с гомогенной (однородной) нагрузкой, буферизация программ и данных вблизи исполнительных процес-

сов, балансировка нагрузки по входам и выходам значительно повышают эффективность и устойчивость такой архитектуры системы *R*3.

В архитектуре клиент/сервер важная роль отводится компьютерам, выполняющим функции сервера. В качестве серверов выбираются высокопроизводительные компьютеры, например фирмы *IBM* серии *RS*6000. Их производительность в десятки и сотни раз превышает возможности персональных компьютеров. Графические мониторы высокого разрешения, тщательно продуманная система интерфейса с пользователем, обязательные сетевые средства, большой объем оперативной памяти и колоссальный объем дискового пространства, расширенные возможности по наращиванию сервера, применение многозадачной системы *AIX* позволяют успешно использовать такие ЭВМ в качестве серверов в вычислительных сетях предприятия.

RISC-технология обеспечивает конвеерность обработки инструкций, использует расширенные файлы регистров, имеет более мощную сверхоперативную память.

Для технического оснащения рабочих мест предлагаются персональные компьютеры современных моделей, например, фирмы *IBM*. Тем самым обеспечивается техническая платформа с возможностью последующего расширения и наращивания мощности системы.

С целью интеграции данных в рамках системы *R*3 используется общая база данных. Для крупных предприятий, особенно концернов, обосновано использование СУБД *Oracle*, позволяющую обрабатывать огромные массивы данных и имеющую высокую производительность.

С целью обеспечения переносимости приложений *R*3 интерфейсы к системному программному обеспечению объединены на одном изолированном уровне, называемом Базисным уровнем (модулем). Функциональность всех компонент, лежащих выше этого уровня, не зависит от аппаратной и программной среды их функционирования. Система *R*3 обладает высокой степенью переносимости, способна работать в полном объеме на всех основных *UNIX* – платформах (*IBM AIX*, *HP-UNIX*, т.п.), а также на платформах *Microsoft Windows NT* и *IBM AS*/400.

Допускается использование различных систем управления БД, таких как *Informix Online, Oracle, ADABAS D, DB*2/6000 фирмы *IBM* и *SQL Server* фирмы *Microsof*t. Графический интерфейс пользователя системы *R*3 (*SAPGUI*) может функционировать на различных настольных системах, таких как *OS*/2 *IBM*, *OSF/Motif*, *Macintosh* и *Widows*.

Системная архитектура *SAP R*/3 приведена на рисунке 4.3.

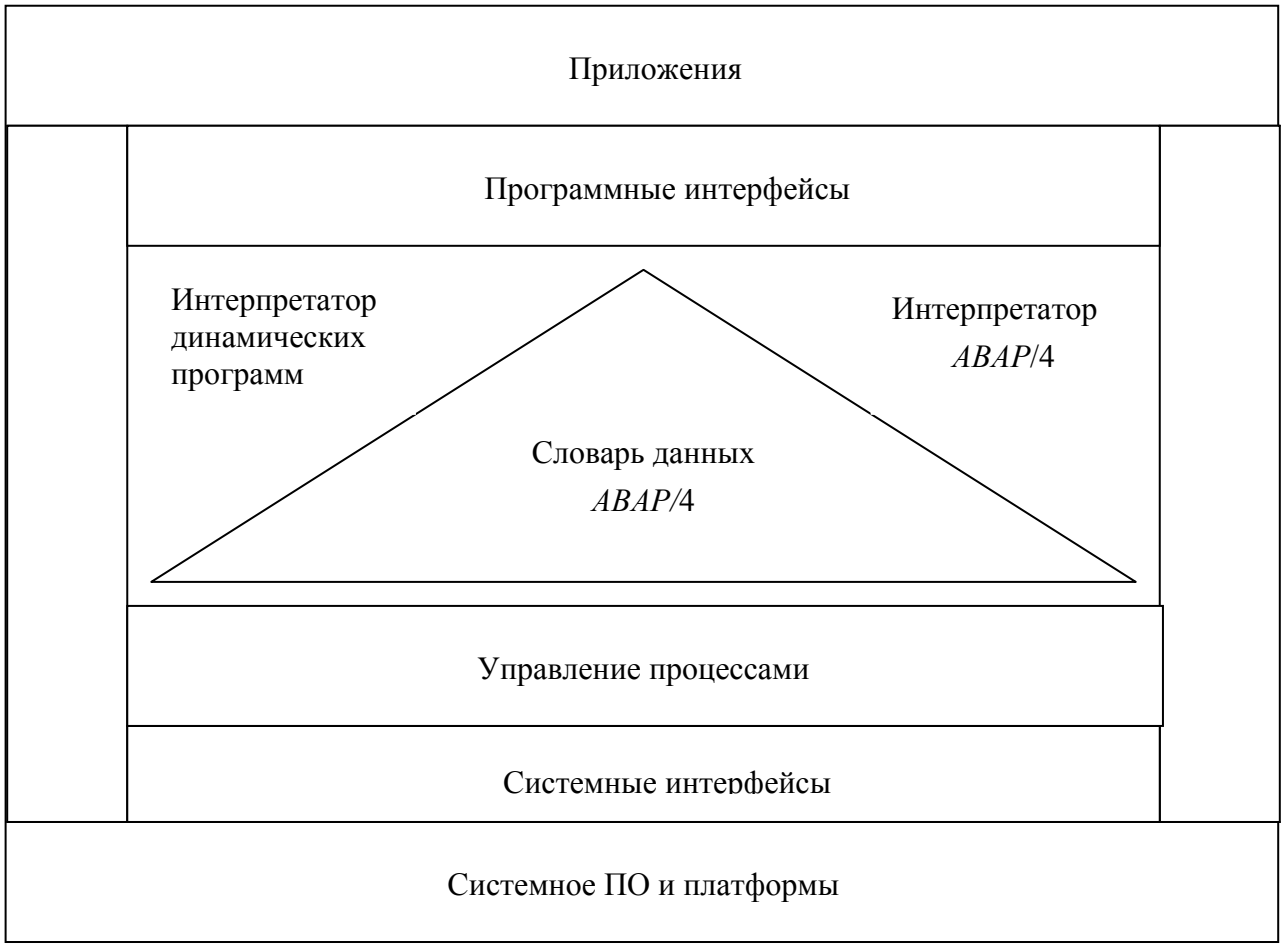


Рис. 4.3. Системная архитектура SAP R/3

Все прикладные программы в системе *R*3 написаны на языке *ABAP*/4, созданном фирмой *SAP*. Экранные программы, или ДинПро (динамические программы), являются управляющими компонентами обработки диалогов. Интерпретаторы ДинПро и *ABAP*/4, составляют программную базу приложений *R*3. В своей работе они используют структуру данных, хранимые в Словаре данных. Эти компоненты находятся вне ядра базисного модуля, что обеспечивает их независимость от операционной системы.

Базисный модуль *R*/3 позволяет использовать приложения на разных платформах, обеспечивая при этом высокую производительность и настройку на индивидуальные требования пользователей. Базисное программное обеспечение также позволяет максимально интегрировать приложения в системной среде, распределять ресурсы и компоненты системы.

Базисное ПО содержит инструменты администрирования всей системы, определяет стабильную архитектурную основу для усовершенствования системы, предоставляет интерфейсы к программным продуктам, внешним по отношению к *SAP*.

Основными компонентами сервера приложений *R*/3 являются:

- диспетчер;
- конфигурируемое число рабочих процессов;
- основной буфер для совместного использования.

Диспетчером является специализированный процесс *R*/3 операционной системы, управляющий ресурсами системы для приложений *R*/3. К основным задачам диспетчера относятся (рис. 4.4):

- равномерное распределение нагрузки между рабочими процессами;
- интерфейс с уровнем представления данных;
- организация процессов коммуникации.

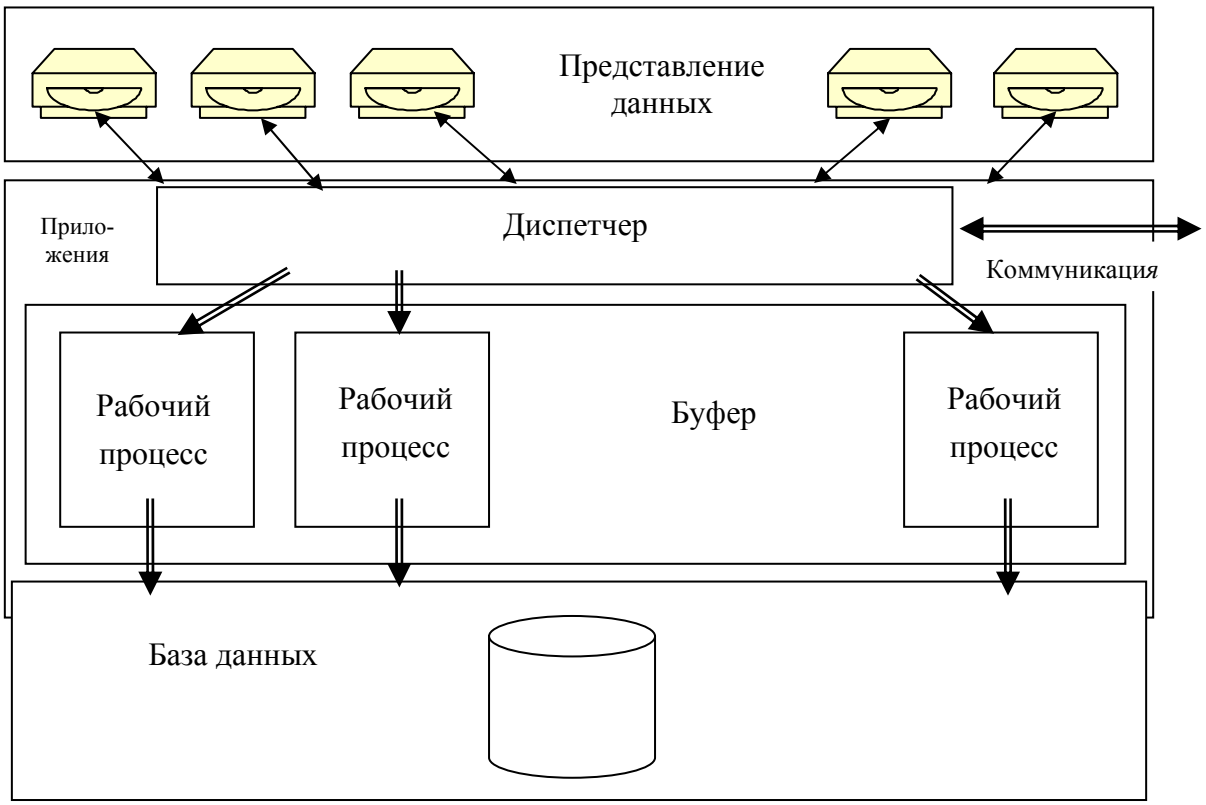


Рис.4.4. Место диспетчера в базовом модуле R/3

При запуске диспетчер считывает параметры запуска системы из специализированных файлов, в которых содержится информация как для всей системы в целом, так и для каждого компонента системы, создает и

инициализирует семафоры, создает области прокрутки, запускает рабочие процессы и устанавливает связь с сервером сообщений.

Для отдельных прикладных сервисов могут быть инсталлированы отдельные серверы приложений (инсталляций *R*/3).

Инсталляцией *R*/3 называется обособленная единица, объединяющая компоненты системы *R*/3, которая оказывает одну или несколько услуг (прикладных процессов). Все прикладные процессы, входящие в одну инсталляцию, запускаются и останавливаются одновременно. Каждая инсталляция имеет свой диспетчер и свои собственные *SAP*-буферные области. Центральная система *R*/3 состоит из единственной инсталляции, обеспечивающей все необходимые *SAP*- услуги.

Центральная система может быть легко расширена и превращена в распределенную систему путем присоединения других инстанций, предоставляющих определенные услуги. На рисунке 4.5 показана организация дополнительного сервера фоновой обработки данных (А), диалогового сервера (Б) и сервера коммуникаций (В).

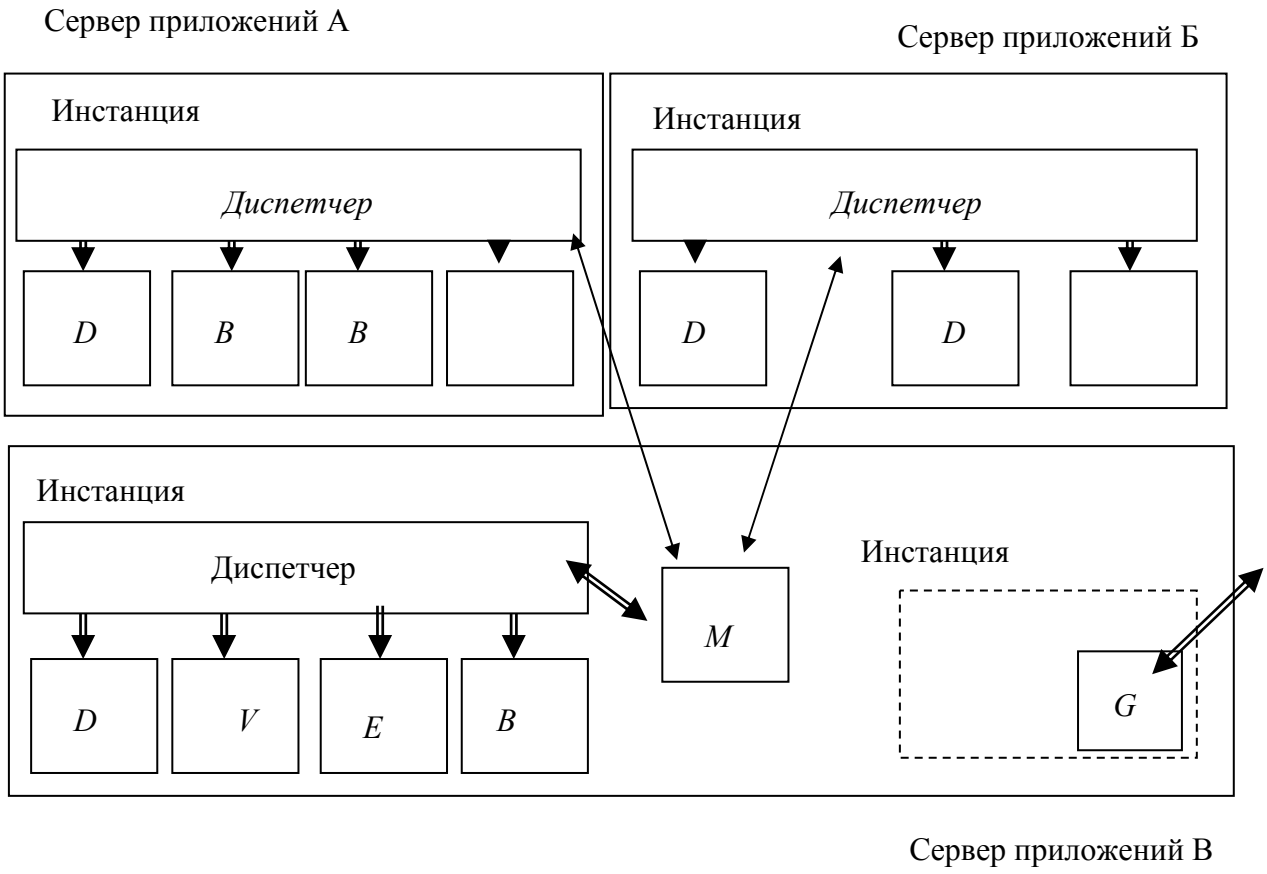


Рис.4.5. Разделяемость на уровне приложений

Как правило, каждая инстанция, предлагающая определенные прикладные процессы, размещается на своей ЭВМ, но не исключена возможность их функционирования на одной ЭВМ.

В зависимости от типа задач, выполняемых в системе *R*/3 в определенный временной интервал, целесообразно изменить набор доступных ресурсов системы, что позволит повысить оптимальность использования ресурсов и повысит производительность системы. Это достигается путем предварительного назначения каждому интервалу времени с отличной от предшествующего временного периода нагрузкой определенного набора рабочих процессов для каждой инстанции системы *R*/3. Каждый такой период называется операционным режимом.

Операционный режим используется также для планирования запуска и остановки отдельных инстанций по календарному графику в зависимости, например, от дня недели и времени суток. В разное время суток рационально изменять соотношения разных типов запущенных рабочих процессов. Так, в дневное время доминируют нагрузки на диалоговые процессы системы, т.к. в течение рабочего дня выполняется много *on-line* запросов к системе от пользователей. В это время можно отложить все некритичные фоновые задания до конца рабочего дня.

Для повышения быстродействия системы (минимизации времени реакции) необходимо установить такой операционный режим, при котором запускается большое число диалоговых процессов и меньше фоновых процессов. В ночное время, когда в системе ведется автоматический сбор статистики и выполняется множество фоновых задач, соотношение рабочих процессов меняется в сторону преобладания фоновых процессов.

Переключение между операционными режимами не требует перезапуска системы и происходит автоматически.

4.2. Рабочие и функциональные процессы системы *R*/3

Рабочий процесс системы *R*/3 включает следующие компоненты:

- процесс динамических программ экранов;
- процесс *ABAP*/4-программ;
- процесс базы данных;
- обработчик задач *R*/3.

При нормальной работе системы *R*/3 количество запущенных рабочих процессов остается постоянным, при этом функции, выполняемые процессом, могут динамически меняться в зависимости от потребностей и

нагрузки на систему в определенный временной интервал. В случае отсутствия нагрузки на рабочий процесс *R*/3, процесс переводится в состояние ожидания, при этом в данный период времени он не требует ресурсов центрального процессора.

В зависимости от реализуемых функций рабочие процессы классифицируются на следующие виды:

1) *B*-фоновые процессы;
2) *D*-диалоговые процессы;
3) *E*-процессы блокировки;
4) *G*-процессы коммуникаций;
5) *M*-процессы обработки сообщений;
6) *S*-процессы вывода на принтер, факс;
7) *V*-процессы обновления данных в БД.

Обработку информации, поступающей от пользователей *R*/3 в режиме *on-line*, ведут диалоговые процессы. Длительность периода времени ограничивается. Для сбора статистических сведений за большой интервал времени, проведение аналитических вычислений используются фоновые процессы. За автоматический запуск фоновых задач отвечает пакетный планировщик.

Спул – процесс *R*/3 формирует данные для печати и передает их спул-системе ОС, которая управляет очередями и обеспечивает выдачу необходимых данных на печать.

Специальные рабочие процессы обновления следят за проведением изменений в БД. Асинхронное обновление обеспечивает высокую производительность в диалоговом режиме, т.к. диалоговая часть транзакций и обновлений БД осуществляется разными процессами.

Целостность данных в распределенной системе обеспечивается механизмом блокировки. Блокировки управляются процессом блокирования с помощью таблицы блокировок, которая располагается в оперативной памяти сервера. Для хранения различных типов данных на каждом сервере приложений используется буферная область памяти.

Для функционирования системы *R*/3 каждому рабочему процессу требуются сетевые ресурсы, которые не должны приводить к простаиванию, как самих процессов, так и системы *R*/3 в целом. При этом между сервером приложений и сервером БД могут проходить огромные объемы данных, превышающие объемы данных, проходящие между сервером приложений и рабочей станцией на несколько порядков. Временные ресурсы БД сопоставимы с остальными ресурсами, что не приводит к простою процесса.

Рабочие процессы *R*/3 запускаются диспетчером каждой инстанции при первоначальном запуске системы. Далее процессы могут находиться либо в режиме работы, либо ожидания, не потребляя и не используя ресурсы процессора. При передаче процессу слишком больших ресурсов памяти система *R*/3 уничтожает этот процесс и запускает новый, тем самым, освобождая память другим процессам.

Перераспределение аппаратных ресурсов и параметров процессов возможно при переходе системы из одного операционного режима в другой (при изменении времен суток).

Рабочие процессы *R*/3 требуют следующих временных и аппаратных ресурсов:

- процессорное время;
- оперативную память для запуска самого процесса;
- программную память в буферной области;
- контекстную область;
- область свертки/развертки;
- дисковую память;
- сетевые ресурсы.

Под функциональными процессами понимают выполнение программы верхнего уровня, вызываемой конечным пользователем для решения прикладной задачи, как в диалоговом режиме, так и фоновом. Функциональные процессы используют программы, написанные на языке четвертого поколения *ABAP*/4, которые называются *SAP*-транзакциями. *SAP*-транзакция представляет собой последовательность взаимосвязанных, логически согласованных шагов диалога вместе с обновлением базы данных (переход БД из одного логического состояния в другое).

В системе *R*/3 транзакции обрабатываются совместным использованием диспетчера, сервера блокирования/разблокирования, обработчика сообщений и обработчика задач обновления данных.

Функциональные процессы задействуют множество аппаратно-программных ресурсов системы. Если функциональный процесс вызывается пользователем, работающим в режиме диалога, то при этом задействуется диалоговый рабочий процесс, который использует программу так и динамические программы экранов, необходимые для взаимосвязи с пользователем. Координирующие действия выполняет диспетчер, управляющий синхронизацией и распределением нагрузки на рабочие процессы системы.

4.3. Разработка математической модели для определения оптимального состава программно-технических ресурсов системы

4.3.1. Определение нагрузки на систему процессами верхнего уровня

Система *R*/3 включает в себя множество функциональных модулей, используемых для ведения различных типов функциональных задач.

Обозначим через Ф множество всех функциональных модулей системы. Обозначим через *k* мощность модуля, под которым будем понимать количество пользователей, работающих в модуле *k*:

$$\Phi_k \rightarrow \mu_{\Phi k},\ \Phi_k \in \Phi.$$

Мощностью всей системы будем называть количество пользователей, работающих во всех модулях, т.е. в системе в целом:

$$M = \sum \mu_{\Phi k.}$$

При установившемся режиме, как правило, не все пользователи непрерывно работают в системе. Пользователи, работающие в данный момент в системе, называются активными пользователями, их число составляет около 65% от общего числа.

Обозначим через *F* множество всех логических (функциональных) процессов, прохождение которых возможно в системе (финансы, планирование персонала и т.д.):

$$F = \mathrm{U}\ F_{\Phi k.}$$

Шагом диалога будем называть такой временной период, в течение которого система отвечает на запрос пользователя.

Мощностью множества функциональных процессов *F* назовем число диалоговых шагов Δ, произведенных при прохождении данных функциональных процессов.

В зависимости от принадлежности процессов определенной сфере задач требуется различное количество программно-технических ресурсов. Так, для модуля материально-технического снабжения требуется гораздо большее одновременно доступное количество таблиц с данными и производится большее количество операций с ними в единицу времени, чем для

модуля финансы. Это приводит к неравномерной нагрузке на систему функциональными процессами разного типа прикладных задач. Для измерения общей нагрузки на систему удобно приводить нагрузку, создаваемую каждым типом функционального процесса, к одному типу функциональных процессов, например модулю финансы *Fi.* и использовать коэффициенты нормализации η_i.

Нормализующий коэффициент модуля *i* вычисляется по формуле:

$$\eta_{i.} = \Delta_i/\Delta_{Fi}\ ,\ \ i\in \Phi.$$

В дальнейшем измерения рационально проводить для этого конкретного модуля, рассчитывая нагрузку для других используя полученные коэффициенты η_i.

Нагрузка на систему, вызываемая прохождением функциональных процессов модуля *i* будет равной:

$$\Delta_i = \eta_i\ \Delta_{Fi}\ ,\ i\in \Phi.$$

Тогда нагрузка на систему, вызываемая прохождением функциональных процессов всех модулей равна:

$$\Delta = \sum(\ \eta_i\ \Delta_{Fi}),\ i\in \Phi.$$

В работающей системе могут быть запущены фоновые задания, отчеты и т.д. Эти работы увеличивают нагрузку. Обычно нагрузка выполнения фоновых задач Δ_B составляет 10%, а выполнения отчетов Δ_R и др. – 7% от нагрузки на систему, вызываемой прохождением функциональных процессов всех модулей:

$$\Delta_{ALL} = \Delta + \Delta_B + \Delta_R\ .$$

4.3.2. Состав программно-техническаих ресурсов системы

Для расчета состава программно-техническаих ресурсов (ПТР) системы необходимо располагать информацией о каждой компоненте структуры корпоративной ИС. Как и было рассмотрено ранее, каждую ИС можно разделить на три сферы задач: представление данных, логику приложений

и хранение данных. Согласно методологии клиент/сервер при использовании распределенной обработки данных компоненты распределяются на трех уровнях иерархии вычислительной системы.

Компонентами сервера баз данных являются:

1) корпус с установленным в нем материнской платой, необходимыми адаптерами и устройствами; внутренним несъемным диском для установки ОС и места для ВЗУО – называемой машиной;

2) отсек для процессорных плат ЦП (до 8-и плат), каждая плата состоит из нескольких процессоров;

3) отсек для плат ОЗУ (до 10-и плат ОЗУ);

4) отсек для внутренних дисков ВЗУ от 4,5 до 18,2 гигабайт.

Сетевые адаптеры выбираются в зависимости от типа сети, адаптеры *SSA* – при установке внешнего дискового массива, адаптеры УА - при установке устройств архивирования.

Компонентами сервера приложений являются:

1) корпус с установленным в нем материнской платой, необходимыми адаптерами и устройствами; внутренним несъемным диском для установки ОС и места для ВЗУО – называемой машиной;

2) отсек для процессорных плат ЦП (до 8-и плат), каждая плата состоит из нескольких процессоров;

3) отсек для плат ОЗУ (до 10-и плат ОЗУ);

4) отсек для внутренних дисков ВЗУ от 4,5 до 36,4 гигабайт.

Сетевые адаптеры выбираются в зависимости от типа сети.

Остальные компоненты системы – сеть, ВЗУ, УА, рабочие станции

В состав сетевого оборудования входят сетевые адаптеры для серверов БД, приложений и рабочих станций, кабели и дополнительное сетевое устройство

Функции ВЗУ может выполнять как набор внутренних дисков сервера БД, так и отдельные дисковые массивы, подключаемые к серверу БД через адаптер *SSA*.

Устройство архивирования создано на оптической библиотеке, подключаемой к серверу БД через адаптер *SCSI Differential.* Объем библиотеки от 18 гигабайт до 1 терабайт и более данных.

Рабочие станции – персональные компьютеры на базе *Intel/Pentium* процессора, ОЗУ – 32 Мб, ВЗУ- 2 Гб, ОС – *Windows NT* и установленным клиентским обеспечением *R*/3.

4.3.3. Составление конфигурационных и ценовых таблиц на устройства системы

Для определения оптимального состава программно-технических ресурсов системы составляются конфигурационные и ценовые таблицы типов машин, типов ВЗУ, типов сетей, устройств архивирования и типов рабочих станций.

Для примера рассмотрим составление конфигурационных и ценовых таблиц для машин. Конфигурационная таблица машин имеет вид (табл. 4.1).

Тип машины однозначно определяет тип процессора, тип плат оперативной памяти, устанавливаемой в ней. Тип процессора однозначно определяет производительность процессора, а тип платы – объем данной платы. Каждая машина в таблице, входящие в нее компоненты и характеристики кодируются.

Таблица 4.1. Типы машин

Код машины	Тип машины	Макс. Кол. ПрП	Макс. Кол. Плат памяти	Мощность одной ПрП (центр)	Мощность одной ПрП (БД)	Мощность одной ПрП (прил.)	Объем одной платы памяти
Ψ_0	Ψ_1	Ψ_2	Ψ_3	Ψ_4^{CL}	Ψ_4^{DB}	Ψ_4^{AP}	Ψ_5
001	$\Psi_1(001)$	*	*	$\Psi_4^{CL}(001)$	*	*	$\Psi_5(001)$
002	*						*
*	*						*
π_m	$\Psi_1(\pi_m)$						$\Psi_5(\pi_m)$

Обозначим через $\psi_1(i)$ функцию, областью определения которой является множество ψ_0, т.е. функцию, заданную на множестве целых чисел, ставящую в соответствие коду машины – ее тип.

Обозначим множество всех типов машин как:

$$\Psi_1 = \{\psi_1(i_1),\ \psi_1(i_2), .. , \psi_1(i_{\pi m})\}, \quad i_\kappa \in \psi_0,$$

$$\Psi_1(i): \psi_0 \rightarrow \Psi_1.$$

Отображение, обратное функции $\Psi_1(i)$, также является функцией, т.к. для каждого j из области значений $\Psi_1(i)$ найдется и будет единственным такое i, что $\Psi_1(i) = j$:

$$\forall\ j \in \Psi_i \exists! i : \Psi_i(i) = j,$$
$$\Psi_1^{-1}(j) : \Psi_1 \to \Psi_0.$$

Функции $\psi_1(i)$ и $\psi_1^{-1}(j)$ устанавливают взаимно однозначное соответствие между типом машины и кодом машины. Тип машины определяет максимальное количество процессорных плат и плат оперативной памяти, устанавливаемых в ней.

По аналогии функции обозначают:

$\Psi_2(i)$ – максимальное количество процессорных плат, устанавливаемых в машине;

$\Psi_3(i)$ – максимальное количество плат оперативной памяти;

$\Psi_4(i)$ – мощность одной процессорной платы. Индекс надстрочный обозначает: *CL* - сервер является централизованным, *DB* - система является распределенной, *AP* - система является распределенной и машина используется как сервер приложений;

$\Psi_5(i)$ – объем одной платы оперативной памяти, устанавливаемой в машину.

Стоимость оборудования (машины, процессорных плат и оперативной памяти) удобно представить в виде таблицы цен (Табл. 4.2).

Таблица 4.2. Цена машины и комплектующих элементов системы

Код машины	Стоимость машины	Стоимость одной процессорной платы	Стоимость одной платы оперативной памяти
Ψ_0	Λ_1	Λ_2	Λ_3
001 * π_m	$\Lambda_1(001)$ * $\Lambda_1(\pi_m)$	$\Lambda_2(001)$ * $\Lambda_2(\pi_m)$	$\Lambda_3(001)$ * $\Lambda_3(\pi_m)$

Здесь через $\Lambda_1(i)$ обозначена функция, областью определения которой является множество Ψ_0 , областью значений является подмножество целых чисел, ставящую в соответствие коду машины – стоимость данной

машины Λ_1, стоимость процессорной платы Λ_2 и стоимость платы оперативной памяти Λ_3:

$$\Lambda_1(i): \Psi_0 \rightarrow \Lambda_1,$$
$$\text{где } \Lambda_1\{ \Lambda_1(i_1), \Lambda_1(i_2),\ldots \Lambda_1(i_{\pi m})\}, i_k \in \Psi_0.$$

Данная таблица взаимно однозначно связана с конфигурационной таблицей (табл.4.1) по столбцу кода машины.

Аналогичную структуру имеют таблицы для других компонент системы, устанавливающие взаимосвязь между конфигурацией и ценой.

4.3.4. Определение необходимой мощности процессорных плат, объема плат оперативной памяти, объема ВЗУ, объема устройств архивирования, типа устройств сети и типа рабочих станций

Типы серверов баз данных и приложений должны быть таковыми, что код сервера, определяемый по типу, должен принадлежать множеству кодов машин (табл.4.1). Это условие записывается формулами:

$$\acute{\varepsilon}_1 = \psi_1^{-1}(\acute{\varepsilon}_1) \in \psi_0,$$
$$\acute{\varepsilon}_2 = \psi_1^{-1}(\acute{\varepsilon}_2) \in \psi_0.$$

Пусть Θ - максимальное допустимое количество серверов приложений ($\acute{\varepsilon}_3$) в системе, тогда:

$$\acute{\varepsilon}_3 \in (0, 1, 2, ..\Theta).$$

Если $\acute{\varepsilon}_3 = 0$, то система является централизованной. При $\acute{\varepsilon}_3 > 0$ система является распределенной.

Количество процессорных плат сервера баз данных ($\acute{\varepsilon}_4$) и каждого сервера приложений ($\acute{\varepsilon}_5$) должны соответствовать допустимому числу процессорных плат для данного кода и соответственно типа машин. Эта зависимость устанавливается по таблице 4.1. Код машины можно получить по типу, используя обратную функцию $\psi_1^{-1}(\acute{\varepsilon}_i)$. Это условие выражается формулами:

$$\acute{\varepsilon}_4 \in (0, 1, 2, ..\psi_2(\psi_1^{-1}(\acute{\varepsilon}_1))) ,$$
$$\acute{\varepsilon}_5 \in (0, 1, 2, ..\psi_2(\psi_1^{-1}(\acute{\varepsilon}_2))) .$$

В распределенной системе (*DB*) вся нагрузка распределяется между серверами баз данных и серверами приложений. Суммарная мощность процессорных плат серверов баз данных должна быть не меньше нагрузки, создаваемой системой на сервер баз данных с учетом коэффициента запаса $\alpha > 0$. Используя функцию $\psi_1^{DB}(i)$ конфигурационной таблицы 4.1, ставящую в соответствие типу машины ее код, получим:

$$\Delta_{ALL} \le (1+\alpha)(\acute{\varepsilon}_4\ \psi_4^{DB}(\psi_1^{-1}(\acute{\varepsilon}_1))).$$

Суммарная мощность процессорных плат всех серверов приложений должна быть не меньше нагрузки, создаваемой системой на серверы приложений с учетом коэффициентов запаса $\alpha > 0$. Поскольку серверы приложений являются гомогенными, т.е. взаимозаменяемыми и одинаковыми по составу компонент, то количество процессорных плат в них одинаково. Используя функцию $\psi_1^{AP}(i)$ конфигурационной таблицы 4.1, ставящую в соответствие коду машины – мощность одной процессорной платы, устанавливаемой в машине, если система является распределенной, и машина используется как сервер приложений, и функцию $\psi_1^{-1}(\acute{\varepsilon}_i)$, ставящую в соответствие типу машины ее код, получим:

$$\Delta_{ALL} \le (1+\alpha)(\acute{\varepsilon}_3\ \acute{\varepsilon}_5\ \psi_4^{AP}(\psi_1^{-1}(\acute{\varepsilon}_2))).$$

4.3.5. Определение необходимого объема плат оперативной памяти

Количество плат оперативной памяти сервера базы данных ($\acute{\varepsilon}_6$) и каждого сервера приложений ($\acute{\varepsilon}_7$) должны соответствовать допустимому числу плат оперативной памяти для данного кода и соответственно, типа машины. Эта зависимость устанавливается при помощи функции $\Psi_3(i)$ из таблицы 4.1. Код машины можно получить по типу, используя обратную функцию $\psi_1^{-1}(\acute{\varepsilon}_i)$. Это условие записывается формулами:

$$\acute{\varepsilon}_6 \in (0, 1, 2, ..\psi_3(\psi_1^{-1}(\acute{\varepsilon}_1))),$$
$$\acute{\varepsilon}_7 \in (0, 1, 2, ..\psi_3(\psi_1^{-1}(\acute{\varepsilon}_2))).$$

Рабочие процессы системы требуют наличие достаточного объема оперативной памяти. При этом каждый тип рабочих процессов характеризуется своим набором требований (табл.4.3)

Таблица 4.3. Рабочие процессы и требуемый объем оперативной памяти

Рабочий процесс	Объем оперативной памяти и код	Место размещения: СБД – сервер баз данных; СП – сервер приложений
Ядро операционной системы *A1X*	χ_{KA}	СБД, на каждом СП
Ядро СУБД *Oracle*	χ_{KO}	СБД
Ядро АСУП *R*/3	χ_{KS}	на каждом СП
Буферная область СУБД *Oracle*	χ_{BO}	СБД
Буферная область АСУП *R*/3	χ_{BS}	на каждом СП
Размещение и функционирование процесса СУБД *Oracle*	χ_{O}	СБД
Размещение и функционирование диалогового рабочего процесса *R*/3	χ_{D}	на каждом СП
Размещение и функционирование рабочего процесса блокировки	χ_{E}	на каждом СП
Размещение и функционирование диспетчера инстанции *R*/3	χ_{I}	на каждом СП
Размещение и функционирование фонового рабочего процесса *R*/3	χ_{J}	на каждом СП
Размещение и функционирование спул процесса *R*/3	χ_{S}	на каждом СП
Размещение и функционирование рабочего процесса обновления	χ_{U}	на каждом СП

Объем оперативной памяти для ядра операционной системы, СУБД *Oracle*, и *R*/3 зависит только от версии этих программных продуктов и не является переменной величиной, зависящей от нагрузки на систему.

Для хранения различных типов данных на сервере приложений используется область памяти, называемая буферной областью. Буферная область состоит из области контекста пользователя, области для хранения выполняемых программ *ABAP*/4, области свертки/развертки, пула таблиц и производственного календаря. Области оперативной памяти для размещения буферов системы *R*/3 и СУБД *Oracle* зависят от нагрузки на систему, т.е. от распределения пользователей *R*/3.

Остальные объемы χ_O, χ_D, χ_E, χ_I, χ_J, χ_S χ_U выделяются под каждый рабочий процесс *Oracle* или *R*/3 и суммарный объем памяти, необходимый для функционирования всех процессов, зависит от количества данных процессов.

В распределенной системе вся нагрузка распределяется между серверами базы данных и серверами приложений. На сервере БД расположена СУБД, на сервере приложений – инстанции *R*/3.

Суммарный объем плат оперативной памяти сервера базы данных должен быть не меньше суммарной потребности в оперативной памяти как операционной системы *A1X*, так и СУБД *Oracle*, с учетом коэффициента запаса $\beta \rangle 0$.

Потребностью в оперативной памяти ОС *A1X* является объем для ядра *A1X*. Потребность в оперативной памяти СУБД *Oracle* складывается из ядра, буферов *Oracle* и процессов *Oracle*.

Суммарная потребность в оперативной памяти сервера БД в распределенной системе будет:

$$X_{DB} = \chi_{KA} + \chi_{KO} + \chi_{BO} + \chi_O^{DB*}.$$

Используя функцию $\psi_5(i)$ конфигурационной таблицы 4.1, ставящую в соответствие коду машины - объем одной платы оперативной памяти, устанавливаемой в ней, и функцию $\psi_1^{-1}(\acute{\varepsilon}_i)$, ставящую в соответствие типу машины ее код, получаем объем оперативной памяти для сервера БД в распределенной системе:

$$X_{DB} \leq (1 + \beta)(\acute{\varepsilon}_6\, \psi_5\, (\psi_1^{-1}(\acute{\varepsilon}_1))).$$

Суммарный объем плат оперативной памяти сервера приложений должен быть не меньше суммарной потребности в оперативной памяти как операционной системы *A1X*, так и АСУП *R*/3, с учетом коэффициента запаса $\beta \rangle 0$. Важно заметить, что серверы приложений являются гомогенными, т.е. взаимозаменяемыми и одинаковыми по составу компонент, количество плат оперативной памяти в них одинаково. На каждом сервере приложений работает одинаковое число нормализованных пользователей, суммарное количество которых по серверам приложений равно общему числу нормализованных пользователей в системе.

Потребность в оперативной памяти ОС *A1X* является объем для ядра *A1X*. Потребность в оперативной памяти АСУП *R*/3 складывается из ядра *R*/3, буфера *R*/3 и процессов *R*/3.

Суммарная потребность в ОП сервера приложений будет:

$$X = \chi_{KA} + \chi_{KS} + \chi_{BS} + \chi_D^{AP*} + \chi_E^{AP*} + \chi_I^{AP*} + \chi_J^{AP*} + \chi_S^{AP*} + \chi_U^{AP*}.$$

Используя функцию $\psi_5(i)$ конфигурационной таблицы, ставящую в соответствие коду машины – объем одной платы оперативной памяти, устанавливаемой в ней, и функцию $\psi_1^{-1}(\acute{\varepsilon}_i)$, ставящую в соответствие типу машины ее код, получаем объем оперативной памяти для сервера приложений:

$$X \le (1 + \beta)\,(\acute{\varepsilon}_7\, \psi_5(\psi_1^{-1}(\acute{\varepsilon}_2))).$$

По вышеописанной методике определяется объем ВЗУ, объем УА, необходимый тип устройств сети и необходимый тип рабочих станций.

4.3.6. Определение оптимальной конфигурации ПТК системы *R*/3

Целевая функция стоимости всех устройств, зависящая от количества и типа устройств ИС, записывается выражением:

$$L = \lambda_1(\psi_1^{-1}(\xi_1)) + \lambda_1(\psi_1^{-1}(\xi_2))\xi_3 + \lambda_1(\psi_1^{-1}(\xi_1))\xi_4 + \lambda_2(\psi_1^{-1}(\xi_2))\xi_3\xi_5 +$$
$$+ \lambda_3(\psi_1^{-1}(\xi_1))\xi_6 + \lambda_3(\psi_1^{-1}(\xi_2))\xi_3\xi_7 + \lambda_4(\psi_7^{-1}(\xi_8)) + \lambda_5(\psi_7^{-1}(\xi_8))\xi_9 + (M + 1 + \Theta)*$$
$$* \lambda_6(\psi_{11}^{-1}(\xi_{10})) + \lambda_7(\psi_{14}^{-1}(\xi_{11})) + \sum \lambda_8(\psi_{17}^{-1}(\xi_{12}^{\Phi\kappa}))\mu_{\Phi\kappa},$$

$$M = \sum_{\Phi\kappa \in \Phi} \mu_{\Phi\kappa},$$

где M - мощность всех пользователей в системе (количество пользователей); Θ- максимальное допустимое количество серверов приложений в системе; $\lambda_1(\psi_1^{-1}(\xi_\kappa))$ - стоимость машин типа ζ_κ, k=1,2; $\lambda_2(\psi_1^{-1}(\xi_\kappa))$ - стоимость процессорных плат машины типа ζ_κ, k=1,2; $\lambda_3(\psi_1^{-1}(\xi_k))$ - стоимость платы оперативной памяти машины типа ζ_κ, k=1,2; $\lambda_4(\psi_7^{-1}(\xi_k))$ - стоимость устройства ВЗУ типа ζ_8; $\lambda_5(\psi_7^{-1}(\xi_8))$ - стоимость дисков ВЗУ типа ξ_8; $\lambda_6(\psi_{11}^{-1}(\xi_{10}))$ - стоимость устройств сети типа ξ_{10}; $\lambda_7(\psi_{14}^{-1}(\xi_{11}))$ - стоимость УА типа ξ_{11}; $\sum \lambda_8(\psi_{17}^{-1}(\xi_{12}))$ - стоимость рабочей станции типа ξ_{12}.

Ищется такое значение переменных ζ_i из областей допустимых значений, при которых целевая функция принимает минимальное значение:

$$L \rightarrow min.$$

При решении ζ_i должны удовлетворять условию минимальной конфигурации. Сервер баз данных с одним процессором, одной платой памяти, ВЗУ, сеть, УА, а также набор рабочих станций всегда должен присутствовать.

4.3.7. Математическая модель

Ниже приводится краткое описание математической модели системы:

$$\Phi_k \rightarrow \mu_{\Phi k},\ \Phi_k \in \Phi.$$

При решении задачи должны обеспечиваться потребности во временных ресурсах:

$$\eta_{\Phi k} = \frac{\Delta_{\Phi k}}{\Delta_{\Phi i}},$$

$$\Delta = \Delta_{Fi} \sum_{\in \Phi} (\eta_{\Phi k} \mu_{\Phi k}),$$

$$\Delta_{ALL} = \Delta + \Delta_B + \Delta_R,$$

и потребности в ресурсах памяти:

Суммарная потребность в оперативной памяти централизованного сервера

$$X = \chi_{KA} + \chi_{KO} + \chi_{KS} + \chi_{BO} + \chi_{BS} + \chi_o^* + \chi_D + \chi_E^* + \chi_I^* + \chi_J^* + \chi_S^* + \chi_U^*,$$

Суммарная потребность в опреативной памяти сервера БД распределенной сиситемы

$$X_{DB} = \chi_{KA} + \chi_{KO} + \chi_{BO} + \chi_O^{DB*},$$

Суммарная потребность в опреативной памяти сервера приложений

$$X_{AP} = \chi_{KA} + \chi_{KS} + \chi_{BS} + \chi_D^{AP*} + \chi_E^{AP} + \chi_I^{AP*} + \chi_J^{AP*} + \chi_S^{AP*}\ \chi_U^{AP*}.$$

Суммарный объем ВЗУ

$$\text{Н} = h_{KO} + h_{KS} + h_{UO}\text{М},\quad \text{М} = \sum_{\Phi\kappa \in \Phi} \mu_{\Phi\kappa},$$

Суммарный объем устройств архивирования

$$\text{А} = a\text{Н}.$$

где а – объем УА в единицах объема ВЗУ, а≥1. Переменные величины: ζ_1 –тип сервера базы данных; ζ_2 – тип сервера приложений; ζ_3 – количество серверов приложений; ζ_4- количество процессорных плат сервера базы

данных; ζ_5 – количество серверных плат каждого сервера приложений; ζ_6- количество плат оперативной памяти сервера базы данных; ζ_7- количество плат оперативной памяти каждого сервера приложений; ζ_8- тип ВЗУ; ζ_9- количество дисков ВЗУ; ζ_{11}- тип УА; ζ_{10}- тип устройств сети; ζ_{12}^{i}- тип рабочих станций для модуля *i*,где $i \in \Phi$ (*AM, CO, WM* и др.).

Тип серверов и их количество выбираются из области допустимых значений:

$$\xi_1 : \psi_1^{-1}(\xi_1) \in \psi_0,$$
$$\xi_2 : \psi_1^{-1}(\xi_2) \in \psi_0,$$
$$\xi_3 : (0,1,2,.....\Theta),$$

где Θ – максимальное допустимое количество серверов приложений в системе.

$$\zeta_4 \in (1, 2, 3,\ldots,\psi_2(\psi_1^{-1}(\zeta_1)));$$
$$\zeta_5 \in (1, 2, 3,\ldots,\psi_2(\psi_1^{-1}(\zeta_2)));$$
$$\zeta_6 \in (1, 2, 3,\ldots,\psi_3(\psi_1^{-1}(\zeta_1)));$$
$$\zeta_7 \in (1, 2, 3,\ldots,\psi_3(\psi_1^{-1}(\zeta_2)));$$
$$\zeta_8 : \psi_7^{-1}(\zeta_8) \rightarrow \psi_6;$$
$$\zeta_9 \in (1, 2, 3,\ldots,\psi_8(\psi_7^{-1}(\zeta_8)));$$
$$\zeta_{10} : \psi_{11}^{-1}(\zeta_{10}) \in \psi_{10};$$
$$\zeta_{11} : \psi_{14}^{-1}(\zeta_{11}) \in \psi_{13};$$
$$\forall \Phi_к \in \Phi \ , \ \xi_{12}^{\Phi к} : \psi_{17}^{-1}(\xi_{12}^{\Phi к}) \in \psi_{16}.$$

Область ограничений по серверной группе для распределенной системы:

$$\Delta_{ALL} \le (1+\alpha)(\xi_4 \psi_4^{DB}(\psi_1^{-1}(\xi_1))),$$
$$\Delta_{ALL} \le (1+\alpha)(\xi_3 \xi_5 \psi_4^{AP}(\psi_1^{-1}(\xi_2))),$$
$$X_{DB} \le (1+\beta)(\xi_6 \psi_5(\psi_1^{-1}(\xi_1))),$$
$$X_{AP} \le (1+\beta)(\xi_7 \psi_5(\psi_1^{-1}(\xi_2))).$$

Для ВЗУ:

$$\mathrm{H} \le (1+\gamma)(\xi_9 \psi_9(\psi_6^{-1}(\xi_8)))/$$

УА:

$$\mathrm{A} \le (1+\delta)(\psi_{15}(\psi_{14}^{-1}(\xi_{11}))).$$

Сеть:

$$\Delta_{ALL} \le (1+k)(\psi_{12}(\psi_{11}^{-1}(\xi_{10}))).$$

Рабочие станции:

нагрузка для всех функциональных модулей

$$\forall \Phi_к \in \Phi, \quad \Delta_{Фк} \le (1+\varsigma)(\psi_{18}(\psi_{17}^{-1}(\xi_{12}^{Фк}))),$$

нагрузка для функциональных модулей AM

$$\Delta_{AM} \le (1+\varsigma)(\psi_{18}(\psi_{17}^{-1}(\xi_{12}^{AM}))),$$

нагрузка для функциональных модулей CO

$$\Delta_{CD} \le (1+\varsigma)(\psi_{18}(\psi_{17}^{-1}(\xi_{12}^{CO}))),$$

.....

нагрузка для функциональных модулей WM

$$\Delta_{WM} \le (1+\varsigma)(\psi_{18}(\psi_{17}^{-1}(\xi_{12}^{WM}))).$$

Блок-схема решения сформулированной задачи приведена на рисунке 4.6

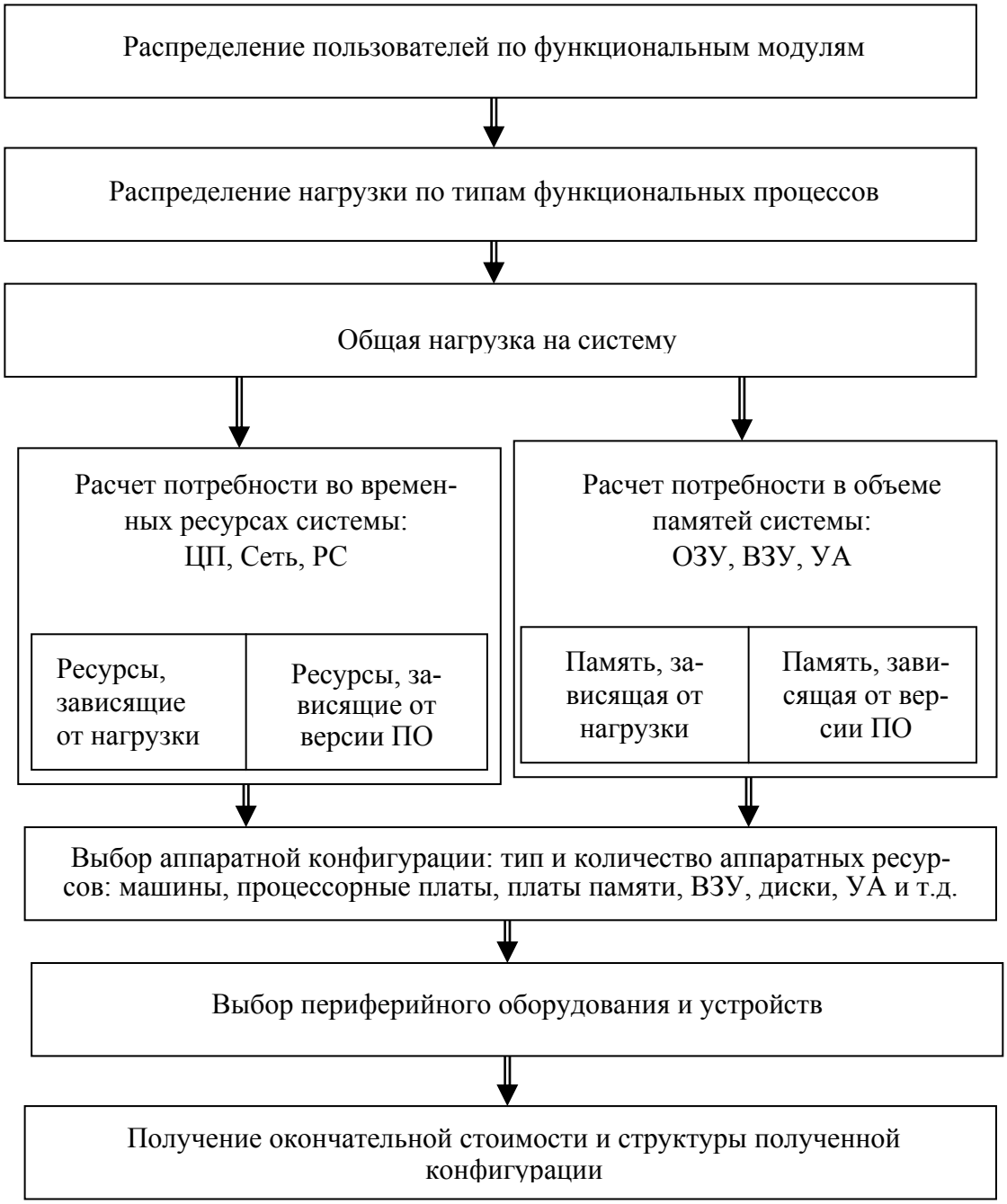


Рис.4.6. Блок-схема решения задачи расчета ресурсов программно-технического комплекса интегрированной системы управления предприятием на базе системы R/3

ИНФОРМАЦИОННЫЙ ПРОЦЕСС ОБРАБОТКИ ДАННЫХ

5.1. Организация вычислительных процессов и обслуживания вычислительных задач

Процесс обработки данных в информационной технологии преследует определенную цель – решение с помощью ЭВМ вычислительных задач, отображающих функциональные задачи той системы, в которой ведется управление. Для реализации этой цели должны существовать модели обработки данных, соответствующие алгоритмы управления и воплощенные в машинных программах.

Процесс обработки данных может быть разбит на ряд связанных между собой процедур: организация вычислительного процесса, преобразование данных и отображение данных.

1. *Организация вычислительного процесса (ОВП)*

Содержание процедуры процесса обработки данных представляет его концептуальный уровень, модели и методы, формализующие процессы обработки данных в ЭВМ – логический уровень, а средства аппаратной реализации процедур – физический уровень процесса.

Процедура ОВП имеет различную функциональную сложность в зависимости от класса и количества решаемых задач, режимов обработки данных, топологии системы обработки данных.

При обработке данных на ЭВМ различают три основных режима: пакетный, разделения времени, реального времени.

При пакетном режиме обработки задания программы с соответствующими исходными данными накапливаются на дисковой памяти ЭВМ, образуя «пакет». Обработка заданий осуществляется в виде непрерывного потока. Размещенные на диске задания образуют входную очередь, из которой они выбираются автоматически последовательно или по приоритетам. Входные очереди могут пополняться в произвольные моменты времени. Такой режим позволяет максимально загрузить ЭВМ, но дает задержки в получении решения из-за того, что некоторое время задание простаивает в очереди.

Режим разделения времени реализуется путем выделения для выполнения заданий определенных интервалов времени (квантов). Предназна-

ченные для обработки задания находятся в ОП ЭВМ одновременно. В течение кванта обрабатывается одно задание, затем выполнение первого задания приостанавливается с запоминанием полученных промежуточных результатов и номера следующего шага программы и в следующий квант обрабатывается второе задание и т.д. Задание при этом режиме все время находится в ОП вплоть до завершения его обработки. При большом числе заданий, для эффективного использования ОП, только что обработанное задание можно временно, до следующего кванта, перемещать во внешнюю память. В режиме разделения времени возможна реализация диалоговых операций.

Режим реального времени используется при обработке данных в ИТ, предназначенных для управления физическими процессами. В таких системах ИТ должна обладать высокой скоростью реакции, чтобы успеть за короткий интервал времени обработать поступившие данные и использовать результаты для управления процессами.

В ЭВМ используются также однопрограммные и мультипрограммные режимы. В режиме разделения времени используется вариант мультипрограммного режима.

Задания в виде программ и данных подвергаются процессу обработки, поступая из системы ввода, системы хранения и по каналам вычислительной сети. В этих условиях остро ставится вопрос планирования и выполнения заданий в вычислительной системе.

Вычислительная среда, в которой протекает процесс обработки данных, может представлять собой одномашинный комплекс, работающий в режиме разделения времени (многопрограммный режим) или многомашинный (многопроцессорный), в котором несколько заданий могут выполняться одновременно на разных ЭВМ (процессорах). В обоих случаях поток заданий должен подвергаться диспетчированию, что означает организацию и обслуживание очереди. Задания, поступившие на обработку, накапливаются в очереди входных заданий. Из этой очереди они поступают на обработку в порядке, определяемом используемой системой приоритетов. Результаты решений задач накапливаются в выходные очереди, откуда они рассылаются либо в сеть, либо на устройство отображения, либо на устройство накопления.

2. *Организация обслуживания вычислительных задач*

При организации и планировании процесса обработки данных в вычислительной системе (ВС) возможны различные методы организации и

обслуживания очередей заданий. При этом преследуется цель получения лучших значений таких показателей, как производительность, загруженность ресурсов, малое время простоя, высокая пропускная способность, разумное время ожидания в очереди заданий.

При организации обслуживания вычислительных задач на логическом уровне создается модель задачи обслуживания, которая может иметь как прямой, так и оптимизационный характер. При постановке прямой задачи данными являются параметры ВС, а решением – показатели эффективности организации вычислительных процессов (ОВП). При постановке оптимизационной задачи задаются требуемые показатели эффективности ОВП, и требуется определить параметры ВС.

В ВС моменты появления заданий являются случайными и случайным является момент окончания вычислительной обработки. Поэтому при проектировании пользуются статистическими данными о среднем количестве поступающих заявок в единицу времени на обработку в ВС, а также о среднем времени решения одной задачи. Эти данные позволяют рассматривать процедуру организации ВП с помощью теории систем массового обслуживания [9]. ОВП можно представить схемой, приведенной на рисунке 5.1.

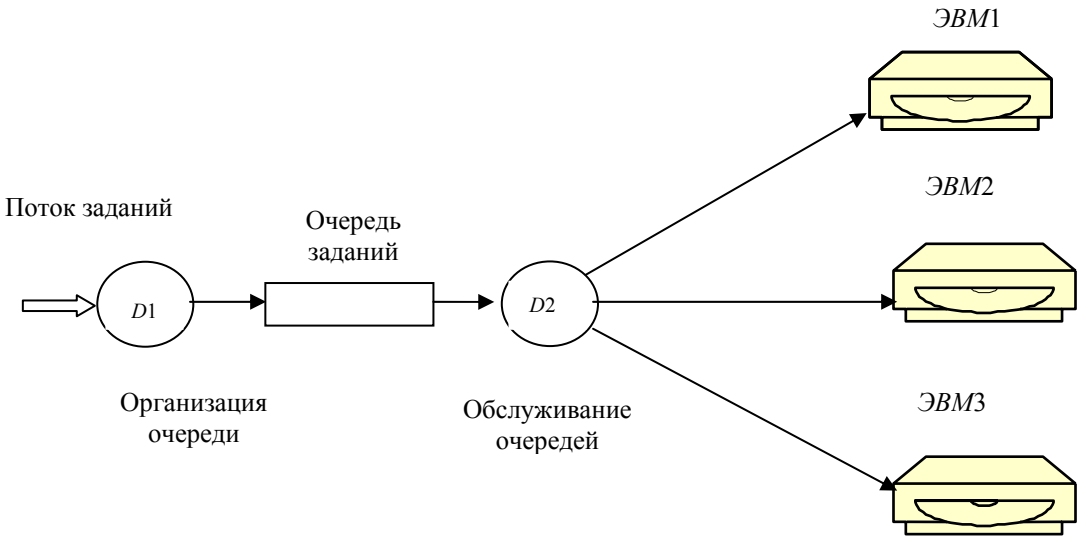


Рис.5.1. Организация обслуживания заданий в многомашинной ВС

Такая схема может быть охарактеризована как система с дискретными состояниями и непрерывным временем. Под дискретным состоянием пони-

мается то, что в любой момент времени система может находиться только в одном состоянии. Число состояний ограничено. Под непрерывным временем подразумевают, что границы перехода из одного состояния в другое не фиксированы. Состояние системы характеризуется числом заданий в очереди плюс число заданий, обрабатываемые ЭВМ. Очередь уменьшается, когда ЭВМ заканчивает обработку задания. Число заданий в системе растет благодаря потоку заданий.

Поток заданий характеризуется интенсивностью λ – средним количеством заданий, поступающим в единицу времени. Среднее время обслуживания одного задания ЭВМ определяет интенсивность потока обслуживания μ:

$$\mu = 1/t_{обсл},$$

где $t_{обсл}$ – среднее время обработки одного задания.

Рассмотрим модель обслуживания вычислительных заданий (рис.5.1), введя следующие предположения:

- в системе протекают Марковские случайные процессы;
- потоки событий (появление заданий, окончание их обработки) являются простейшими;
- число заданий в очереди не ограничено, но конечно.

Случайный процесс, протекающий в системе, называется Марковским.

Простейший поток событий характеризуется стационарностью (независимость параметров во времени), ординарностью (события в потоке появляются поодиночке) и «беспоследствием» (появляющиеся события не зависят друг от друга).

Обозначим состояния рассматриваемой системы:

S_0 – в системе нет заданий;

S_1 – в системе одно задание и оно обрабатывается на ЭВМ1;

S_n – в системе n заданий и они обрабатываются на ЭВМ1, ЭВМ2,..ЭВМn ;

S_{n+1} – в системе (n+1) задание, n заданий обрабатываются на ЭВМ и одно задание стоит в очереди;

S_{n+m} - в системе (n+m) заданий, n заданий обрабатываются на ЭВМ и m заданий стоят в очереди.

Рост числа заявок в системе происходит под воздействием их потока с интенсивностью λ, а уменьшение – под воздействием потока обслужива-

ния с интенсивностью μ. Размеченный граф состояний системы приведен на рисунке 5.2.

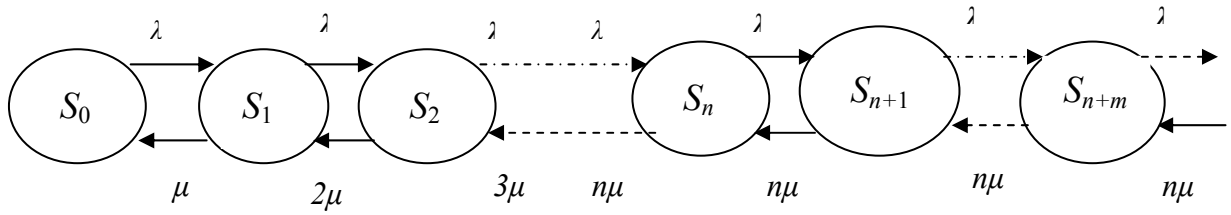


Рис.5.2. Граф состояний многоканальной системы обслуживания с неограниченной очередью

Увеличение числа одновременно работающих машин приводит к росту интенсивности обслуживания от μ до $n\mu$. Дальнейший рост числа заявок переводит систему в состояние $n+1$, $n+2$,...$n+m$, а интенсивность потока обслуживания будет оставаться неизменной, равной $n\mu$.

При исследовании такой вероятностной системы важно знать значение вероятностей состояний, с помощью которых можно вычислить показатели эффективности такие, как количество заданий в системе, время ожидания обработки, пропускная способность и др.

Дискретная система в любой момент времени может находиться только в одном состоянии, поэтому

$$\sum_{i=1}^{k} P_i(t) = 1,$$

где k- число возможных состояний системы.

В процессе работы реальные вычислительные системы быстро достигают установившегося режима. Тогда вероятности состояний не будут зависеть от времени. Для вычисления финальных вероятностей используют систему дифференциальных уравнений Колмогорова, которые превращаются в систему алгебраических уравнений. На основе графа (рис.5.2) может быть записана следующая система алгебраических уравнений [9]:

$$\lambda P_0 = \mu P_1;$$
$$(1\mu + \lambda)P_1 = \lambda P_0 + 2\,\mu P_2;$$
$$(2\mu + \lambda)P_2 = \lambda P_1 + 3\,\mu P_3;$$

$$
\begin{aligned}
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
&(n\mu+\lambda)P_n = \lambda P_{n\text{-}1} + n\mu P_{n+1};\\
&(n\mu+\lambda)P_n = \lambda P_{n\text{-}1} + n\mu P_{n+1};\\
&(n\mu+\lambda)P_{n+1} = \lambda P_n + n\mu P_{n+2};\\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
&(n\mu+\lambda)P_{n+j} = \lambda P_{n+j\text{-}1} + n\mu P_{n+j+1};\ j>=1.
\end{aligned}
$$

Финальные вероятности состояний системы в результате решения системы уравнений описываются следующими уравнениями:

$$
P_0 = \left(1 + \sum_{i=1}^{n} \frac{\rho^i}{i!} + \frac{\rho^{n+1}}{n!(n-\rho)}\right)^{-1},
$$

$$
P_i = \frac{\rho^i}{i!} P_0,\ i=1,2,\ldots n,
$$

$$
P_{n+j} = \frac{\rho^{n+j}}{n^j n!} P_0,
$$

где P_0 –вероятность состояния S_0, при котором в системе заявок нет;
$\rho=\lambda/\mu$ – параметр системы, показывающий, сколько в среднем заявок приходит в систему за время обслуживания заявки одной ЭВМ (одним каналом обслуживания); P_i - вероятность состояния системы S_i , i=1, 2,...n;
P_n – вероятность того, что все ЭВМ заняты обслуживанием заявок;
P_{n+j} – вероятность того, что все ЭВМ системы заняты обработкой заданий и j заявок стоят в очереди.

Приведенные формулы имеют смысл тогда, когда очередь конечна, т.е. $\rho/n < 1$ или $\lambda/n\mu < 1$.

Это выражение говорит о том, что в среднем число заданий, приходящих в вычислительную систему в единицу времени, должно быть, меньше числа обрабатываемых заданий в единицу времени всеми ЭВМ системы. При $\rho/n>1$ очередь растет до бесконечности и такая система не справится с потоком заданий. Появляются задания, ожидающие обработки вечно.

Основными показателями эффективности работы системы являются:

- среднее число занятых каналов (ЭВМ)

$$\bar{k} = \frac{\lambda}{\mu} = \rho,$$

- среднее число заданий в очереди

$$L_{оч} = \frac{\rho^{n+1} \mathrm{P}_0}{nn!(1-\rho/n)^2},$$

- среднее число заданий в системе

$$L_{сист} = L_{оч} + \bar{k},$$

- среднее время пребывания задания в системе

$$W_{ист} = \frac{L_{сист}}{\lambda},$$

- среднее время пребывания задания в очереди

$$W_{оч} = \frac{L_{оч}}{\lambda}.$$

Для уменьшения времени пребывания задания в системе, а значит и в очереди, требуется при заданной интенсивности потока заявок либо увеличивать число обслуживающих ЭВМ, либо уменьшить время обслуживания каждой ЭВМ, либо и то и другое вместе.

С помощью теории массового обслуживания можно получить аналитические выражения и при других дисциплинах обслуживания очереди и конфигурациях вычислительной системы.

При немарковских процессах в системе и не простейших потоках аналитические выражения получить трудно. В таких случаях моделирование проводят с помощью метода статистических испытаний (метод Монте-Карло), который позволяет создать алгоритмическую модель, включающую элементы случайности. Путем многократного запуска модели получают статистические данные, обработка которых дает значения финальных вероятностей состояний.

Организация очереди, поддержание ее структуры возлагаются на диспетчеры *D*1, а передача заданий из очереди на обработку в ЭВМ, поддержание дисциплины обслуживания в очереди (поддержка системы приоритетов) осуществляется диспетчером *D*2 (рис.5.1). В вычислительной

системе диспетчеры реализуются в виде управляющих программ, входящих в состав ОС ЭВМ.

Появление заданий в технологическом процессе обработки данных является случайным, но при решении задач по программе должны быть учтены и минимизированы связи решаемой задачи с другими функциональными задачами, оптимизирован процесс обработки по ресурсному и временному критериям. Поэтому составной частью процедуры организации ВП является планирование последовательности решения задач по обработке данных.

5.2. Организация планирования обработки вычислительных задач

Эффективность обслуживания вычислительных задач зависит от среднего времени обслуживания $t_{обсл} = 1/\mu$, поэтому в вычислительной системе (ВС) требуется решать проблему минимизации времени обработки заданий. Иногда эта проблема трансформируется в задачу максимизации загрузки устройств ЭВМ, являющихся носителями ресурсов. К ресурсам ЭВМ относятся объемы оперативной и внешней памяти, время работы процессора, время обращения к внешним устройствам (внешняя память, устройства отображения). Эти ресурсы ограничены, поэтому требуется найти наилучшую последовательность решения поступающих на обработку вычислительных задач (планирование). Для планирования необходимо знать, какие ресурсы и в каком количестве требует каждая из поступивших задач. Анализ потребности в ресурсах задач производится на основе ее программы решения. После проведения такого анализа становится ясным, какая задача требует каких ресурсов и в каком объеме. Эти данные позволяют перейти к планированию вычислительного процесса.

Критерии, используемые при планировании, зависят от степени определенности алгоритмов решаемых задач. Крайними случаями является:

- порядок использования устройств ЭВМ при решении задач строго задан их алгоритмами;

- порядок использования устройств ВС в задачах заранее не известен.

Для первого случая приемлемым является критерий минимизации суммарного времени решения вычислительных задач, для второго максимизация загрузки устройств ВС.

Рассмотрим модель планирования ВП при минимизации суммарного времени [9]. Каждая программа решения задачи обработки данных вклю-

чает типовые процедуры из набора $П_1$, $П_2$,… $П_m$. Матрица ресурсозатарат T, приведенных ко времени, будет выглядеть так:

$$T = \begin{array}{c|c} & R_1\ R_2 \ldots\ldots\ldots R_n \\ \hline П_1 & \\ П_2 & \tau_{ij} \\ . & \\ П_m & \end{array},$$

где τ_{ij} – затраты j-го ресурса при выполнении i-ой процедуры; R_j – затраты j-го ресурса, $j = 1, 2, ..n$.

С использованием матрицы ресурсозатрат вычисляют суммарные затраты каждого из ресурсов по всем программам решения вычислительных задач. Это позволяет составить матрицу ресурсозатрат по всем программам:

$$T_п = \begin{array}{c|c} & R_1\ R_2 \ldots\ldots\ldots R_n \\ \hline З_1 & \\ З_2 & t_{ij} \\ & \\ З_m & \end{array},$$

t_{ij} - затраты j-го ресурса, приведенного к времени, на выполнение i-й задачи.

В вычислительной системе чаще всего ресурсы используются последовательно. Поэтому на основе матрицы Tп можно выделить последовательность прохождения через обработку задач, которая минимизирует суммарное время. Одним из методов решения подобных задач является метод Джонсона, относящийся к теории расписаний.

При использовании этого метода учитываются следующие ограничения:

1) для любого устройства ВС выполнение последующей вычислительной задачи не может начинаться до окончания предыдущей;

2) каждое устройство в данный момент может выполнять только одну вычислительную задачу;

3) начавшееся выполнение задачи не должно прерываться до полного его завершения.

Если в процессе обработки данных с использованием n устройств на решение поступает m задач, то определение оптимальной последовательности поступления для решения этих задач, минимизирующей общее время обработки, потребует перебора $(m!)^n$ вариантов.

Алгоритм Джонсона, полученный для двух видов ресурса (n=2) требует перебора $m(m+1)/2$ вариантов.

Например, для m=6 и n=2 необходимо проделать 21 перебор вместо 518400 полных переборов.

Рассмотрим алгоритм Джонсона для двухстолбцовой матрицы [9]:

	R_1	R_2
З$_1$		
$T_п$ = З$_2$	t_{ij}	
З$_m$		

1. В матрице $T_п$ определяется $t_{min} = min\ (t_{ij})$.

2. Из задач З$_1$, … З$_m$ выбираются задачи, для которых ресурсоемкость хотя бы по одному устройству была равна $t_{min.}$.

3. Задачи группируют по минимальному времени их исполнения на первом R_1 и вторим R_2 устройствах.

4. В начале последовательности обрабатываемых задач ставят З$_i$(t_{min},, t_{i2}) , в конец - З$_i$ ($t_{i1,}$ t_{min},).

5. Вставленные в последовательность задачи исключаются из матрицы $T_{п.}$

6. Для новой матрицы повторяются пункты 1–3.

7. Задачи З$_i$ (t_{min},, t_{i2}) и З$_i$ ($t_{i1,}$ t_{min},), полученные из новой матрицы, ставятся в середину составленной ранее последовательности и т.д.

При $n>2$ задачу планирования по критерию минимума времени обработки сводят к задаче Джонсона путем преобразования матрицы $T_п$ и проверки выполнения выше описанных условий. В противном случае используют эвристические алгоритмы, в основе которых лежит процедура выбора из поступивших задач наиболее трудоемких и расположение их в порядке убывания времени выполнения.

При планировании по критерию максимума загрузки устройств ВС матрицы ресурсоемкости преобразуются в матрицы загрузки устройств. Из этих матриц формируют по каждому устройству потоки задач, выборки из которых позволяют сформировать оптимальную последовательность задач, подлежащих обработке.

Реализация функций и алгоритмов планирования ВП происходит с помощью управляющих программ ОС вычислительной системы. Программа планировщик определяет ресурсоемкость каждой поступившей на обработку задачи и располагает их в оптимальной последовательности. Подключение ресурсов в требуемых объемах к программе выполнения задач осуществляет по запросу планировщика программа супервизор, которая также входит в состав ОС.

Таким образом, одной из важных процедур информационного процесса обработки данных является ОВП, которая выполняет функции обслуживания поступающих на обработку заданий (очередей) и планирования (оптимизации последовательности) их обработки. На программно-аппаратном уровне эти функции выполняют специальные управляющие программы ОС. Разнообразие методов и функций, используемых в алгоритмах ОВП, зависят от допустимых режимов обработки данных в ВС. В наиболее простой ВС, как персональный компьютер, не требуется управление очередями заданий и планирование вычислительных работ. В ПК применяют в основном однопрограммный режим работы, поэтому их ОС не имеют в своем составе программ диспетчитрования, планировщика и супервизора. В серверах и мэйнфреймах, подобные управляющие программы оказывают решающее влияние на работоспособность и надежность вычислительной системы. Например, к *UNIX* – серверам могут обращаться с заданиями одновременно сотни пользователей, а к мэйнфреймам типа *S/*390 – тысячи пользователей.

5.3. Преобразование данных

Процедура преобразования данных связана с процедурой ОВП, поскольку программа преобразования данных поступает в ОП ЭВМ и начинает исполняться после предварительной обработки управляющими программами процедуры ОВП. Процедура преобразования состоит в том, что ЭВМ выполняет типовые операции над структурами и значениями данных (сортировка, выборка, арифметические и логические действия, создание и изменение структур и устройств данных и т.п.) в количестве и последовательности, заданных алгоритмом решения вычислительной задачи. На физическом уровне вычисления реализуются последовательным набором машинных команд (программой). На логическом уровне алгоритм преобразования данных выглядит как программа, составленная на алгоритмическом языке программирования. Программы с алгоритмических языков с помощью программ-трансляторов переводятся в последовательность кодов машинных команд.

Программа преобразования данных состоит из описания типов данных и их структур, которые будут применяться при обработке, и операторов, указывающих какие типовые действия и в какой последовательности необходимо проделать над данными и их структурами.

Таким образом, управление процедурой преобразования данных осуществляется в первую очередь программой решения вычислительной задачи, и если решается автономная задача, то никакого дополнительного управления процедурой преобразования не требуется. Управление требуется, если информационная технология организуется для периодического решения комплекса взаимосвязанных функциональных задач управления, когда необходимо оптимизировать процедуру преобразования данных по критерию минимизации времени обработки данных, либо по критерию минимизации объема затрачиваемых вычислительных ресурсов. Первый критерий особенно важен в режиме реального времени, а второй – в мультипрограммном режиме.

Программа решения вычислительной задачи преобразует значения объявленных типов данных. В процессе выполнения программы происходит постоянная циркуляция потоков значений данных из памяти ЭВМ и обратно. При выполнении программы к одним и тем же значениям данных могут обращаться различные процедуры и операции, сами операции обработки данных могут между собой комбинироваться различным образом, многократно повторяться и дублироваться.

Следовательно, задачей управления процедурой преобразования данных является, с одной стороны, минимизация информационных потоков между памятью ЭВМ и процессами, с другой – исключение дублирования операций в комплексе функциональных программ.

Первая часть задачи может быть формализована, если структурировать программу на типы применяемых в ней операций, совокупности используемых в них данных (информационных устройств) и связи между ними. Тогда модель этой части задачи преобразования данных может быть представлена в виде двудольного графа, состоящего из множества узлов-операций, соединенных дугами с множеством узлов информационных устройств (рис. 5.3) [9].

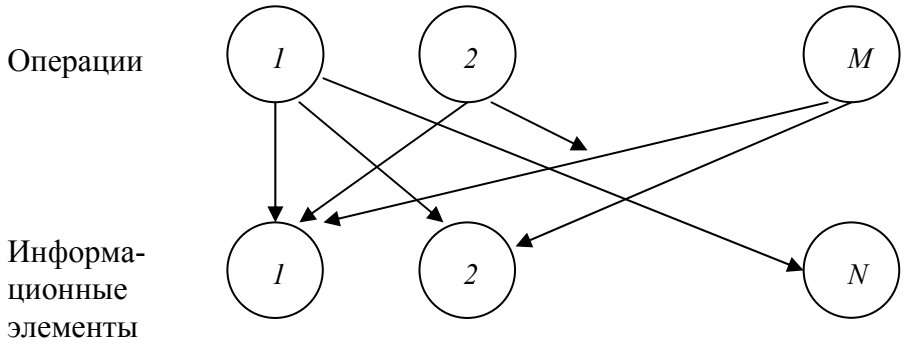


Рис. 5.3. Граф преобразования данных

Этот граф можно сделать раскрашенным, пометить различным цветом дуги, относящиеся к различным информационным элементам. Тогда задача минимизации информационных потоков будет состоять в разбиении раскрашенного графа на подграфы (модули), при котором минимизируется суммарное число дуг различного цвета, связывающих выделенные подграфы.

Для удобства математического описания задачи управления процедурой преобразования и метода ее решения сведем граф (рис. 5.3) к табличной форме [9]. Расположим по строкам выполняемые операции, а по столбцам - элементы множества идентификаторов исходных, промежуточных и выходных данных, связанных с выполнением операций. На пересечении строки и столбца ставится единица, если операция и информационный элемент связаны. В результате получаем матрицу L:

$$L = \begin{array}{c|c} & D_1\ D_2 \ldots\ldots\ldots D_n \\ \hline A_1 & \\ A_2 & \\ \vdots & l_{i,j} \\ A_m & \end{array}$$

При таком представлении задача состоит в разбиении множества операций преобразования данных матрицы L на непересекающиеся подмножества (модули), суммарное число информационных связей между которыми минимально. При решении задачи должны учитываться ограничения: на число выделяемых подмножеств (модулей); на число информационных устройств, входящих в один модуль; на число информационных связей между выделяемыми модулями; на совместимость операций в модулях.

Эта задача может быть сведена к задаче линейного программирования и решена с использованием стандартных программ.

Алгоритм решения большой и сложной задачи и комплекса задач может содержать многократное использование типовых операций в различных комбинациях. Эти комбинации тоже могут многократно использоваться в соответствующих частях большой программной системы. Поэтому второй частью задачи управления процедурой преобразования данных является выделение в алгоритмах решения задач общих операционных комбинаций, выделение их в общие модули и упорядочение таким образом общей схемы алгоритма обработки данных. Эта задача на логическом уровне может быть представлена как задача укрупнения графов алгоритмов [9].

Алгоритм можно представить в виде древовидного графа, узлами которого являются операции над данными, а дугами - связи (отношения) между операциями в алгоритме. Операции в алгоритме выполняются последовательно параллельно. В корне графа расположена начальная операция

A_0, от которой после ее выполнения происходит переход к операции A_1 или A_2, затем к A_3, A_4, …. A_m (рис. 5.4).

Граф можно разметить, написав возле дуг число обращений r_{ij} от операции A_i к операции A_j в процессе выполнения алгоритма. Число обращений в детерминированных алгоритмах больше 1.

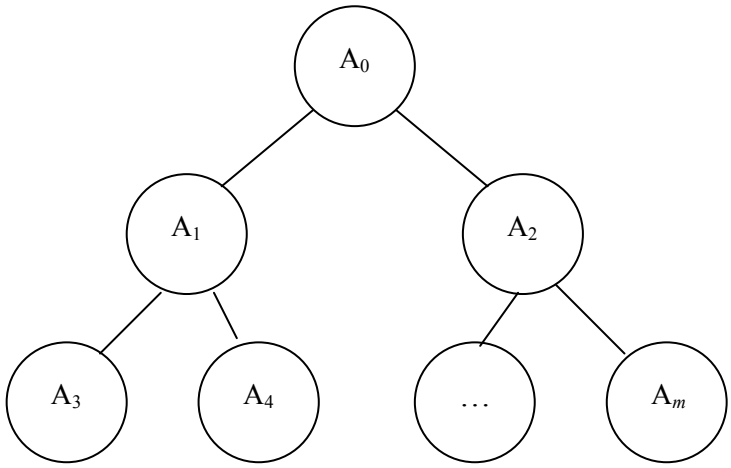


Рис. 5.4. Граф алгоритма

В вероятностных алгоритмах указывается вероятность обращения от операции A_i к операции A_j. При анализе алгоритмов решения вычислительных задач можно выделить общие совокупности операций (пересечение графов) (рис. 5.5).

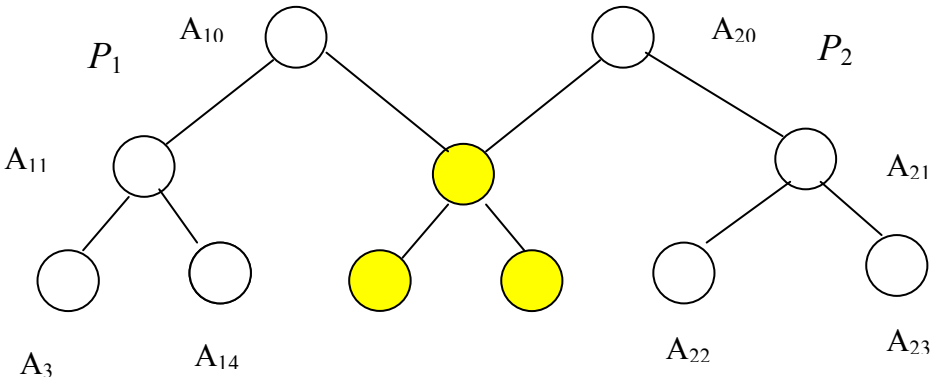


Рис. 5.5. Объединение графов алгоритмов

На рис.5.5 алгоритмы P_1 и P_2 имеют три общие операции, составляющие подмножество операций, входящие одновременно и во множество операций алгоритма P_1 и P_2.

Найдя такие пересечения алгоритмов, общие операции вместе с их отношениями выделяют в отдельные модули. Тогда совокупность алгоритмов может быть представлена в виде вычислительного графа процедуры преобразования данных, в которых определена последовательность выполнения моделей программой системы. Фрагмент вычислительного графа представлен на рисунке 5.6.

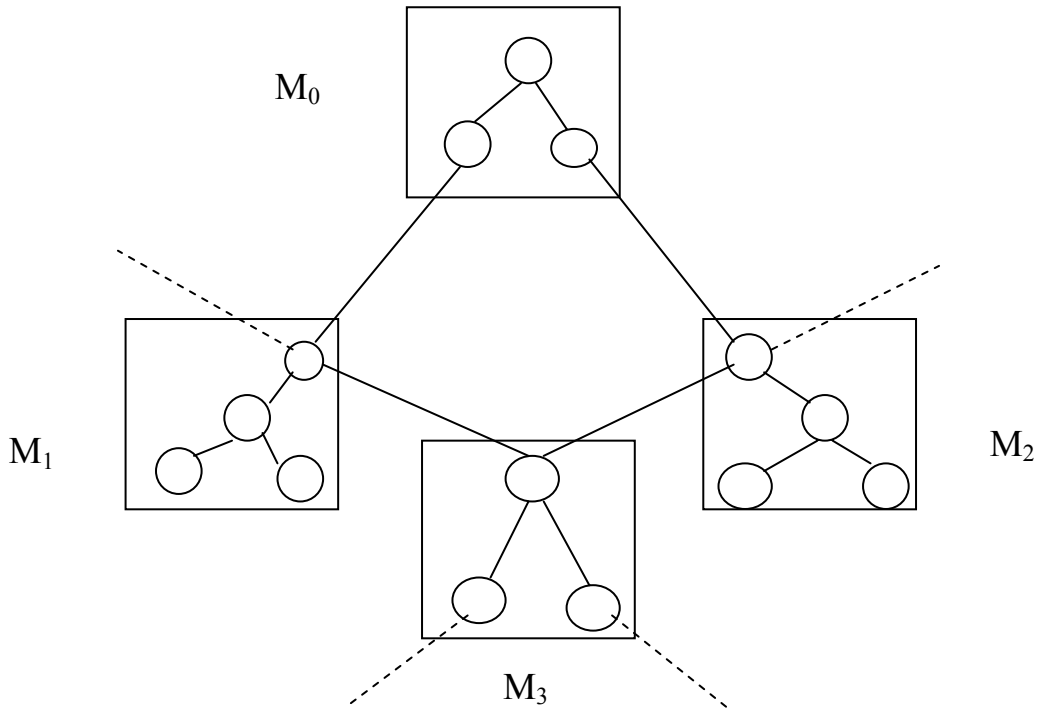


Рис. 5.6. Фрагмент вычислительного графа программной системы

Здесь главным является вычислительный модуль M_0, ему подчинены модули M_1, M_2, M_3. На самом нижнем уровне располагаются модули, выполняющие элементарные типовые операции.

Такая организация алгоритмов преобразования данных позволяет на физическом уровне создать ясную и надежную систему обработки, минимизирующую межоперационные связи. Методом реализации изложенного

подхода является метод структурного программирования, применяемый при создании программных комплексов.

Процедура преобразования данных на физическом уровне осуществляется с помощью аппаратных средств вычислительной системы (процессоры, ОЗУ, ВЗУ), управление которыми производится машинными программами, реализующими структурированную совокупность алгоритмов решения вычислительных задач.

5.4. Нетрадиционная обработка данных

1. *Параллельная обработка*

Необходимость параллельной обработки может возникнуть, когда требуется уменьшить время решения данной задачи, увеличить пропускную способность, улучшить использование системных ресурсов [9].

Для распараллеливания необходимо соответствующим образом организовать вычислительный процесс. Сюда относится следующее:

1) составление параллельных программ, т.е. отображение в явной форме параллельной обработки с помощью надлежащих конструкций языка, ориентированного на параллельные вычисления;

2) автоматическое обнаружение параллелизма. Последовательная программа автоматически анализируется, и в результате может быть выявлена явная или скрытая параллельная обработка. Последняя должна быть преобразована в явную форму.

Рассмотрим граф, описывающий последовательность процессов большой программы (рис.5.7).

На рисунке видно, что выполнение процесса P_5 не может начаться до завершения процессов $P_{2,}$ и P_3 . В свою очередь выполнение процессов P_2 и P_3 не может начаться до завершения процесса P_1. В данном случае для выполнения фрагмента программы достаточно трех процессов.

Ускорение обработки на многопроцессорной системе определяется отношением времени однопроцессорной последовательной обработки T_s к времени многопроцессорной параллельной обработки T_m:

$$U = \frac{T_s}{T_m}.$$

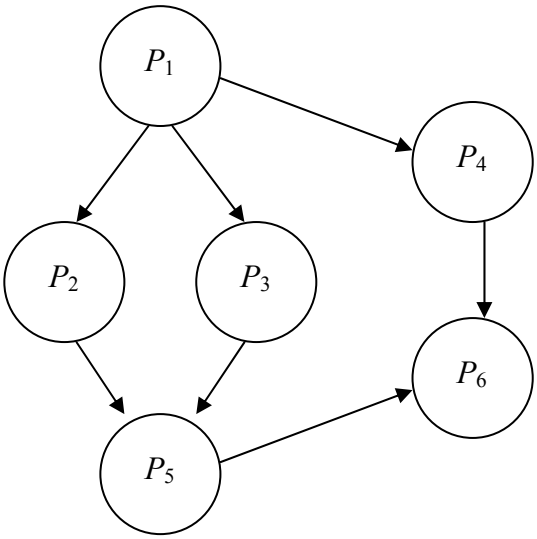


Рис. 5.7. Граф выполнения большой программы

При автоматическом обнаружении параллельных вычислений различают в последовательной программе явную и скрытую параллельную обработку. Хотя в обоих случаях требуется анализ программы. Различие между этими видами обработки состоит в том, что скрытая параллельная обработка требует некоторой процедуры преобразования последовательной программы, чтобы сделать возможным ее параллельное выполнение.

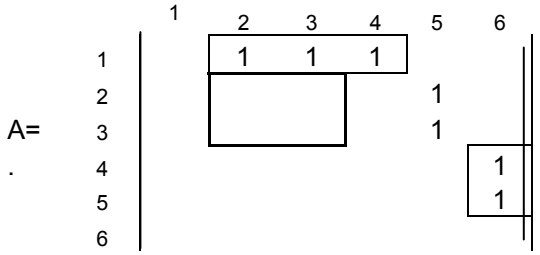
При анализе программы строиться граф потока данных. Чтобы обнаружить явную параллельность процессов анализируются множества входных (считываемых) переменных *R (Read)* и выходных (записываемых) переменных *W (Write)* каждого процесса.

Два процесс P_i и P_j могут выполняться параллельно при следующих условиях:

$$R_i \cap W_j = 0;$$
$$W_i \cap R_j = 0;$$
$$W_i \cap W_j = 0.$$

Это означает, что выходные данные одного процесса не должны модифицироваться другим процессом и никакие два процесса не должны модифицировать общие переменные. Явная параллельная обработка может быть обнаружена среди процессов, удовлетворяющих этим условиям.

Для обнаружения параллелизма граф схему выполнения программы удобно представить в матричной форме. Строки и столбцы матрицы условно нумеруются процессами. Элемент матрицы a_{ij}=1, если выход процесса P_i поступает на вход процесса P_j, в противном случае - равен нулю. Ниже приведем граф-схему в виде матрицы смежностей:



Из матрицы видно, что процессы P_2, P_3, P_4 не могут начаться до завершения процесса P_1 , а после его выполнения дальше могут выполняться параллельно.

Для использования скрытой параллельной обработки требуются преобразования программной конструкции такие, как уменьшение высоты деревьев арифметических выражений, преобразование линейных рекуррентных соотношений, замена операторов, преобразование блоков *IF, DO* к каноническому виду и распределение циклов.

2. *Конвейерная обработка*

Конвейерная обработка улучшает использование аппаратных ресурсов для заданного набора вычислительных процессов, каждый из которых применяет эти ресурсы заранее предусмотренным способом. Хорошим примером конвейерной организации является сборочный транспортер на производстве, например при сборке часов, на котором изделие последовательно проходит все стадии вплоть до готового продукта. Преимущество этого способа состоит в том, что каждое изделие вдоль своего пути перемещения использует одни и те же ресурсы, и как только некоторый ресурс освобождается данным изделием, он сразу же может быть использован следующим изделием, не ожидая, пока предыдущее изделие достигнет конца сборочной линии. Если транспортер несет аналогичные, но не тождественные изделия, то это последовательный конвейер. Если все изделия одинаковы, то это векторный конвейер.

На рисунке 5.8 показано устройство обработки команд, представляющее последовательный конвейер.

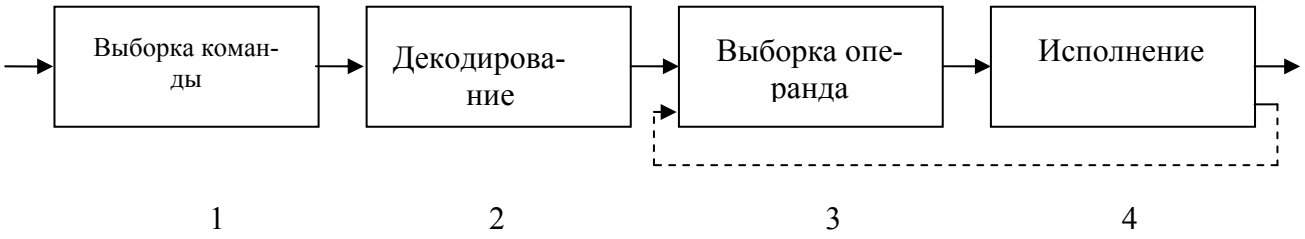


Рис.5.8. Четырехступенчатое устройство обработки команд

Ступени конвейера являются: выборка команды из памяти, декодирование, определение адреса и определение операнда, исполнение. Ускорение обработки в данном устройстве измеряется отношением времени T_i , необходимого для выполнения L заданий, т.е. выполнения L циклов на одной обрабатывающей ступени, ко времени T_p выполнения той же обработки на конвейере. Обозначим через t_i время обработки на i-ой ступени, а через t_j- соответствующее время обработки для самой медленной ступени (рис.5.9).

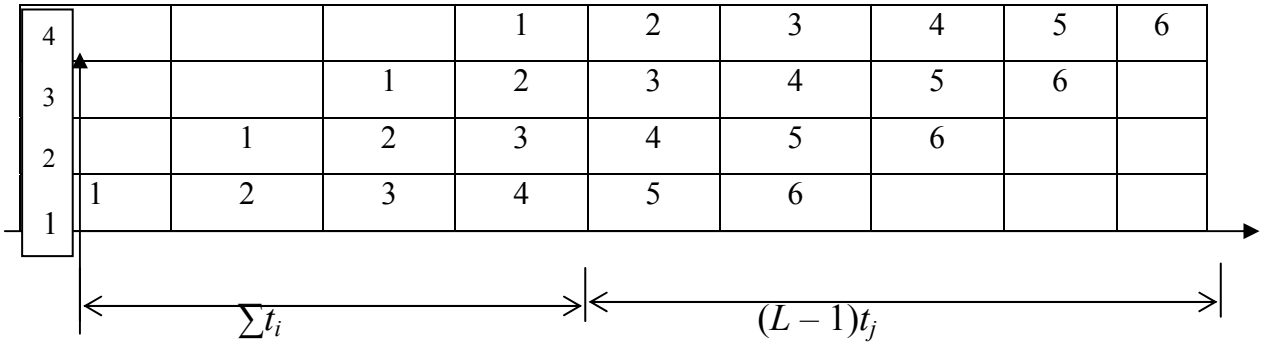


Рис.5.9. Временная диаграмма работы четырехступенчатого устройства обработки команд

Тогда, если L заданий (команд) проходят через конвейер с n ступенями, то эффективность конвейера определяется следующим выражением:

$$\frac{T_s}{T_p} = \frac{L\sum_{i=1}^{n} t_i}{\sum_{i=1}^{n} t_i + (L-1)t_j}$$

$$\text{для} \quad t_i = t_j \to \frac{nL}{n+L-1}.$$

В векторных конвейерах создается множество функциональных устройств, каждое из которых выполняет определенную операцию с парой операндов, принадлежащих двум разным векторам. Эти пары подаются на функциональное устройство и со всеми элементами пар векторов функциональные преобразования проводят одновременно.

Для предварительной подготовки преобразуемых векторов используются векторные регистры, на которых собираются подлежащие обработке векторы.

Типичное использование векторного конвейера – это процесс, вырабатывающий по двум исходным векторам A и B результирующий вектор C для арифметической операции $A + B \to C$. В этом случае на конвейер поступает множество одинаковых команд.

3. *Классификация архитектур вычислительных систем*

Многопроцессорные системы (МПС), ориентированные на достижение сверхбольших скоростей работы, содержат десятки и сотни простых процессоров с упрощенными блоками управления. Отказ от универсальности применения таких ВС и специализация их на определенном круге задач, допускающих эффективное распараллеливание вычислений, позволяют строить их с регулярной структурой связей между процессорами.

Удачной является классификация Флина, которая строится по признаку одинарности или множественности потоков команд и данных [9].

Структура ВС один поток команд, один поток данных *SISD* (*Single Instruction stream, Single Data stream*) на однопроцессорной ЭВМ приведена на рисунке 5.10

Структура многопроцессорной ВС архитектуры один поток команд, много потоков данных *SIMD* (*Single Instruction stream, Multiple Data stream*) содержит некоторое число одинаковых сравнительно простых быстродействующих процессоров, соединенных друг с другом и памятью данных регулярным образом так, что образуется сетка (матрица), в узлах которой размещаются процессоры (рис.5.11).

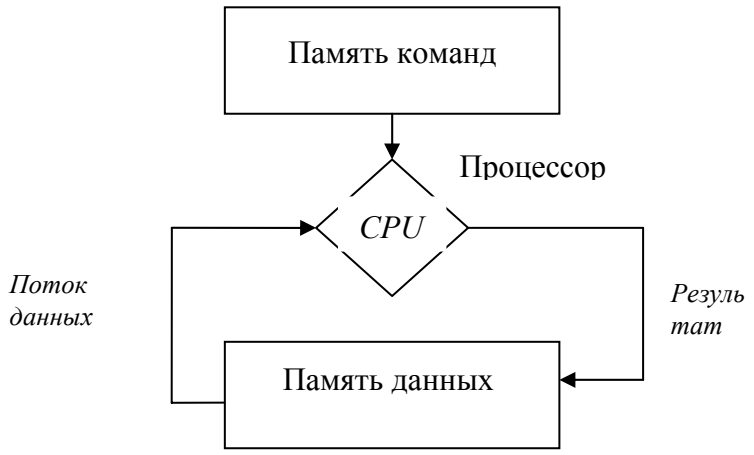


Рис.5.10. Структура ВС архитектуры SISD

В этой структуре возникает сложная задача распараллеливания алгоритмов решаемых задач для обеспечения загрузки процессоров. В ряде случаев эти процессы решаются в конвейерной системе.

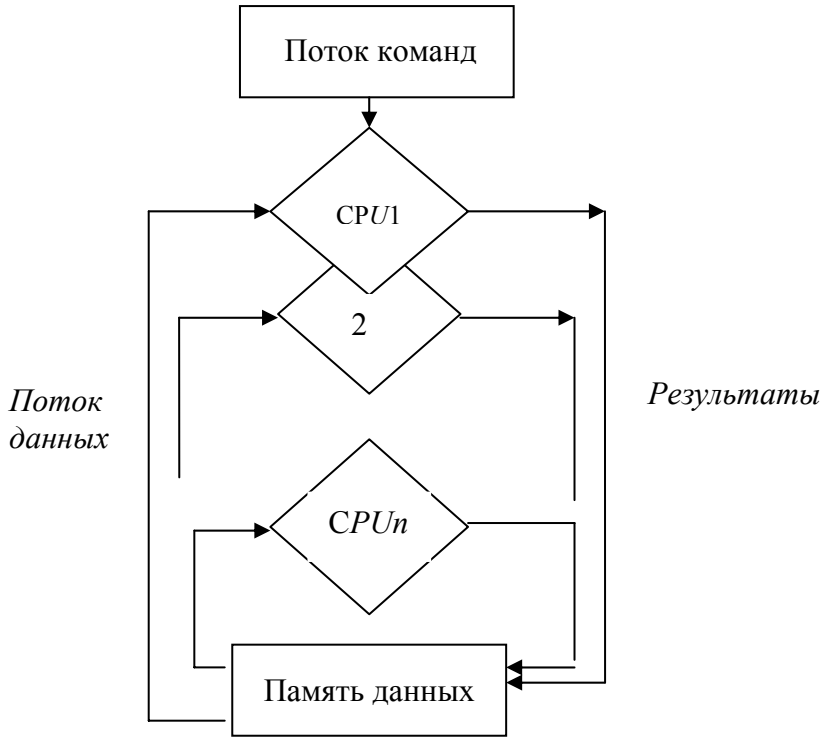


Рис.5.11. Структура ВС архитектуры SIMD

Структура конвейерной микропроцессорной системы много потоков команд, один поток данных *MISD* (*Multiple Instruction stream, Single Data stream*) имеет регулярную структуру в виде цепочки последовательно соединенных процессоров или специальных вычислительных блоков, так что информация на выходе одного процессора является входной информацией для следующего в конвейерной цепочке (рис.5.12).

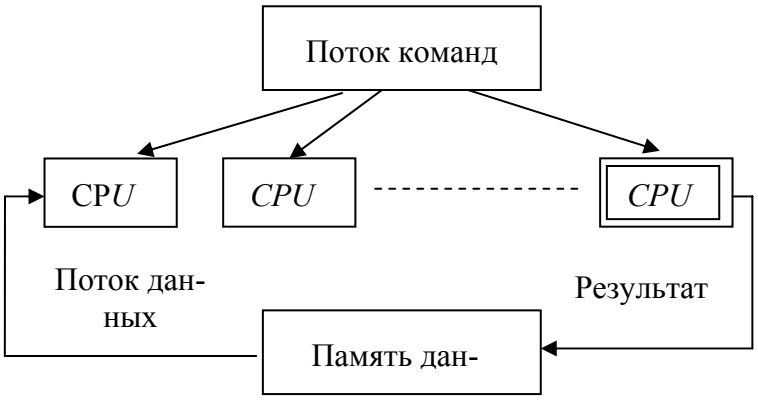


Рис.5.12. Конвейерная микропроцессорная система MISD

Процессоры (специальные вычислительные блоки) образуют конвейер, на вход которого одиночный поток данных доставляет операнды из памяти. Каждый процессор обрабатывает соответствующую часть задачи, передавая результаты соответствующему процессору, который использует их в качестве исходных данных. Таким образом, решение задачи для некоторых исходных данных развертывается последовательно в конвейерной цепочке. Это обеспечивает подведение к каждому процессору своего потока команд, т.е. имеется множественный поток команд.

Структура ВС с архитектурой много потоков команд, много потоков данных *MIMD* (*Multiple Instruction stream, Multiple Data stream*) приведена на рисунке 5.13.

Существует несколько типов *MIMD*, к ним относятся: мультипроцессорные системы, системы с мультиобработкой, многомашинные системы, компьютерные сети.

1. *Типы мультипроцессорных систем*

В зависимости от того, как устроен коммутатор, существуют три типа мультипроцессорных систем: системы с шинной коммутацией, с матричной коммутацией и системы с многопортовой памятью.

Системы с шинной коммутацией отличаются простотой архитектуры. Недостатком является то, что при работе возникают ограничения в передаче команд и данных по шине.

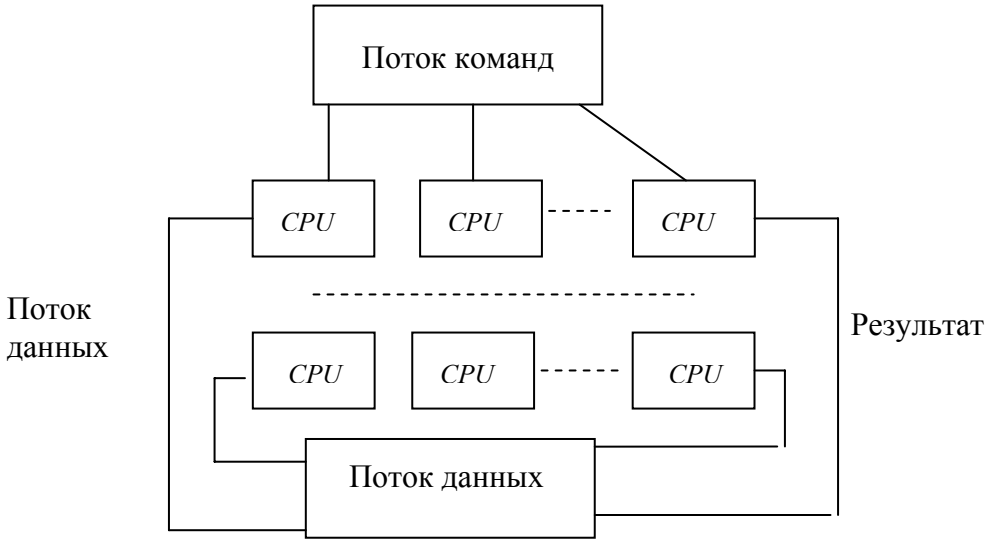


Рис.5.13. Структура ВС архитектуры MIMD

Системы с матричной коммутацией (рис.5.14) характеризуются высоким быстродействием, зависящим от коммутатора. Недостатком этой архитектуры является ее дороговизна.

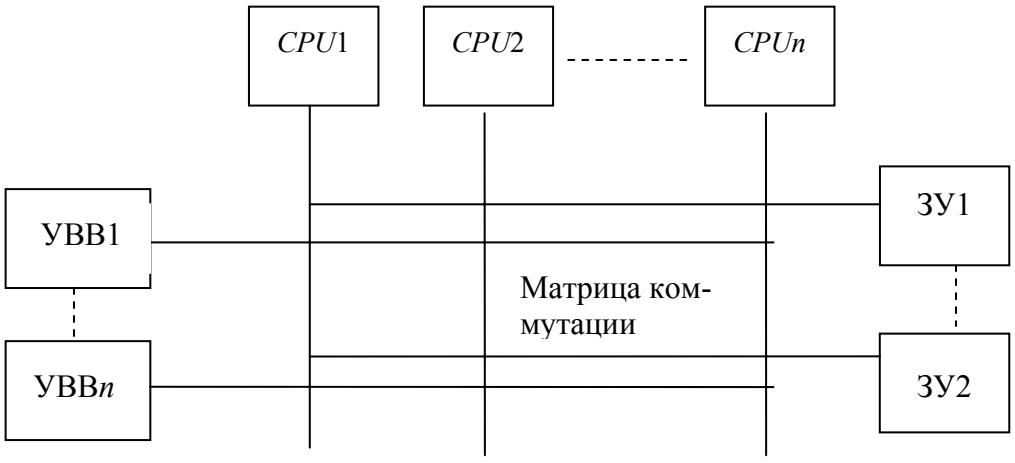


Рис.5.14. Мультипроцессорная система с матричной коммутацией

В системе с многопортовой памятью число процессоров обычно невелико. Функции коммутатора распределены по запоминающим устройствам, в результате чего они являются самыми дорогими блоками. Расширение системы зависит от количества портов запоминающих устройств и достигается очень сложно (рис. 5.15)

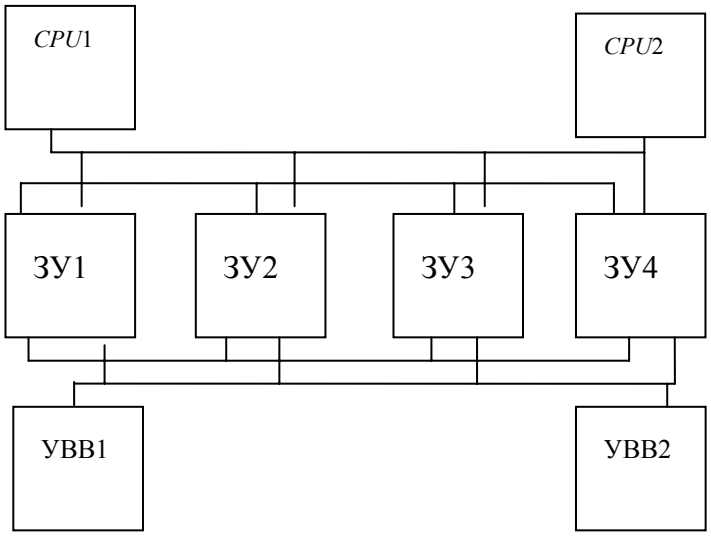


Рис.5.15. Система с многопортовой памятью

4. *Концепция вычислительных систем с управлением потоком данных*

Существуют трудности автоматизации параллельного программирования для обеспечения эффективного использования матричных ВС при решении широкого круга задач. Поэтому актуальным является поиск путей построения высокопроизводительных ВС с управлением потока данных, называемых потоковыми ВС [9].

В системах с управлением потоками данных предполагается наличие большого числа специализированных операционных блоков для определения видов операций (сложения, умножения и т.д., отдельных для разных типов данных). Данные снабжаются указателями типа данных (тегами), на основании которых по мере готовности данных к обработке они загружаются в соответствующие свободные операционные блоки. При достаточ-

ном количестве операционных блоков может быть достигнут высокий уровень распараллеливания вычислительного процесса.

Во всех выше рассмотренных машинах и ВС порядок выполнения операций над данными при решении задач строго детерминирован, он однозначно определяется последовательностью команд программы.

Принципиальное отличие потоковых машин состоит в том, что команды выполняются не в порядке следования команд в тексте программы, а по мере готовности их операндов. Как только будут вычислены операнды команды, она может захватывать свободное операционное устройство и выполнять предписанную ей операцию. В этом случае последовательность, в которой выполняются команды, уже не является детерминированной.

Идея процессора, управляющего потоком данных, приведена на рисунке 5.16.

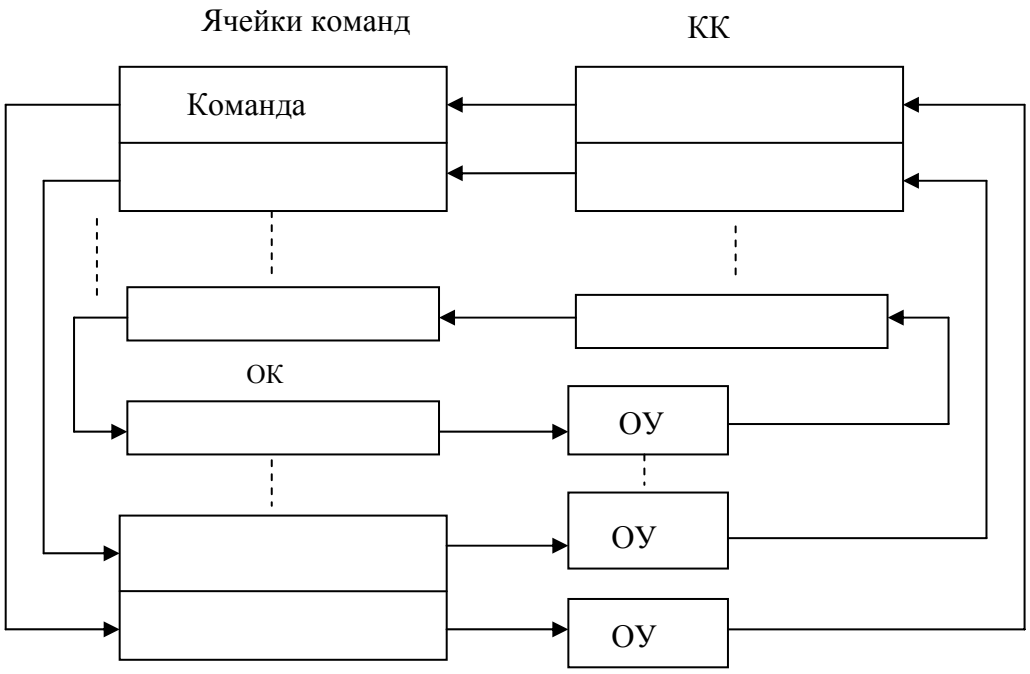


Рис.5.16. Процессор с управлением потоком данных

Потоковая программа размещается в массиве ячеек команд. Команда наряду с кодом операции содержит поля, куда заносятся готовые операнды и поле, содержащее адрес команды, в которые должна быть направлена в качестве операнда результат операции. Кроме того, каждой команде по-

ставлен в соответствие двухразрядный тег (располагаемый в управляющем устройстве), разряды которого устанавливаются в 1 при занесении в тело команды соответствующих операндов. В состоянии тега 11 (оба операнда готовы) инициируется запрос к операционному коммутатору (ОК) на передачу готовой команды в соответствующее коду операции операционное устройство (ОУ). Результат выполнения команды над ее непосредственно адресуемыми операндами направляется через командный коммутатор (КК) согласно указанным в команде адресам в ячейки команд и помещается в поля операндов. Далее указанная процедура циклически повторяется. Управление этим процессом полностью децентрализовано и не нуждается в счетчике команд.

5.5. Управление ресурсами вычислительных систем

1. *Однопроцессорные системы оперативной обработки*

В системах оперативной обработки в качестве основного критерия эффективности используется среднее время обслуживания заявок. В случае, когда априори известны времена решения задач, минимальное среднее время ответа получается при использовании алгоритма *SPT* (*Shortest-Processing-Task-first*). Алгоритм начинает решение задач в порядке убывания времени решения.

В реальных системах оперативной обработки априорная информация о временах решения задач, как правило, отсутствует. Чтобы воспользоваться принципами планирования на основе *SPT*, в систему вводят средства, обеспечивающие выявление коротких и длинных работ непосредственно в ходе вычислительного процесса.

Простейшее правило работ, обеспечивающее выполнение указанного требования, задается алгоритмом циклического обслуживания или алгоритмом *RR* (*Round-Robin*). Алгоритм *RR* приведен на рисунке 5.17.

Заявки на выполнение работ поступают с интенсивностью λ в очередь О, откуда они выбираются и исполняются процессором. Для обслуживания отдельной заявки отводится постоянный квант времени q, достаточный для выполнения нескольких тысяч операций. Если работа была выполнена за время q, то она покидает систему. В противном случае она вновь поступает в конец очереди и ожидает предоставления ей очередного кванта процессорного времени.

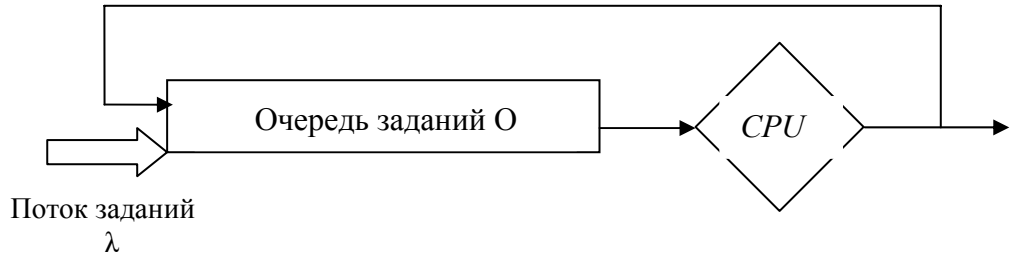


Рис. 5.17. Алгоритм RR

Для обеспечения еще более быстрой реакции системы на короткие работы в системах оперативной обработки используются алгоритмы многоуровневого циклического планирования. Одним из таких алгоритмов является алгоритм *FB* (*Foreground-Background*) (рис.5.18).

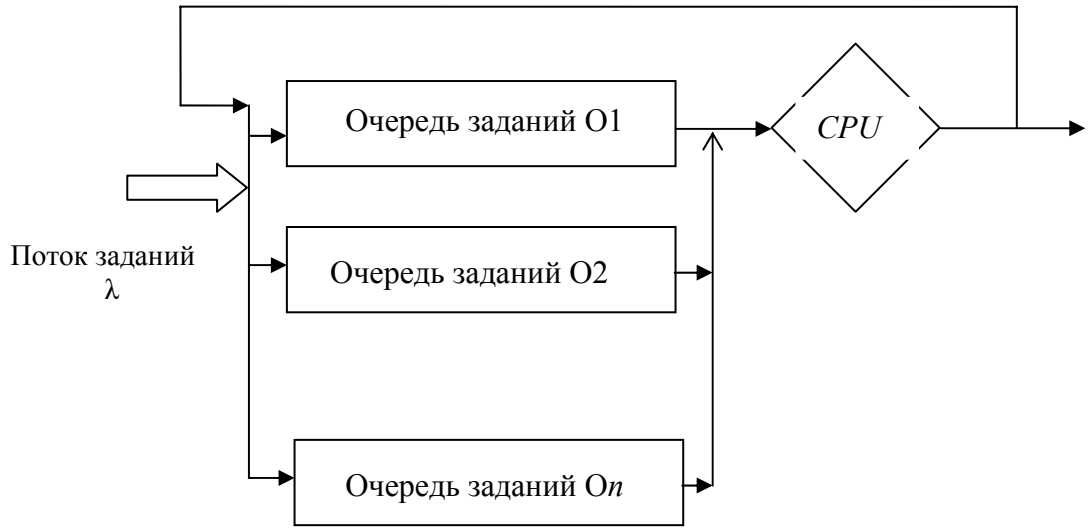


Рис.5.18. Алгоритм FB

Заявки на выполнение работ поступают в очередь О1. Работы, стоящие в очереди О1, получают квант процессорного времени *q*. Если за это время работа была выполнена, то она покидает систему. В противном случае заявка на работу переносится в очередь О2, откуда она может быть занесена в очереди О3, О4,....О*n*. Очереди обслуживаются в следующем порядке. Если имеется хотя одна заявка в очереди О1, то эта заявка непременно обслуживается. Заявки из очереди О2 обслуживаются при условии, что нет заявок в очереди О1. Аналогично заявки из очереди О*m* обслужи-

ваются только в том случае, если все очереди О1, О2, ..О*m*-1 пусты. Заявка, достигшая последней очереди О*n,* остается в ней до полного завершения работы.

Применяются модификации алгоритма *FB*, различающиеся по величине квантов времени, предоставляемых заявкам из разных очередей. Возможно планирование на основе постоянной величины кванта времени или с использованием квантов переменной длительности, которая возрастает по мере увеличения номера очереди.

Одна из таких модификаций – алгоритм планирования *FB* с учетом приоритетов работ. Работы, поступающие в систему, разделяются в зависимости от приоритетов 1, ..*n* на *n* потоков l_1, l_2, ..l_n. Приоритеты задач относительные, т.е. поступление в систему заявки более высокого приоритета не прерывает процесс обработки менее приоритетных заявок. При освобождении ресурса более приоритетные заявки будут назначены в первую очередь. Работа с высшим приоритетом поступает в очередь О1, а работы с низким приоритетом – в очередь О*n*. Работам, выбираемым на обслуживание из разных очередей, выделяются кванты времени различной длительности, причем заявкам из очереди О*m* выделяется больший по продолжительности квант времени, чем заявкам из очереди О*m*-1, *m*=2, 3, ..*n*.

Приоритеты работам могут назначаться исходя из трудоемкости работ. Если трудоемкости работ известны хотя бы приближенно, то работам с большей трудоемкостью присваиваются низкие приоритеты, и они сразу поступают в очереди соответствующего приоритета, в которых получают большие кванты времени. В результате этого трудоемкие работы не будут задерживать процесс выполнения менее трудоемких работ. Если трудоемкость работы была занижена, т.е. ее приоритет оказался завышен, то после окончания выделенного для нее кванта времени, работа переместится в очередь следующего, более низкого приоритета.

Алгоритм планирования с учетом приоритетов очень эффективен для систем с ограниченной емкостью оперативной памяти, не позволяющей разместить в ней программы всех работ, выполняемых системой. В таком случае в оперативной памяти размещается только небольшая часть программ, а остальные программы хранятся во внешней памяти на магнитном диске. Все программы циклически обслуживаются предоставлением им кванта процессорного времени, поэтому они вызываются в оперативную память поочередно, а получив квант обслуживания, удаляются из нее во внешнюю память (на диск). Процесс циклического завершения программ в

оперативной памяти называется свопингом. Если система работает со свопингом и все без исключения работы поступают в первую очередь, причем всем очередям выделяются одинаковые кванты времени, то затраты ресурсов системы на свопинг крайне большие. Для уменьшения непроизводительных затрат целесообразно трудоемкие работы сразу же размещать в очередях с низкими приоритетами и выделять им большие по длительности кванты времени.

При выполнении процедур вытеснения на диск записываются область занимаемой задачей основной памяти и информации о текущем состоянии задач, необходимая для продолжения ее работы. Разделение ресурсов задачами базируется на периодическом уменьшении приоритетов задач, находящихся в основной памяти, и как только приоритет задачи в основной памяти становиться меньше приоритета задачи на диске, выполняется процедура вытеснения.

Приоритетность программ для систем со свопингом может назначаться в соответствии с алгоритмом Корбато. Здесь априорно принимается следующее предположение: программы с большой длиной более трудоемкие. Исходя из этого предположения, приоритеты программ присваиваются на основе формулы:

$$p = \lfloor \log_2 (L_n / L_q + 1) \rfloor,$$

где $[x]$ - целая часть x; L_n-длина программы в байтах; L_q-число байт, передаваемых между оперативной и внешней памятью за время q, равное минимальной длине кванта.

Отношение L_n/L_q определяет число квантов времени, необходимых для загрузки программы в оперативную память и для вывода ее из оперативной памяти.

2. *Многопроцессорные системы при обработке пакетов задач с прерываниями*

Рассмотрим систему с n идентичными процессорами, на которых необходимо решить L независимых задач. Каждая задача решается одним процессором в течение времени t_i, i=1, 2,...L. Требуется найти алгоритм, который для каждого данного пакета (набора) строил бы расписание решения задачи на процессорах системы, обеспечивающее минимально возможное время решения. При этом достигается максимально возможная производительность системы. Например, в двухпроцессорной системе и

при наборе задач с временами t_i (3, 3, 2, 2, 2) возможны различные варианты назначения задач на решение. Приведем некоторые из них (рис. 5.19).

Обозначения: а) работает один процессор; б) работают два процессора; в) используется алгоритм Макнотона

Рис.5.19. Варианты расписаний

Наилучшим по критерию минимизации общего времени решения задач является вариант в), для которого время T пакета задач совпадает с соответствующим оптимальным значением $T=T_o=\theta$ и в данном случае равно величине:

$$\Theta = max\ (max\ t_i, (1/n)\sum t_i).$$

Величина Θ является нижней границей для оптимального значения T_o. Действительно, длина T любого расписания не может быть меньше ни $max\ t_i$ - максимального из времен решения задач пакета П, ни величины $(1/n)\sum t_i$, дающей длину расписания в том случае, когда до момента завершения решения последней из задач пакета ни один процессор не простаивает, т.е. все процессоры имеют 100% -ю загруженность.

В общем случае, даже при n=2 задача поиска оптимального значения T при условии решения задач является *NP* трудной, т.е. все известные алгоритмы ее решения имеют трудоемкость, экспоненциально зависящую от L. Однако если допустить возможность прерывания решения задач пакета до завершения их обслуживания, то могут быть предложены полиномиально-трудоемкие алгоритмы, приводящие к расписанию оптимальной длины T_0. При этом считается, что после прерывания решение задачи мо-

жет быть возобновлено с точки прерывания на любом процессоре, не обязательно на том, на котором она первоначально решалась. Число прерываний должно быть по возможности меньшим, т.к. с каждым актом прерывания связаны потери машинного времени на загрузку-выгрузку задач из оперативной памяти.

Рассмотрим предложенный Макнотоном в 1959 г. алгоритм построения оптимального по длине расписания с не более чем с n-1 прерываниями.

Алгоритм Макнотона заключается в предварительном упорядочении задач по убыванию времени решения и назначении задач последовательно по порядку номеров одну за другой на процессоры системы справа налево от уровня θ.

Примем n=2, L=4, времена решения задач: t_i (5, 4, 3, 2).

Тогда $\theta = max(5, ½(5+4+3+2))=7$.

Расписание, полученное в соответствии с алгоритмом Макнотона, имеет вид, показанный на рисунке 5.20.

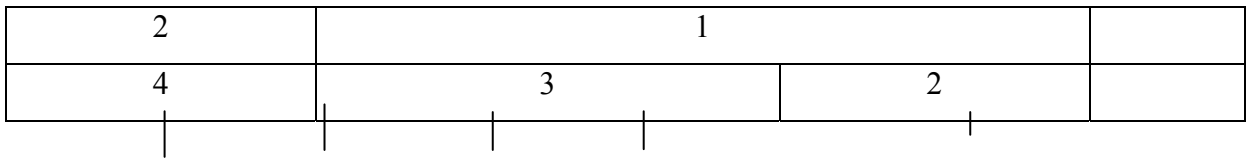


Рис. 5.20. Оптимальное расписание, T=7

В данном случае число прерываний равно 1. Покажем, что n-1 (максимальное число прерываний для расписания, полученное в соответствие с алгоритмом Макнотона) является достижимой границей числа прерываний.

Пусть $L=n+1$, $t_i=n$, $i=1,\ldots n+1$, а расписание, получаемое в соответствии с алгоритмом Макнотона, имеет вид, показанный на рисунке 5.21.

Число прерываний в этом случае, как видно из рисунка 5.21, равно (n-1), что и требовалось показать. Покажем теперь, что любое оптимальное расписание для этого пакета задач также имеет не менее n-1 прерываний. Очевидно, что в любом оптимальном расписании ни один процессор не простаивает на интервале (0, n-1). Предположим, что существует некоторое оптимальное расписание с числом прерываний меньшим n-1. Тогда, по крайней мере, два процессора (предположим для определенности $P_к$ и P_1) обслуживают заявки без прерываний. Очевидно, эти процессоры обслужи-

вают некоторые задачи Z_{ik} и Z_{i1} в интервале (0, n) без прерываний (если решение этих задач начинается позже момента времени t-0, значит, до этого момента на этих процессорах решались некоторые другие задачи, решение которых прерывается в момент начала решения задач Z_{ik} и Z_{i1}). Найдутся моменты времени t, t' такие, что $n \leq t < t' \leq n+1$, и в интервале (t, t') хотя бы один процессор простаивает, а потому рассматриваемое решение не может быть оптимальным.

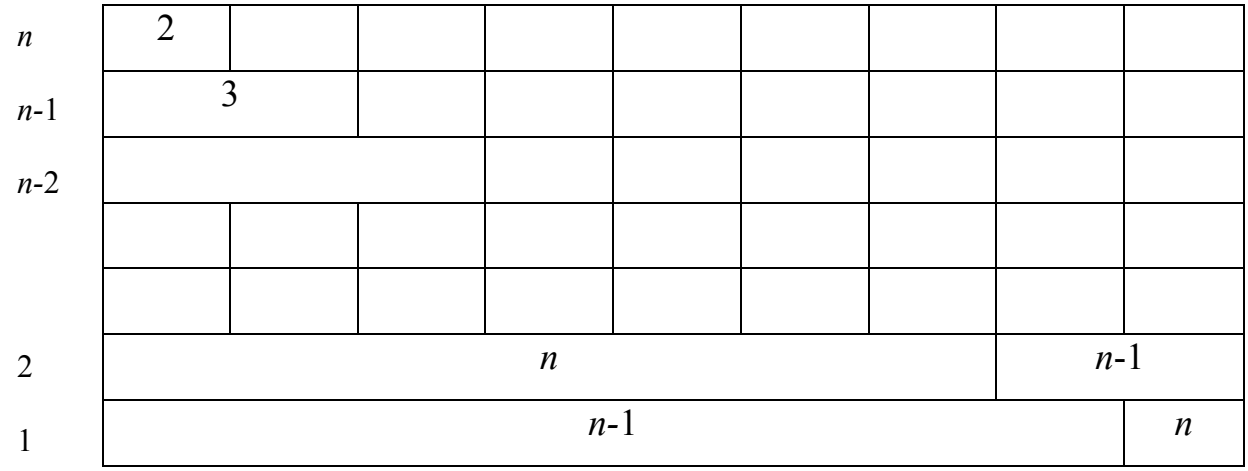


Рис. 5.21. Расписание для n-процессорной системы по Макнтону

Так как мы пришли к противоречию, делаем вывод о том, что предположение о числе прерываний, меньшем n-1, в оптимальном расписании ложно.

2. *Многопроцессорные системы при обработке пакетов независимых задач без прерываний*

Рассмотрим систему, содержащую n идентичных процессоров, на которой необходимо решить без прерываний набор из L независимых задач со временами решения t_i, i=1, 2, ..L. Получение расписания с минимальным временем решения и в этом случае является *NP*- трудной задачей. Один из наиболее эффективных и нетрудоемких алгоритмов организации таких вычислений – алгоритм *LPT* (*Longest-Processing Task firs t*- самая длинная задача решается первой), являющийся частным случаем алгоритма критического пути для независимых задач. Суть этого алгоритма заключается в назначении задач в порядке убывания времени решения на освобождающиеся процессоры. Сотрудником фирмы *BellLaboratories,* США, Грэхемом в 1967 г. был получен следующий результат: при использовании алгоритма

LPT для распределения любого пакета П=(Z_i) независимых задач без прерываний в системе с n идентичными процессорами справедливо:

$$T \leq (4/3\text{-}1/3n)T_0,$$

где T - время решения пакета П при распределении задач алгоритмом *LPT*; T_0 - длина соответствующего оптимального расписания.

Очевидно, что $T_0 = \frac{1}{n}\sum_{i=1}^{I} t_i$.

Приведенная оценка является наилучшей.

5.6. Производительность вычислительных систем

1. *Средства переработки информации (СПИ). Однопроцессорные вычислительные системы*

Основными функциями СПИ, являются:

- прием информации от средств управления, внешних источников сообщений непосредственно или через системы передачи данных (СПД);
- выполнение арифметико-логических операций в ходе вычисления алгоритмов;
- управление процессом обмена со средствами обработки информации, средствами управления и внешними ЗУ;
- выдача информации на средства отображения информации непосредственно внешним потребителем или через СПД;

Структуры СПИ разнообразны. Они могут быть:

- однопроцессорными ЭВМ;
- многопроцессорными (мультипроцессорными) вычислительными системами;
- многомашинными комплексами.

Однопроцессорный вариант построения СПИ является простейшим. Его основные характеристики: быстродействие, состав системы команд, точность вычислений, надежность.

Быстродействие может быть рассчитано по формуле:

$$V_k = \frac{1}{\sum_{l=1}^{n} \beta_l \cdot t_{kl}}, \quad \sum_{l=1}^{n} \beta_l = 1,$$

где β_l – вероятность появления l – ой команды; t_{kl} – длительность выполнения l – ой команды; n – количество команд, которые может реализовать арифметико-логическое устройство (АЛУ).

В большинстве случаев память имеет двухуровневую структуру: внутреннюю и внешнюю. Основные характеристики внутренней памяти: информационная емкость и быстродействие. Внешняя память характеризуется:

- емкостью;
- скоростью обмена информации;
- средним временем доступа к информации по заданному адресу.

Обмен информацией с другими устройствами ВС, как правило, реализуется аппаратными средствами, которые называются устройствами обмена (УО). УО реализуют доступ (обращение) к периферийным устройствам и формируют управляющие сигналы обмена.

В УО используются различные режимы работы. Как правило, с быстродействующими устройствами идет одиночный или групповой обмен массивами данных, а медленно действующие устройства обслуживаются поочередно. Обмен с каждым из таких устройств ведется отдельными байтами или разрядами. Для реализации таких режимов работы в УО предусмотрены селекторные и мультиплексные каналы.

Каналы в зависимости от физической реализации могут быть автономными или встроенными. Автономный канал – это отдельная схема в конструктивном исполнении. Она позволяет обслуживать параллелизм в работе канала УО и *CPU*. Встроенный канал формируется обычно на регистрах и управляющих схемах. Обмен информации производится процессором и приводит к уменьшению производительности процессора.

Обмен данными характеризуется производительностью:

$$N(k) = \sum_{l=1}^{m} \sum_{i=1}^{k} n_i N_{il} \ ,$$

где N_{il} – число операций l – го типа в i-м операторе; n_i – среднее количество выполнения операций в программе; m – количество типов операций; k – решаемая задача.

Пусть за время T решается M типов задач. Каждая задача реализует подмножество операторов. Тогда общее количество вычислений:

$$N_в = \sum_{j=1}^{M} N(k_j).$$

Время, затрачиваемое на реализацию набора задач:

$$T = \sum_{i=1}^{M} t_{y_i} + \sum_{i=1}^{M} t_{вi}^{н},$$

где t_{yi} – среднее время управления при решении i – ой задачи; $t_{вi}^{н}$ - не совмещенное время решения i – ой задачи.

Требуемая производительность вычислений определяется по формуле:

$$V = \frac{N_в}{T} = \frac{\sum_{j=1}^{M} N(k_j)}{(\sum_{i=1}^{M} t_{y_i} + \sum_{i=1}^{M} t_{вi}^{н})}.$$

В реальной системе могут возникать неисправности, обнаружение которых влечет следующие действия:

1) восстановление работоспособности и повторное включение части программы;

2) изменение конфигурации системы и повторное выполнение части программы.

Это требует дополнительных затрат времени $T_п$, поэтому реально приходится увеличить скорость вычислений:

$$V = \frac{N_в}{T - Tп} = \frac{N_в}{T}\left(\frac{1}{(1 - \frac{Tп}{T})}\right).$$

На начальных этапах проектирования, при отсутствии полной информации, принимают $T_п/T = 0{,}3 \div 0{,}6$, тогда формула примет вид:

$$V = \frac{N_в}{T}(1{,}4 \div 2{,}5),$$

т.е. у реальной информационной системы производительность должна быть в 1,4 ÷ 2,5 раз выше по сравнению с безотказным режимом работы.

2. *Операционный метод анализа загрузки устройств ВС*

Метод используется для анализа загрузки и модернизации существующих ВС. Рассмотрение параметров ВС позволяет анализировать производительность ВС и ее отдельных устройств. Чтобы количественно определить меру производительности надо установить элементарную единицу работы для системы.

В режиме пакетной обработки данных единицей работы является задание, которое пользователь предоставляет операционной системе как единое целое.

При интерактивной обработке единицей работы является взаимодействие, состоящее из двух частей: системной и терминальной. Системная часть взаимодействия – это обработка команды, поданной пользователем с терминала. Терминальная часть соответствует действиям пользователя, начиная от получения результата выполнения предыдущей команды и до подачи следующей команды.

Рабочей нагрузкой ВС называется совокупность поступающих на обработку программ, данных и терминальных команд за некоторый период времени. Обычно предполагается, что рабочая нагрузка не чувствительна к изменению производительности ВС, однако ускорение выполнения задания и терминальных команд иногда вызывает более частое применения их пользователем.

Исходные данные для расчета производительности ВС и отдельных ее устройств получают экспериментально с помощью аппаратных и программных измерительных систем.

Параметры, которые обычно фиксируются программами–измерителями, следующие:

C – количество выполненных заданий;

T – полезное время;

B – время выполнения заданий (Пример: $B(1)$ – время работы процессора);

d – число обращений к магнитным дискам;

n – количество напечатанных строк;

m – число сеансов работы в диалоге;

v – суммарное время сеансов.

Расчет производительности работы ВС и ее отдельных устройств будем проводить методами операционного анализа.

Фиксируется некоторый период времени T. Время, в течение которого система обрабатывала задание, обозначим $B(0) \le T$, а количество обработанных заданий за это время обозначим $C(0)$. Индекс 0 означает параметры всей системы.

Рассчитаем параметры системы:

1) Коэффициент использования: $U(0) = \frac{B(0)}{T}$.

2) Среднее время выполнения одного задания: $S(0) = \frac{B(0)}{C(0)}$.

3) Интенсивность выходного потока заданий: $X(0) = \frac{C(0)}{T}$.

Величина X также называется пропускной способностью ВС, и часто используется в качестве критерия производительности ВС, который надо максимизировать:

$$X(0) = \frac{U(0)}{S(0)}.$$

4) Время выполнения задания: $S(0) = Ft$,

где F – среднее количество команд в задании; t – время выполнения одной команды.

В этом соотношении величины t и F зависят как от аппаратуры, так и от программного обеспечения. Поэтому настройка ВС часто предусматривает изменение конфигурации ВС, а также усовершенствование применяемых программ.

Реальная ВС состоит из нескольких устройств, и задание в процессе выполнения захватывает в определенной последовательности эти устройства.

Рассмотрим следующие параметры для отдельных устройств:

$C(i)$ – количество заданий, покинувших i устройство за период времени T;

$C(i, j)$ – количество заданий, покинувших i устройство, и поступивших на j устройство за время T;

$B(i)$ – время занятости i устройства за время T.

Обозначения кодов:

(0) – система в целом;

(1) – центральный процессорный элемент;

(2) – магнитный диск, сервер;

(3) – принтер, сервер.

Окончание обработки задания системой будем рассматривать как переход к нулевому устройству.

Для отдельных устройств определяются:

$U(i)$ – коэффициент использования устройства;

$S(i)$ – среднее время занятости устройства;

$X(i)$ – интенсивность выходного потока заданий с устройства.

Вводятся дополнительные параметры для устройств:

1) $q(i, j)$ – вероятность перехода задания от устройства i к устройству j:

$$q(i,j) = \frac{C(i,j)}{C(i)}, \qquad \sum q(i,j) = 1.$$

$q(0, j)$ – вероятность данного вида означает поступление новых заданий:

$q(0, 1) = 1$ – задания поступающие к центральному процессору.

$q(0, j) = 0,\ j \neq 1,$

$q(i, i) = 0.$

2) $V(i)$ – коэффициент посещения i устройства:

$$V(i) = \frac{X(i)}{X(0)} = \frac{C(i)}{C(0)}.$$

Из соотношения $q(i,j) = \frac{C(i,j)}{C(i)}$ можно определить количество заданий, поступающих в j-е устройство: $C(j) = \sum_i C(i) \cdot q(i,j)$.

Поделив обе части на T, получим соотношение для интенсивности потоков:

$$X(j) = \sum X(i) \cdot q(i,j).$$

Поделим это соотношение на $X(0)$ получим коэффициент посещения j-ого устройства:

$$V(j) = q(0, j) + \sum_{i=1} V(i) \cdot q(i, j), \quad V(0) = 1.$$

Условию $j = 1$ соответствуют ВС с одним центральным процессором, когда от остальных устройств возможен переход только к процессору, или локальные ВС с одним сервером и соединением типа “звезда”.

3) $R(i)$ – среднее время ответа i-го устройства:

$$R(i) = \frac{W(i)}{C(i)},$$

где $W(i)$ – суммарное время ожидания и выполнения заданий: $W(i) \geq B(i)$, $R(i) \geq S(i)$.

4) Средняя длина очереди к i устройству определяется:

$$n(i) = \frac{W(i)}{T}.$$

Поэтому можно определить среднее время ответа устройства (закон Литтла):

$$R(i) = \frac{n(i) \cdot T}{C(i)} = \frac{n(i)}{X(i)}.$$

Сумма средних длин очередей к устройствам называется коэффициентом мультипрограммирования, и вычисляется по следующей формуле:

$$N = \sum n(i) = \sum \frac{W(i)}{T} = \frac{W}{T}, \text{ поскольку } \quad W = \sum W(i),$$

где W – суммарное старт-стопное время выполнения задания при условии, что период наблюдения T не содержит простоев ВС.

Среднее время ответа ВС определяется соотношением:

$$R(0) = \frac{N}{X(0)} = \frac{\sum n(i)}{X(0)} = \sum V(i) \cdot R(i).$$

Функциональные связи устройств ВС удобно описывать в виде графа, вершины которого обозначают номера устройств, а дуги связи между устройствами $q(i, j) > 0$. Рассмотрим два класса ВС. Схема А содержит центральный процессор с двумя каналами ввода/вывода, запуск заданий осуществляется в пакетном режиме.

Схема Б с интерактивной нагрузкой, создаваемой *m* терминалами.

Обе схемы являются системами с центральным процессорным устройством, для которых коэффициенты посещения можно записать следующими формулами (5.1).

$$
\begin{aligned}
&X(0) = X(1) \cdot q(1,0) \\
&X(1) = X(0) + X(2) + X(3) + \ldots + X(k) \\
&\ldots. \\
&X(i) = X(1) \cdot q(1,i) \\
&V(1) = \frac{1}{q(1,0)} \\
&\ldots. \\
&V(i) = \frac{q(1,i)}{q(1,0)}
\end{aligned}
\qquad (5.1)
$$

где k – число устройств.

Системой с центральным устройством может быть локальная ЭВМ, так и сеть из машин с соединением типа «звезда». В последнем случае устройствами считаются отдельные машины. Сеть с соединением типа «звезда» относится к схеме класса Б.

В системе с центральным устройством компьютер - сервер содержит процессор (процессор-сервер), диск (диск-сервер); хост-компьютеры соответствуют отдельным терминалам ВС.

Если хост-компьютер работает в автономном режиме от сервера, то в такой системе мы имеем терминальную часть взаимодействия (обдумывание). А в случае приема-передачи данных, захватывание ресурсов сервера имеем системную часть взаимодействия.

3. *Настройка устройств ВС*

Под настройкой ВС понимается процесс изменения ее конфигурации и регулирования ее параметров, направленный на увеличение производительности.

Будем рассматривать методы настройки, которые можно использовать как для автономно работающей ЭВМ, так и для ЭВМ, работающей в составе сети. Для автономно работающей ЭВМ необходима разработка специальных программ для фиксации времени выполнения заданий. Для ЭВМ, работающих в составе сети, необходимые измерения входят в состав статистики транзакции, которая фиксируется сетевым программным обеспечением.

Выполняемое задание создает различные нагрузки для отдельных устройств ВС. С увеличением числа одновременно выполняемых заданий N у всех устройств системы будет расти коэффициент использования $U(i)$. Среди i устройств может оказаться одно устройство, у которого коэффициент использования может быть близким к единице. Это устройство будет создавать в системе основные задержки для выполнения заданий. Такое устройство называется насыщенным. Для повышения производительности ВС можно заменить насыщенное устройство на более быстродействующее, или снизить нагрузку на него путем изменения структуры базы данных или модификации программы пользователя.

Для характеристики насыщенного устройства (схема А) пользуются следующим соотношением:

$$\frac{U(i)}{U(j)} = \frac{V(i) \cdot S(i)}{V(j) \cdot S(j)}.$$

У насыщенного устройства нагрузка $U(d)$ будет максимальной среди нагрузок других устройств:

$$U(d) = max\ (U(i)), \text{ т.е. } V(d)\ S(d) \rightarrow max\ (V(i)\ S(i))$$

При возрастании коэффициента мультипрограммирования ($N \to \infty$) мы будем иметь $U(d) \to 1$, и в этом случае мы получим, что $X(d) = \frac{1}{S(d)}$.

Поскольку $V(0)=1$, то можно записать соотношение для интенсивности выполнения заданий:

$$\frac{X(0)}{X(d)} = \frac{1}{V(d)}.$$

Следовательно, интенсивность выходного потока заданий не может превысить величины:

$$X' \le \frac{1}{V(d) \cdot S(d)}.$$

Предельный случай можно отобразить графически (рис.5.22)

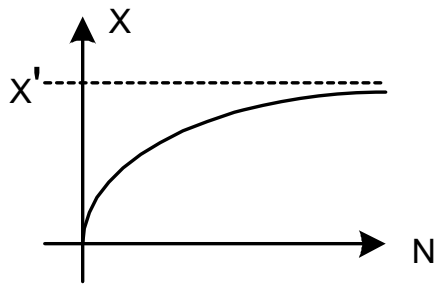


Рис. 5.22. Интенсивность выходного потока заданий

Можно вывести соотношение для минимального среднего времени ответа. Это соотношение можно получить с учетом неравенства, что среднее время выполнения одного задания не может быть больше среднего времени ответа одного устройства:

$$S(i) < R(i),$$

поэтому минимальное время ответа будет:

$$R''(i) = V(i)S(i).$$

Это имеет место для $N = 1$.

Интенсивность выходного потока заданий X в зависимости от N ограничивается величиной X' (рис.5.22).

Для повышения интенсивности выходного потока заданий могут быть использованы следующие возможности настройки ВС:

1) Замена насыщенного устройства на более быстродействующее ($S(d)\downarrow$, т.е. $X'\uparrow$).

2) Заменить хранение производной информации на ее динамическое вычисление или, наоборот, в зависимости от того, какое устройство является насыщенным ($V(d)\downarrow$, т.е. $X'\uparrow$).

3) Модифицировать прикладную программу, чтобы сократить число команд, подаваемых на насыщенное устройство.

4) Если насыщенным является внешнее запоминающее устройство, то в качестве временной меры можно осуществить реорганизацию файлов на данном устройстве.

Насыщенным устройством на практике обычно оказывается центральный процессор или система магнитных дисков.

Рассмотрим соотношения описывающие систему типа Б.

Основные соотношения для вычислительной сети получают на основе закона Литтла. Среднее время одного терминального взаимодействия складывается из двух составляющих: системного времени ответа R и времени обдумывания ответа пользователем Z.

Соотношение

$$M = (R+Z)X(0),$$

где M - количество включенных терминалов, определяет среднее время терминальных взаимодействий:

$$R = \frac{M}{X(0)} - Z \ . \qquad (5.2)$$

Когда число терминалов равно единице ($M = 1$) мы имеем $R=R(0)$.

С увеличением числа терминалов M среднее время ответа R отображается кривой, имеющей асимптоту, определяемую выражением (5.2). Асимптота 1 (рис.5.23) определяет допустимое количество терминалов по заданному времени ответа.

$M(d)$ – количество терминалов, создающее насыщенное состояние системы.

При $M > M(d)$ резко возрастает время ответа.

С течением времени эксплуатации системы возрастает опыт пользователей. Это приводит к уменьшению времени обдумывания Z и, следовательно, к увеличению системного времени ответа R.

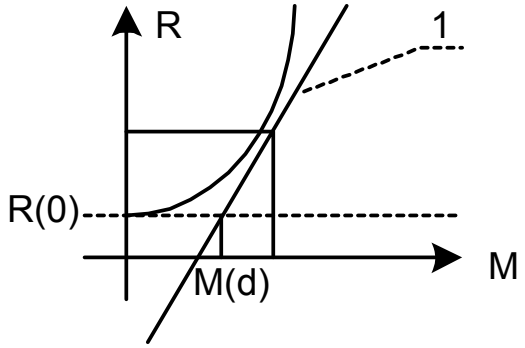


Рис. 5.23. График среднего времени ответа

При неизменном числе терминалов в процессе эксплуатации системы увеличивается коэффициент использования $U(d)$ насыщенного устройства, что приближает необходимость настройки ВС или проведения мероприятий по модернизации ВС.

Одним из допустимых решений является сокращение числа активных терминалов путем перехода к иерархической сетевой структуре. В этой архитектуре «избыточные» терминалы получают доступ к собственному серверу, соединенному с основным сервером сети. Выбор терминалов для нижних уровней иерархической сети из числа действующих терминалов производится методами кластерного анализа на основе близости информационных потребностей у пользователей действующих терминалов.

Пример:

По результатам измерений среднее время выполнения одного задания на ЭВМ составляет 2 минуты ($S(0) = 2$). Интенсивность потока запросов $X(0) = 20$ запросов / час = 1/3 1 / мин.

Определить коэффициент использования ЭВМ - $U(0)$.

Решение:

$$X(0) = \frac{U(0)}{S(0)}, \qquad X(0)S(0) = U(0), \qquad U(0) = \frac{1}{3} \cdot 2 \cong 0{,}67 \text{ или } 67\%.$$

5.7. Производительность мультипроцессорных систем с общей и индивидуальной памятью

Вычислительные системы, содержащие несколько процессоров, связанных между собой и с общим для них комплектом внешних устройств, называются мультипроцессорными системами (МПС). Производительность МПС увеличивается за счет того, что мультипроцессорная организация создает возможность для одновременной обработки нескольких задач или параллельной обработки различных частей одной задачи.

Для обеспечения непрерывности функционирования системы во времени, сохранения ее работоспособности все устройства системы дублируются. Такая система должна содержать не менее двух процессоров, т.е. она строиться как МПС.

Наиболее существен в структурной организации МПС способ связи между процессорами и памятью системы. По этому признаку МПС разделяются на МПС с общей памятью (полнодоступной) и индивидуальной (раздельной).

В МПС с общей памятью каждый процессор имеет доступ к любому модулю памяти, которые могут функционировать независимо друг от друга и в каждый момент времени обеспечивать одновременные обращения в целях записи или чтения слова информации, число которых определяется числом модулей. Возникающие конфликтные ситуации разрешаются коммутатором, начинающим обслуживать первым устройство с наибольшим приоритетом. Каждый процессор может инициировать работу любого канала ввода-вывода.

В МПС с индивидуальной памятью каждый из процессоров обращается в основном к своему модулю памяти. Для обмена данными между подсистемами «процессор-модуль памяти» в процессорах предусмотрены блоки обмена, обеспечивающие передачу сегментов информации между общей памятью и модулем памяти. Блок обмена может работать как селекторный канал: операция обмена инициируется процессором, и передача данных выполняется с параллельной работой процессора.

Принцип индивидуальной памяти позволяет исключить коммутатор в канале «процессор-модуль памяти», вследствие чего увеличивается быстродействие процессоров и уменьшаются затраты оборудования. Отрицательным последствием является потеря ресурсов быстродействия в процессе обмена информацией между модулями памяти и общей памятью системы.

В случае, когда каждый процессор равновероятно обращается к любому сегменту данных, МПС строится по схеме с общей памятью, исключающей необходимость в обмене информацией между модулями памяти.

1. *Характеристики МПС с общей памятью*

В общей памяти такой системы размещаются все программы и данные, используемые в процессе функционирования системы (рис. 5.24).

Такая организация характерна для управляющих систем, ограничения на время реакции которых исключает возможность размещения информации во внешней памяти.

Будем рассматривать однородную МПС, работающую в режиме разделения нагрузки, которую можно моделировать схемой одной многоканальной системы массового обслуживания. Пусть в МПС поступает поток заданий с интенсивностями $\lambda_1 \ldots \lambda_M$. Обслуживание заявок сводится к выполнению соответствующих программ, средние трудоемкости которых равны $\theta_1 \ldots \theta_M$ операций в расчете на один прогон программы. Примем, что обслуживание заявок выполняется на основе дисциплины *FIFO*. В этом случае можно считать, что система обслуживает однородный поток заявок, поступающих с интенсивностью:

$$\Lambda=\sum\lambda_i \ .$$

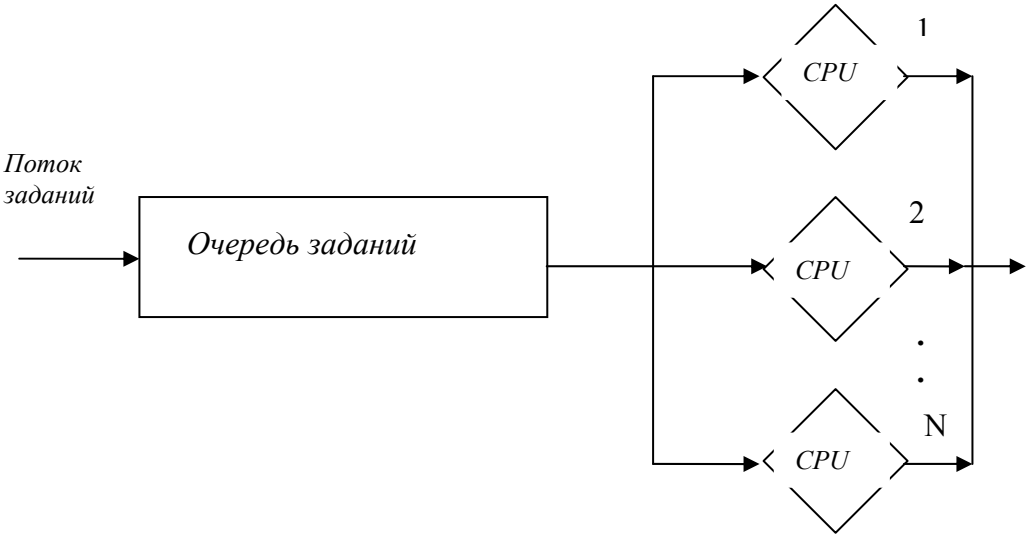


Рис. 5.24. Модель МПС с общей памятью

Для обслуживания любой заявки из суммарного потока требуется в среднем следующее число процессорных операций:

$$\Theta=\sum(\lambda_i/\lambda)\theta_i$$

Примем, что заявка захватывает процессор до полного завершения обслуживания. В этом случае средняя длительность обслуживания заявки υ процессором производительностью B будет равна:

$$\upsilon = \Theta / B$$

с интенсивностью обслуживания $\mu = 1/\upsilon$.

Параметры системы (λ, N, υ) должны отвечать условию существования стационарного режима, когда в очереди пребывает конечное число заявок, и, следовательно, имеет место конечное время ожидания и пребывания заявок.

На каждый процессор поступает N-я доля заявок, отдельный процессор обслуживает поток с интенсивностью λ/N. При этом загрузка процессора будет:

$$\rho = (\lambda/N)\upsilon = \lambda / \mu_{\Sigma},$$

где $\mu_{\Sigma} = N\mu$ – суммарная интенсивность обслуживания заявок N-процессорной системой.

Стационарный режим существует при $\rho < 1$.

Характеристики системы можно получить аналитически, если предположить, что поток заявок – пуассоновский и длительность обслуживания распределена по экспоненциальному закону со средним временем υ.

Суммарная загрузка R системы массового обслуживания, содержащей N каналов, определяется средним числом каналов, занятых обслуживанием заявок:

$$R = \lambda/\mu = N\rho.$$

Для стационарного режима $R < N$, т.к. $\rho < 1$.

Тогда, согласно теории массового обслуживания, вероятность пребывания в системе $n = 0,1,2,\ldots$ заявок, обслуживаемых процессором и стоящих в очереди, равна:

$$p_n = \begin{cases} p_0 \dfrac{N^n}{n!}\rho^n \text{ при } 0 \le n \le N; \\ p_n = p_0 \dfrac{N^n}{N!}\rho^n \text{ при } n \succ N; \end{cases} \qquad (5.3)$$

$$p_0 = \left[\frac{N^{N-1}\rho^N}{(N-1)!(1-\rho)} + \sum_{n=0}^{N-1} \frac{N^n \rho^n}{n!} \right]^{-1},$$

где ρ=λ/(*N*μ) - загрузка процессора *N*-процессорной системы; *n* – количество заявок в *N*-процессорной системе; p_0 – вероятность того, что в системе нет ни одной заявки, т.е. все *N*-процессоров простаивают.

Характер изменения вероятности p_n при изменении суммарной нагрузки четырехпроцессорной системы представлен на рисунке 5.25

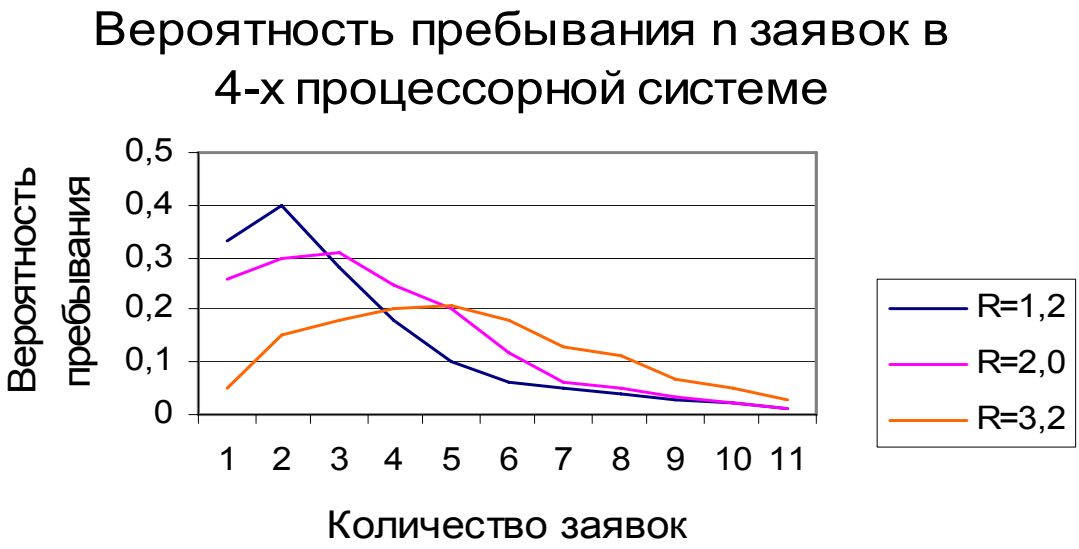


Рис.5.25. Вероятности пребывания заявок в 4-х процессорной системе
Обозначение: R – загрузка системы

Распределение числа заявок в системе носит унимодальный характер. С увеличением нагрузки максимальное значение p_n сдвигается в сторону больших *n*. Распределение содержит всю информацию, необходимую для определения характеристик МПС. Средняя длина очереди заявок, ожидающих обслуживания в *N*-процессорной системе, находится исходя из формул (5.3), как математическое ожидание случайной величины (*n*-*N*)>*0,*

равное числу заявок в очереди:

$$l = \frac{N^{N-1}\rho^{N+1}}{(N-1)!(1-\rho)^2} p_0, \qquad (5.4)$$

где p_0 определяется по формуле (5.3).

Среднее число заявок, пребывающих в системе равно:

$$m=l+R, \qquad (5.5)$$

где l - среднее число заявок в очереди, определяемое формулой (5.4);
R- суммарная загрузка МПС.

Для систем без потерь заявок среднее время ожидания и среднее время пребывания заявок в системе равно соответственно:

$$W=l/\lambda,$$
$$u=m/\lambda.$$

Подставляя в эти соотношения выражения из (5.4), (5.5) получаем:

$$W = \frac{R^N}{\mu(N-1)!(N-R)^2} p_0. \qquad (5.6)$$

$$u=W+\upsilon=W+1/\mu.$$

Одной из важных характеристик системы является вероятность ненулевого ожидания заявок, когда в момент поступления очередной заявки все N процессоров заняты обслуживанием. Эта вероятность равна:

$$p_r(W \succ 0) = \sum_{n=N}^{\infty} p_n = p_0 \frac{N^N}{N!} \sum_{n=N}^{\infty} \rho^n = \frac{N^N \rho^N}{N!(1-\rho)} p_0. \qquad (5.7)$$

Из сравнения формул (5.6) и (5.7) вытекает следующее выражение для среднего времени ожидания заявок:

$$W=p_r(W>0)/(\mu N(1-\rho)).$$

Вероятность нулевого ожидания заявок, т.е. когда в момент поступления заявок хотя бы один процессор свободен, равна:

$$p_r(W=0)=1-p_r(W>0).$$

2. *Характеристики МПС с индивидуальной памятью*

В МПС с индивидуальной памятью множество программ обслуживания и связанных с ними данных $P=(P_1, P_2 \ldots P_M)$ разделяется на подмножества $Q_1, Q_2 \ldots Q_N$, размещаемых в памяти соответствующих процессоров $Пр_1, Пр_2, \ldots Пр_N$. В результате этого каждый процессор ориентирован на обслуживание заявок определенных типов. МПС работает в режиме разделения функций.

В простом случае процессоры обмениваются информацией с общей памятью, количество которой может быть незначительно. В этом случае можно пренебречь влиянием процессов обмена на процесс обслуживания заявок. В таком случае можно считать, что процессоры функционируют независимо и работу N-процессорной системы в режиме разделения функций можно рассматривать как процесс функционирования N одинаковых систем массового обслуживания (рис. 5.26).

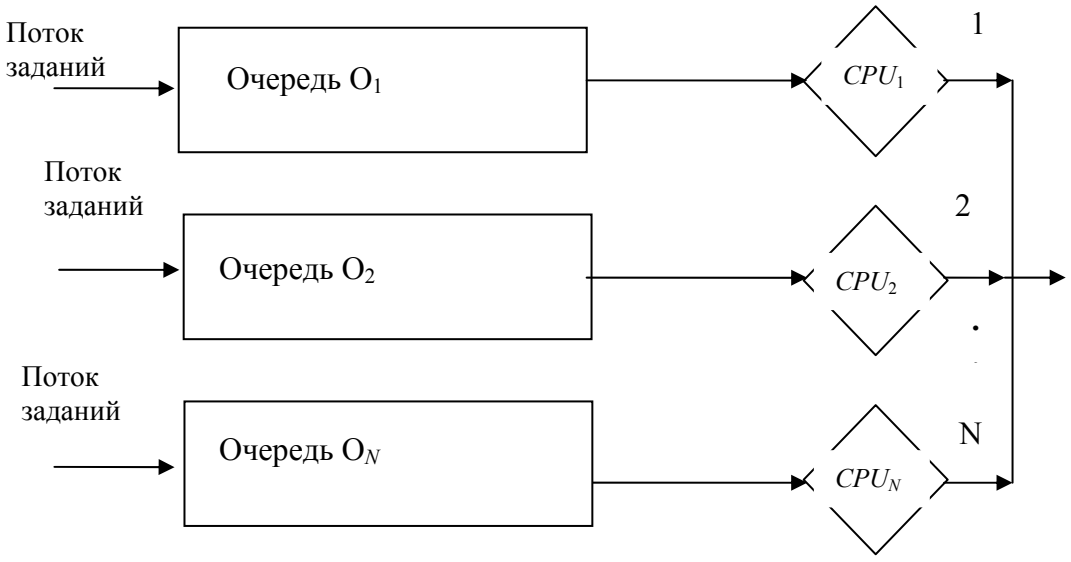


Рис.5.26. МПС с индивидуальной памятью

Каждая из систем состоит из потока заявок, поступающих с интенсивностью λ_i, очереди $О_i$ и процессора $Пр_i$. Для этой модели характеристики обслуживания заявок могут быть вычислены в предположении, что входящие потоки пуассоновские, при произвольном распределении длительностей и дисциплины обслуживания заявок.

При экспоненциальном распределении длительностей обслуживания и дисциплине *FIFO* среднее время ожидания заявок в системе с номером i=1, 2, …N и загрузкой $\rho_i=\lambda_i/\mu_i < 1$ равно:

$$W_i=(\rho_i/(1-\rho_i))\upsilon_i ,$$

среднее время пребывания заявки:

$$u_i = W_i + \upsilon_i = (1/(1-\rho_i))\upsilon_i ,$$

среднее число заявок в очереди:

$$l_i = W_i\lambda_i = \rho_i^2/(1-\rho_i)$$

и среднее число заявок в системе:

$$m_i = u_i\lambda_i = \rho_i/(1-\rho_i) .$$

МПС как единый объект обслуживает суммарный поток заявок, поступающих на вход системы с интенсивностью:

$$\lambda = \sum_{i=1}^{N} \lambda_i .$$

Заявки из суммарного потока с вероятностью λ_i/Λ будут ожидать обслуживания в среднем W_i единиц времени. С учетом этого среднее время ожидания заявки из суммарного потока определяется выражением:

$$W = \sum_{i=1}^{N} (\lambda_i / \lambda) W_i .$$

Аналогично среднее время пребывания заявки в системе составит:

$$U = \sum_{i=1}^{N} (\lambda_i / \lambda) u_i .$$

В случае, когда каждый процессор обслуживает 1/*N*-*ю* часть суммарного потока заявок и средняя длительность обслуживания одинакова для всех процессоров и равна υ, имеет место равномерное распределение нагрузок. Для этого случая имеют место следующие соотношения:

$$\lambda_i = \lambda/N,$$

$$\rho_1 = \ldots = \rho_N,$$

$$W = (\rho/(1-\rho))\upsilon,$$

$$U = (1/(1-\rho))\upsilon.$$

5.8. Точность процесса обработки информации. Баланс погрешностей

Важной характеристикой процесса обработки информации является точность. Обычно точность характеризуют величиной погрешности.

Мерами погрешности являются абсолютная погрешность, относительная и относительная приведенная погрешность.

Абсолютная погрешность единичного вычисления параметра y определяется по формуле:

$$\Delta_y = A_y - a_y,$$

где A_y - истинное значение параметра y; a_y - вычисленное значение параметра y.

Абсолютная погрешность вычисления Δ_y измеряется в тех единицах, что и параметр y.

Для сопоставления погрешностей разных величин используют относительную погрешность, которая является безразмерной величиной или выражается в процентах:

$$\delta_y = \frac{\Delta_y}{a_y} 100\% .$$

Расчетное значение относительной погрешности δ_y зависит от значения рассчитываемой величины a_y.

Для исключения такой зависимости пользуются относительной приведенной погрешностью (ОПП), приводимой к номинальному значению (диапазону изменения) рассчитываемой величины $a_y^н$

$$\delta_y = \frac{\Delta_y}{a_y^н} .$$

В зависимости от величины ОПП различают вычислители (системы):

- прецизионные вычислители, δ_y<0,5 – 1,5%;
- точные системы, δ_y может принимать значения: 0,5%; 1,0%; 2,5%;
- системы невысокой точности, δ_y до 10%.

В технических системах используют точные вычислители.

Погрешности вычислений в зависимости от порождающих факторов и особенностей появления делят на погрешности аппроксимации, вычислительные погрешности, трансформационные погрешности, параметрические погрешности, динамические погрешности.

Погрешности вычислений имеют случайный характер и оцениваются законами распределения. В основном используются симметричные плотности распределения вероятностей. Чаще всего погрешности вычислений аппроксимируются нормальным и равномерным законами распределения. Эти законы характеризуются двумя параметрами: математическим ожиданием и дисперсией.

Математическое ожидание (МО) абсолютной погрешности вычислений $M[\Delta]$ оценивается значением средней абсолютной погрешности вычислений. Дисперсия погрешности вычислений оценивается квадратом среднеквадратического отклонения расчетных значений относительно МО:

$$D[\Delta] = \sigma^2,$$

где σ - среднеквадратичное отклонение погрешности вычислений (СКО).

Для расчетного определения погрешности вычислений вводятся следующие допущения:

1) В составляющих погрешностей отсутствуют систематические компоненты. Тогда математическое ожидание погрешностей вычисления будет равной нулю $M[\Delta]=0$.

2) Компоненты погрешности взаимно не коррелированны. Тогда дисперсия погрешности вычислений будет определяться суммой дисперсий отдельных компонент (баланс погрешностей):

$$\sigma^2 = \sigma_a^2 + \sigma_в^2 + \sigma_т^2 + \sigma_п^2 + \sigma_д^2 , \qquad (5.8)$$

где σ_a^2 - дисперсия аппроксимации; $\sigma_в^2$ - дисперсия вычислительной погрешности; σ_m^2 - дисперсия трансформационной погрешности; $\sigma_п^2$ - дисперсия параметрической погрешности; $\sigma_д^2$ - дисперсия динамической погрешности

Параметры ВС определяют инструментальную составляющую погрешности системы:

$$\sigma_и^2 = \sigma_в^2 + \sigma_m^2 + \sigma_п^2 .$$

Погрешности аппроксимации σ_a^2 и динамическая погрешность $\sigma_д^2$ вычислений мало зависят от технических характеристик ВС.

Погрешность аппроксимации возникает в результате замены оператора, определяющего закон преобразования входных данных в выходные арифметическим оператором. Возникающая при этом погрешность представляет собой разность выходных величин оператора $\Phi(X(t), t)$ и арифметического оператора $\Phi_a(X(t), t)$ в один и тот же момент времени при одинаковых входных данных:

$$\Delta_a = \Phi_a(X(t), t) - \Phi(X(t), t).$$

Величина погрешности зависит от структуры алгоритма, т.е. выбранного численного метода решения задач.

При известном законе распределения плотности вероятностей максимальная вероятная ошибка вычисления может быть оценена по формуле:

$$\Delta_{амак} = \gamma \sigma_a ,$$

где γ зависти от закона распределения и задаваемой вероятности.

Если задаться вероятностью P=0,99, то для равномерного закона распределения имеем $\gamma = \sqrt{12}$, а для нормального $\gamma = 3$. Тогда дисперсия погрешности аппроксимации при нормальном законе распределения может быть вычислена по формуле:

$$\sigma_a^2 = \left(\frac{\Delta амак}{\gamma}\right)^2 \qquad (5.9)$$

Трансформационная погрешность обусловлена влиянием погрешностей квантования составляющих вектора входных данных при аналого-цифровом преобразовании и определяется разрядностью аналого-цифрового преобразователя (АЦП). Погрешность квантования описывается равномерным законом распределения плотности вероятностей. Дисперсия погрешности квантования вычисляется по формуле:

$$\sigma_x^2 = \frac{(\Delta x)^2}{12},$$

где Δx – абсолютная погрешность АЦП, определяемая ценой младшего разряда преобразователя.

Погрешность исходных данных σ_x^2 пересчитывается к расчетной выходной величине y:

$$y = \Phi_a\,(x), \qquad (5.10)$$

где y – расчетная выходная величина; x – входная информация; Φ_a – арифметический оператор.

Тогда дисперсию трансформационной погрешности можно рассчитать, линеаризовав зависимость (5.10):

$$\sigma_т^2 = \left(\frac{\partial \varphi(x)}{\partial x}\right)^2 \sigma_x^2, \qquad (5.11)$$

где σ_x^2 - дисперсия входных данных, поступающих в вычислитель.

Вычислительная погрешность зависит от количества вычислительных операций, разрядности ЭВМ, а также формы представления данных. Вычисления можно представить совокупностью отдельных вычислительных операций. Дисперсия погрешности одной вычислительной операции при равномерном законе распределения погрешности может быть оценена по формуле:

$$\sigma_{01}^2 = \frac{(\Delta y)^2}{12},$$

где Δy – цена младшего разряда расчетной выходной величины; σ_{01}^2 - погрешность одной вычислительной операции.

Если в программе реализуется N вычислительных операций, тогда погрешность вычислений будет накапливаться:

$$\sigma_в^2 = N \cdot \sigma_{01}^2, \qquad (5.12)$$

где $\sigma_в^2$ - погрешность вычислений.

Параметрическая погрешность связана с погрешностями квантования констант вычислительных алгоритмов. Зависит от разрядности и формы представления чисел, имеет равномерный закон распределения. Дисперсия параметрической погрешности вычисляется по формуле:

$$\sigma_п^2 = \frac{(\Delta k)^2}{12}, \qquad (5.13)$$

где Δk – абсолютная погрешность хранения константы.

Динамическая погрешность обусловлена конечным временем выполнения вычислений и, следовательно, запаздыванием в решении задач:

$$\Delta_д = y_д(n\Delta t) - y_a(n - k)\Delta t),$$

где Δt – шаг квантования; $к\Delta t$ – запаздывание вычислений на k–тактов.

Дисперсия динамической погрешности вычислений при равномерном законе распределения плотности вероятностей ошибок равна:

$$\sigma_д^2 = \frac{(\Delta_д)^2}{12}. \qquad (5.14)$$

Подставив в формулу (5.8) составляющие дисперсии погрешностей (5.9), (5.11)-(5.14) получаем общую дисперсию погрешности вычислений.

На практике широко используют также дополнительные характеристики ошибок, получаемые из основных, выше приведенных. Так, например, максимальная вероятная ошибка:

$$\Delta_{max} = \gamma \sigma,$$

где коэффициент γ зависит от закона распределения ошибок.

Рассмотренные выше погрешности тесно связаны с основными параметрами вычислительных алгоритмов и техническими характеристиками вычислительной системы (рис.5.27).

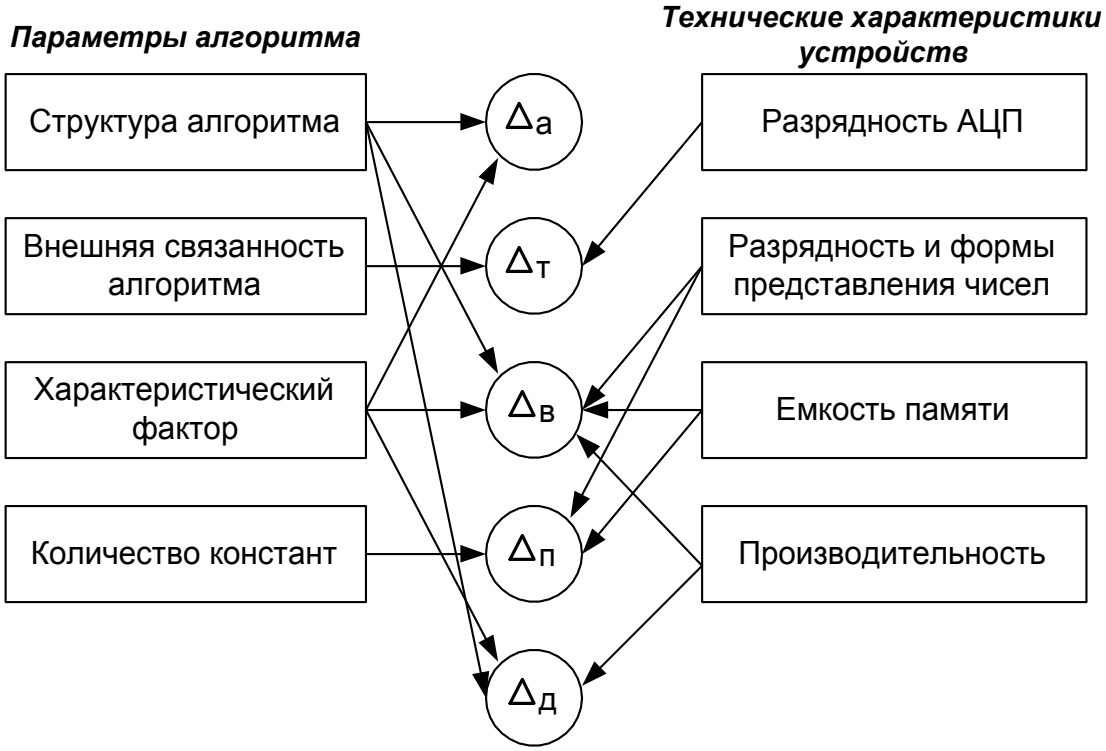


Рис.5.27. Структура формирования погрешностей в ИВС

Погрешность аппроксимации зависит от структуры алгоритма и влияет на характеристический вектор программы, на количество констант, на емкость памяти и требуемую производительность ВС. Трансформационная погрешность определяется разрядностью АЦП и влияет на объем входной информации. Вычислительная погрешность определяется, с одной стороны, количеством округляемых и масштабируемых арифметических операций и структурой алгоритма, а с другой - разрядностью и формой представления чисел. Чтобы уменьшить вычислительную погрешность, применив, например, программные методы повышения разрядности вычислителей, требуется дополнительная затрат памяти, машинного времени и усложнение программы, что ведет к связи вычислительной погрешности с объемом памяти и производительностью ВС. Параметрическая погрешность зависит от разрядности и формы представления чисел и влияет в ос-

новном на требуемую емкость памяти. Кроме того, величина параметрической погрешности определяет минимальную возможную величину констант и, следовательно, их количество. Динамическая погрешность связана со структурой алгоритма. Для ее уменьшения используются специальные алгоритмы (например, экстраполяция), что приводит к изменению характеристического вектора программы или увеличению производительности вычислителя.

5.9. Надежность и контроль информации в информационных системах

Причинами потерь работоспособности ИС могут быть ошибки в программах и неисправности в аппаратуре. Ошибки в программах в значительной степени исправляются в процессе эксплуатации системы, поэтому будем рассматривать ошибки, вызываемые только отказом аппаратуры.

Для обнаружения ошибок в выдаваемой информации в ИС должны быть предусмотрены средства обнаружения ошибок. Эти средства позволяют предотвратить использование недостоверной информации в принимаемых решениях.

Задачи контроля могут решаться программными и аппаратными средствами. При этом меняются затраты на решения этих задач. Другая задача обеспечения качества информации и сохранения данных осуществляется путем защиты информации. Вопросы защиты рассматриваются в специальной дисциплине.

1. *Контроль информации*

По характеру обслуживания информационные системы различают:

- ремонтируемые в процессе эксплуатации;
- неремонтируемые в процессе эксплуатации.

Вероятность исправной работы устройств системы $P_i(t)$ подчиняется случайному закону распределения. Примем, что плотность распределения отказов устройств в системе носит экспоненциальный закон распределения:

$$P_i(t) = \exp(-\lambda_i \cdot t),$$

где t - время работы устройств системы; λ_i - интенсивность отказа устройств.

Обычно при расчетах выбирается последовательная схема соединения устройств, как менее надежная, тогда:

$$P(t) = \prod_{i=1}^{n} P_i(t) = \exp(-\sum_{i=1}^{n} \lambda_i \cdot t),$$

где $\sum_{i=1}^{n} \lambda_i = \lambda$ - интенсивность отказов системы в целом.

Важным характеристическим свойством экспоненциального закона является то, что вероятность появления неисправности в системе на интервале времени [t, $t + \Delta t$] не зависит от времени предшествующей работы t, а зависит от длительности интервала Δt. Действительно:

$$P(t, t + \Delta t) = P(\Delta t) = \exp(-\lambda \cdot \Delta t).$$

Из опыта эксплуатации систем известно, что возникшая неисправность существует в течение некоторого периода времени $\Delta t_н$. После чего она может исчезнуть самостоятельно, либо в результате ремонта.

Для изучения влияния неисправностей на выходную информацию рассмотрим моменты возникновения неисправностей в течение времени вычислений. Рассмотрим на примере системы реального времени (рис. 5.28).

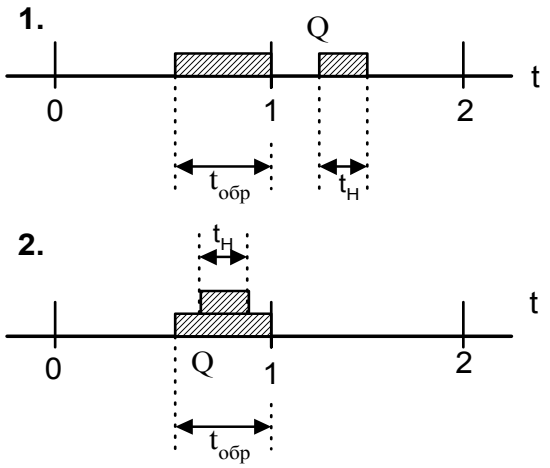


Рис. 5.28. Моменты возникновения неисправностей в процессе вычислений

Возможны следующие случаи возникновения неисправностей Q:

1) t < Q. Устройство исправно на интервале [0, t], следовательно, пользователю выдается достоверная информация. Вероятность отсутствия неисправностей в системе (исправная работа системы): $P(t) = P\{t < Q\}$

2) t > Q. Система неисправна на интервале [0, t]. Вероятность получения ошибки в выдаваемых данных определяется вероятностью неисправной работы системы $Q(t)$:

$$P(t) + Q(t) = 1,$$
$$Q(t) = 1 - P(t).$$

Если в ИС не принять меры по обнаружению ошибок в выдаваемой информации, то данные на выходе будут, либо верными с вероятностью $P(t)$, либо будут содержать ошибку с вероятностью $Q(t)$.

Для обнаружения ложной информации и предотвращения ее выдачи пользователям в системе применяются средства контроля.

Любой метод контроля характеризуется периодичностью включения $T_К$ и продолжительностью контроля $\tau_К$. При этом любой метод контроля требует дополнительных затрат ресурсов и времени.

Статистика показывает, что затраты на контроль могут достигать 10-15 % от стоимости всей аппаратуры [10].

В устройствах контроля могут также возникать неисправности, и с другой стороны устройства контроля могут не обнаруживать каких-либо ошибок в системе:

$$P^*(t) + Q_O^*(t) + Q_H^*(t) = 1,$$

где $P^*(t)$ – вероятность исправной работы устройства контроля; $Q^*_{О}(t)$ – вероятность неисправной работы, когда ошибка обнаруживается; $Q^*_{Н}(t)$ – необнаруженные ошибки в работе устройства контроля.

Рассмотрим возможные сочетания основной и контрольной информации, получаемой в один и тот же момент времени. Все случаи сведены в таблицу 5.1.

Таблица 5.1. Возможные сочетания основной и контрольной информации

Состояние аппаратуры контроля	Состояние основной аппаратуры		
	исправна	неисправна, ошибка не обнаруживаемая	неисправна, ошибка обнаруживаемая
Исправна	Ошибки нет PP^*	Ошибки нет $Q_н P^*$	Ошибка $Q_о P^*$
Неисправна, ошибка не обнаруживаемая	Ошибки нет $PQ_н^*$	Ошибки нет $Q_н Q_н^*$	Ошибки нет $Q_о Q_н^*$
Неисправна, ошибка обнаруживаемая	Ошибка $PQ_о^*$	Ошибка $Q_н\, Q_о^*$	Ошибка $Q_о Q_о^*$

Анализируя таблицу 5.1 можно найти вероятность того, что основная информация на выходе устройства с контролем будет верной:

$$\widetilde{P}(t) = P(t)P^*(t) + P(t)Q_H^*(t),$$
$$\widetilde{P}(t) = P(t)(P^*(t) + Q_H^*(t)) = P(t)(1 - Q_O^*(t)).$$

В последующих формулах для простоты записи упустим временной аргумент (t).

Вероятность того, что основная информация, выданная из устройства с контролем в систему, будет ложной. Это возможно в случае необнаруженного отказа собственно ИС $Q_Н$ при исправной работе средств контроля, а также в случае возникновения отказа самой ИС и необнаруженного отказа средств контроля:

$$\widetilde{Q}_H = Q_H P^* + QQ_H^*.$$

Обнаруживаемый отказ в работе системы имеет место, когда возникает сигнал ошибки, выдаваемый устройствами контроля. Он может быть в двух случаях:

1) независимо от работы собственно ИС, или
2) обнаружение отказа ИС и исправной работе средств контроля:

$$\widetilde{Q}_O = Q_O^* + Q_O P^*.$$

Анализ соотношений для ИС с контролем и без контроля позволяют сделать следующие выводы:

1) Введение контроля уменьшает вероятность выдачи ложной информации:

$\widetilde{Q}_H < Q$

$Q - \widetilde{Q}_H > 0;$

$Q - (Q_H.P^* + Q \cdot Q_H^*) = Q - Q \cdot Q_H^* - Q_H \cdot P^* = Q(1 - Q_H^*) - Q_H \cdot P^* =$

$= Q(P^* + Q_O^*) - Q_H \cdot P^* = Q \cdot P^* + Q \cdot Q_O^* - Q_H \cdot P^* = P^*(Q - Q_H) + Q \cdot Q_O^* > 0$

2) Повышается вероятность правильного функционирования ИВС, т.е. выдачи верной информации. Если при этом принять меры к исправлению ошибочной информации, то вероятность правильного функционирования системы с контролем определяется суммой:

$$\widetilde{P} + \widetilde{Q}_O = 1 - Q^{\sim}_{\text{н}}.$$

3) Уменьшается вероятность выдачи информации за счет включения аппаратуры контроля и появления ложных сигналов “ошибка” при неисправности аппаратуры контроля:

$$P - \widetilde{P} > 0\,.$$

Итак, контроль повышает достоверность информации, выдаваемой в систему за счет исключения неверной информации по сигналу «ошибка».

2. *Оценка эффективности способов контроля*

Эффективность контроля оценивается условной вероятностью того, что выданная информация будет верной. Эта характеристика называется достоверностью информации и вычисляется по формуле:

$$J = \frac{\widetilde{P}}{\widetilde{P} + \widetilde{Q}_H} = \frac{\widetilde{P}}{1 - \widetilde{Q}_O} = \frac{P(1 - Q_O^*)}{1 - Q_O^* - P^* \cdot Q_O} = \frac{P}{1 - \dfrac{Q_O \cdot P^*}{(1 - Q_O^*)}}.$$

Обозначим: $K = \dfrac{Q_O}{Q}$,

где K – характеристика ИС. Это условная вероятность обнаружения ошибки, показывает качество метода контроля;

$$h = \frac{P^*}{1 - Q_O^*},$$

где h - характеристика средств контроля, определяет достоверность контрольной информации.

Обобщенным показателем качества контроля является:

$$g=Kh=Q_oP^*/(Q(1-Q_o^*).$$

Достоверность информации характеризуется условной вероятностью правильного функционирования средств контроля.

Тогда:

$$J = \frac{P}{1 - QKh} = \frac{P}{1 - Qg}. \qquad (5.15)$$

Обобщенный показатель качества контроля $g = K \cdot h$ является условной вероятностью того, что ошибка работы ВС будет обнаружена и контролирующая аппаратура будет исправно функционировать в случае возникновения отказа собственно в ИС и отсутствии обнаруживаемой неисправности в средствах контроля.

Пример:

Сравнить надежность построения двух вариантов ИС:

1) $P = 0{,}8$; $h = 0$;

2) $P = 0{,}8$; $K = 0{,}75$; $P^* = 0{,}99$; $Q_O^* = 0{,}0075$.

Часто пользуются понятием недостоверности информации. Недостоверность информации вычисляется по формуле:

$$H = 1 - J \text{, или } H = Q\,(1-g)\,.$$

Для первого варианта g=0, в этом случае недостоверность информации будет равной:

$$H = Q = (1 - P) = 1 - 0{,}8 = 0{,}2\,,$$

т.е. 20% выдаваемой информации будет недостоверной.

Для второго варианта имеем:

$$h = \frac{P^*}{1 - Q_O^*} = \frac{0{,}99}{1 - 0{,}0075} \cong 0{,}996,$$
$$g = Kh = 0{,}75 \cdot 0{,}996 = 0{,}7481,$$
$$H = 0{,}2 \cdot (1 - 0{,}7481) = 0{,}05.$$

Изменение недостоверности информации будет вычисляться как разность:

$$\Delta H = 0{,}2 - 0{,}05 = 0{,}15 .$$

Достоверность выдаваемой информации в системе с контролем возросла на величину:

$$\frac{0{,}15}{0{,}2} \cdot 100\% = 75\% .$$

Некоторые выводы из формулы (5.15):

1) Если ВС не контролируется (K = 0), то в этом случае $J = P$. Достоверность информации численно равна вероятности исправной работы аппаратуры.

2) Контроль идеальный (K=1, h=1, g =1), тогда

$$J = \frac{P}{1 - Q} = \frac{P}{P} = 1 .$$

Система будет выдавать только верную информацию.

3) Если качество метода контроля идеально (K=1), но реализация этого метода неидеальна (h < 1), то обобщенный показатель качества контроля (g < 1) и достоверность информации на выходе системы J < 1.

4) При повышении обобщенного показателя качества контроля ($g \to 1$) достоверность выдаваемой информации увеличивается, т.е. $P \le J < 1$.

В общем случае система состоит из многих устройств. При этом некоторые устройства могут иметь контроль, а другие нет.

Рассмотрим данный случай. Предположим наихудший вариант, когда схема включения устройств последовательная. Устройства, не охваченные контролем: K_i = 0. Достоверность работы этих устройств равна:

$$Ji = \prod_{i=1}^{n} P_i \cdot$$

Пусть также есть другая группа устройств, которая содержит схемы контроля. Они имеют характеристику $\widetilde{P}_J$, тогда достоверность выдаваемой информации будет:

$$J_J = \frac{P_J}{1 - Q_J g_J}.$$

Надежность системы состоящей из устройств с контролем и без контроля будет:

$$\widetilde{P} = \prod_{i=1}^{n} P_i \cdot \prod_{j=1}^{m} \widetilde{P}_J .$$

По формуле достоверности работы системы получим:

$$J = \frac{P}{1 - \widetilde{Q}_O},$$

где $\widetilde{Q}_O = 1 - \prod_{j=1}^{m} (1 - \widetilde{Q}_{OJ})$. После подстановки получаем:

$$J = \prod_{i=1}^{n} P_i \cdot \frac{\prod_{j=1}^{m} \widetilde{P}_J}{\prod_{j=1}^{m} (1 - \widetilde{Q}_{OJ})} = \prod_{i=1}^{n} J_i \cdot \prod_{j=1}^{m} J_J \qquad (5.16)$$

Из формулы (5.16) видно, что наличие неконтролируемых устройств существенно снижает общую достоверность информации, выдаваемой системой.

Во многих случаях с целью повышения достоверности информации в одном и том же устройстве применяют несколько методов контроля с использованием различных независимых схем контроля.

5.10. Расчет распределения требуемой надежности. Методы распределения надежности. Резервирование устройств

Важным этапом выбора структуры ИС является распределение надежности по группам технических средств: процессоры, модули памяти, внешние устройства и т.д.

Понятие надежности связано с типом ИС. Например, для информационно-управляющих систем, работающих в реальном масштабе времени, требуется, чтобы выдаваемая управляющая информация поступала на объект в заданные моменты времени и чтобы она была верной.

В общем случае потеря работоспособности ИС может вызываться ошибками в программе, неисправностями в аппаратуре и ошибками обслуживающего персонала. В расчете надежности будем учитывать только те неисправности, которые вызываются аппаратурой.

Для исключения поступления недостоверной информации потребителям, в ИС должны предусматриваться средства обнаружения ошибок в выдаваемой информации с целью предотвращения использования недостоверной информации при управлении.

Под понятием надежности ИС понимается свойство системы выполнять заданные функции, сохраняя во времени значения установленных показателей в заданных пределах при заданных условиях эксплуатации.

Показателем надежности непрерывно функционирующих во времени ИС, служит вероятность правильного функционирования системы в течение времени t - $P(t)$. Время t во многом определяется условиями эксплуатации информационной системы, часто выбирается равным: t=100 ч.; t = 1000, 2000 ч. (для электронной аппаратуры); t = 5000 ч. (для компьютеров).

Кроме основной характеристики показателя надежности системы $P(t)$ вводят понятия среднего времени наработки на отказ, среднего времени восстановления системы и др.

1. *Расчет распределения требуемой надежности*

Для расчета будем рассматривать схему последовательного соединения устройств без резервирования, когда отказ любого из устройств приводит к отказу всей системы (рис.5.29):

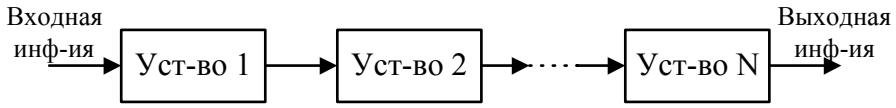


Рис. 5.29. Схема расчета надежности информационной системы

При решении поставленной задачи предполагается известным достигнутый уровень вероятности безотказной работы группы технических средств $P_i(t)$ $i = 1,2,...N$ и вводятся следующие допущения:

1) При решении любой задачи в ИС используются все технические средства системы;

2) Схема расчета надежности последовательная (как на рис.5.29);

3) Вероятность правильного решения задачи системой в заданном интервале времени $P_З(t)$ зависит только от правильной работы технических средств $P_З(t) = P(t)$;

4) Все задачи, решаемые системой, имеют одинаковую заданную вероятность правильного решения.

Тогда можем определить вероятность правильного решения задачи:

$$P_З(t) = P(t) = \prod_i P_i(t), \quad i = 1, 2, \ldots N.$$

Если ввести понятие вероятность неисправной работы группы устройств системы:

$$\mathrm{H}_i(t) = 1 - P_i(t),$$

тогда

$$P_З(t) = \prod_i (1 - H_i(t)).$$

Расчет распределения надежности заключается в нахождении таких значений характеристик устройств $H_1^*, H_2^*, \ldots, H_N^*$, чтобы обеспечивалось соотношение:

$$\prod_i (1 - H_i^*(t)) \geq P_З(t). \qquad (5.17)$$

Существует несколько принципов выбора значений $H_i^*(t)$:

1) Принцип равной надежности;

2) Принцип равных затрат. Этот принцип заключается в том, что средства на повышение надежности S должны быть распределены поровну между всеми группами технических средств;

3) Принцип обратной пропорциональности, в котором распределение средств на повышение надежности проводится обратно пропорционально величине удельных затрат;

4) Принцип минимума дополнительных затрат.

Принцип равной надежности

При равной надежности требуется получить соотношения:

$$H_1^* = H_2^* = \ldots = H_N^*.$$

Подставив данное соотношение в (5.17), получим:

$$\prod_i (1 - H_i^*) = P_З \ , \quad (1 - H_i^*)^N = P_З \ , \quad (1 - H_i^*) = \sqrt[N]{P_З} \ ,$$ откуда

$$H_i^* = 1 - \sqrt[N]{P_З} \ ,$$

где H_i^* - характеристики отдельных устройств системы.

Если *i*-я группа устройств имеет надежность $H_i(t)$, то при проектировании должны предусматриваться меры по ее уменьшению на величину:

$$\Delta H_i = H_i - H^* .$$

Различия в сложности и затратах на повышение надежности с различных уровней до одного уровня для разных технических средств при данном подходе не учитываются.

Принцип минимума дополнительных затрат

Требуется определить ΔH_i при минимальных затратах. Критерием оптимизации являются дополнительные затраты:

$$S = \sum_{i=1}^{N} S_i \ ,$$

где S_i – затраты на отдельное устройство.

Выделяемые средства на отдельные устройства можно выразить соотношением:

$$S = \sum_i \Delta H_i \delta_i \ ,$$

где ΔH_i – величина, показывающая насколько надо уменьшить показатели надежности; δ_i – удельные затраты.

При этом должно выполняться следующее условие:

$$\begin{aligned} H_i^* &= H_i - \Delta H_i^* \\ \prod_i (1 - H_i + \Delta H_i) &\succ P_З \end{aligned} .$$

После пренебрежения слагаемыми второго порядка малости получаем:

$$\begin{aligned} 1 - \sum_i H_i + \sum_i \Delta H_i &\succ P_З \\ \sum_i \Delta H_i &\succ P_З + \sum_i H_i - 1 \end{aligned} .$$

Задача заключается в нахождении минимума затрат при выполнении ограничений:

$$\begin{cases} \min \sum_i \Delta H_i \delta_i, \\ \sum_i \Delta H_i \succ P_З + \sum_i H_i - 1, \qquad i = 1, 2, ...N, \\ \Delta H_i \geq 0. \end{cases}$$

Функция цели и функция связи описываются линейными выражениями, поэтому для решения может использоваться метод линейного программирования.

Далее выбирается способ, обеспечивающий достижение требуемого уровня надежности для каждого устройства:

$$P_i^*(t) = (1 - \mathrm{H}_i^*(t)).$$

После этого уточняется структура КТС и параметры ИС.

2. *Расчет надежности невосстанавливаемых систем при общем резервировании*

Рассмотрим показатель надежности ИС при постоянном включении резервных устройств (рис.5.30)

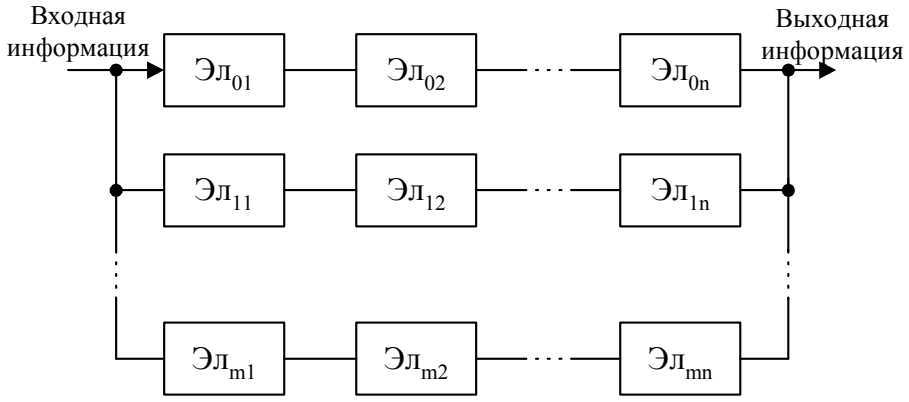


Рис.5.30. Структурная схема с постоянно включенными резервными устройствами

Основная цепь ИС содержит *n* устройств с вероятностью безотказной работы

$$P_{oi}(t), \text{ где } i = 1, 2,...n.$$

В случае отказа любого элемента основная цепь заменяется резервной, содержащей те же n устройств. Вероятность безотказной работы резервных устройств обозначим:

$$P_{ji}(t), \quad j = 1,2,...m,$$

где m – количество резервных цепочек.

Тогда общее число резервных устройств будет равно:

$$N = nm.$$

Рассмотрим случай использования идеальных ключевых устройств для коммутации резервных цепочек. Отказ системы с резервированием наступает при отказе основной и всех резервных цепочек.

Вероятность отказа данной системы будет равной:

$$Q = \prod_{j=0}^{m} Q_j = \prod_{j=0}^{m} (1 - P_j),$$

где P_j – вероятность безотказной работы одной цепочки.

Для цепи из n устройств имеем:

$$P_j(t) = \Pi P_{ij}(t),$$

где $P_{ij}(t)$ – вероятность безотказной работы i – го устройства j –ой цепочки.

После подстановки получим расчетную формулу:

$$Q = \prod_{j=0}^{m} (1 - \prod_{i=1}^{n} P_{ji}).$$

Вероятность безотказной работы для невосстанавливаемых систем с резервированием равна:

$$P = (1 - Q) = 1 - \prod_{j=0}^{m} (1 - \prod_{i=1}^{n} P_{ji}).$$

С увеличением кратности резервирования среднее время безотказной работы системы возрастает, но очень медленно. Наибольший прирост наблюдается при переходе от не резервируемой системы к резервируемой с кратностью резервирования, равной m=1.

Рассмотренное резервирование целесообразно применять при резервировании сравнительно простых систем.

3. *Особенности расчета надежности невосстанавливаемых систем при поэлементном резервировании*

Структурная схема ИС приведена на рисунке 5.31. Система состоит из n резервированных последовательно соединенных секций. Каждая секция содержит (m+1) параллельно включенных устройств.

Система исправна при исправности всех n секций. Отказ секции возникает при одновременном отказе всех (m+1) устройств секции.

В расчетах принимаем идеальными коммутирующие ключи.

Вероятность исправной работы системы будет определяться вероятностью исправной работой всех секций:

$$P = \prod_{i}^{n} P_i \ ,$$

где P_i – вероятность исправной работы секции.

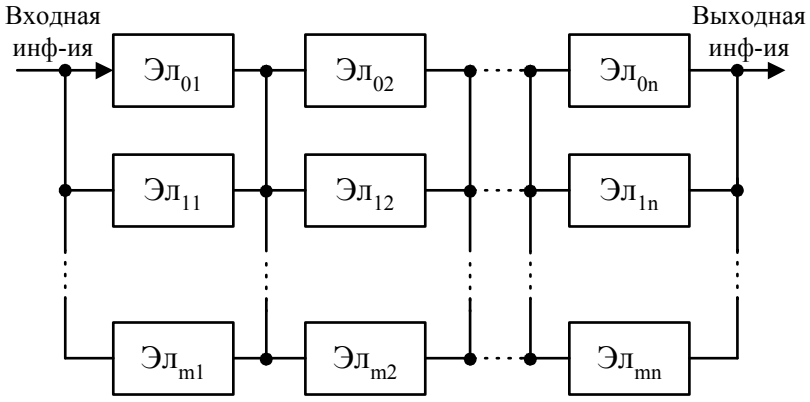


Рис.5.31. Схема поэлементного резервирования устройств системы

Вероятность исправной работы секции равна:

$$P_i = 1 - \prod_{j=0}^{m} (1 - P_{ij}) \ .$$

Тогда вероятность исправной работы системы будет равной:

$$P = \prod_{i=1}^{n} (1 - \prod_{j=0}^{m} (1 - P_{ij})) .$$

При равной надежности основных и резервных устройств $P_{ij}(t)=p_i(t)$ расчетная формула упрощается:

$$P(t) = \prod (1 - (1 - p_i(t)^{m+1}) , \qquad i = 1, 2, \ldots n.$$

Вероятность исправной работы ИС с резервированием каждого устройства выше вероятности исправной работы системы с общим резервированием.

В ИС используются также смешанные схемы соединения устройств. В таких схемах сначала рассчитывается вероятность безотказной работы ее составных частей, образующих систему основного соединения, а затем находиться вероятность безотказной работы всей системы.

Данные по надежности приводятся в технических характеристиках на устройства. При задании надежностных характеристик отдельных устройств системы в виде интенсивности отказов λ, в предположении экспоненциального закона распределения вероятностей отказов, надежностные характеристики $P(t)$ можно получить расчетным путем по формуле:

$$P_i(t) = \exp(-\lambda_i \cdot t) .$$

МЕТОДОЛОГИИ И ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

6.1. Жизненный цикл программного обеспечения информационных систем

Жизненный цикл программного обеспечения (ЖЦ ПО) определяется как период времени, который начинается с момента принятия решения о необходимости создания ПО и заканчивается в момент его полного изъятия из эксплуатации. Основным нормативным документом, регламентирующим состав процессов ЖЦ ПО, является международный стандарт *ISO/IEC* 12207:1995 «*Information Technology–Software Life Cycle Processes*». Он определяет структуру ЖЦ, содержащую процессы, действия и задачи, которые должны быть выполнены во время создания ПО (его российский аналог ГОСТ Р ИСО/МЭК 12207-99 введен в действие в июле 2000 г.). В данном стандарте процесс определяется как совокупность взаимосвязанных действий, преобразующих некоторые входные данные в выходные. Каждый процесс характеризуется определенными задачами и методами их решения, исходными данными, полученными от других процессов, и результатами.

В соответствии со стандартом ГОСТ Р ИСО/МЭК 12207-99 все процессы ЖЦ ПО разделены на три группы (рис. 6.1) [12].

1. Основные процессы ЖЦ ПО

Процесс приобретения состоит из действий и задач заказчика, приобретающего ПО. Данный процесс охватывает следующие действия: инициирование приобретения; подготовку заявочных предложений; подготовку и корректировку договора; надзор за деятельностью поставщика; приемку и завершение работ.

Процесс поставки охватывает действия и задачи, выполняемые поставщиком, который снабжает заказчика программным продуктом или услугой. Данный процесс включает следующие действия:

1) инициирование поставки;
2) подготовку ответа на заявочные предложения;
3) подготовку договора;

4) планирование;
5) выполнение и контроль;
6) проверку и оценку;
7) поставку и завершение работ.

Рис. 6.1. Процессы жизненного цикла программного обеспечения

Процесс разработки предусматривает действия и задачи, выполняемые разработчиком и охватывает работы по созданию ПО и его компонентов в соответствии с заданными требованиями, включая оформление проектной и эксплуатационной документации, подготовку материалов, необходимых для проверки работоспособности и соответствующего качества

программных продуктов, материалов, необходимых для организации обучения персонала, и т.д.

Процесс разработки включает следующие действия:

1) подготовительную работу;
2) анализ требований к системе;
3) проектирование архитектуры системы;
4) анализ требований к ПО;
5) проектирование архитектуры ПО;
6) детальное проектирование ПО;
7) кодирование и тестирование ПО;
8) интеграцию ПО;
9) квалификационное тестирование ПО;
10) интеграцию системы;
11) квалификационное тестирование системы;
12) установку ПО;
13) приемку ПО.

Процесс эксплуатации охватывает действия и задачи оператора организации, эксплуатирующей систему. Данный процесс включает следующие действия:

1) подготовительную работу;
2) эксплуатационное тестирование;
3) эксплуатацию системы;
4) поддержку пользователей.

Процесс сопровождения предусматривает действия и задачи, выполняемые сопровождающей организацией (службой сопровождения). Данный процесс активизируется при изменениях (модификациях) программного продукта и соответствующей документации, вызванных возникшими проблемами или потребностями в модернизации либо адаптации ПО. В соответствии со стандартом *IEEE*-90 под сопровождением понимается внесение изменений в ПО в целях исправления ошибок, повышения производительности или адаптации к изменившимся условиям работы или требованиям.

2. Вспомогательные процессы ЖЦ ПО

Процесс документирования предусматривает формализованное описание информации, созданной в течение ЖЦ ПО. Данный процесс состоит из набора действий, с помощью которых планируют, проектируют, разрабатывают, выпускают, редактируют, распространяют и сопровождают до-

кументы, необходимые для всех заинтересованных лиц, таких, как руководство, технические специалисты и пользователи системы.

Процесс документирования включает следующие действия:

1) подготовительную работу;
2) проектирование и разработку;
3) выпуск документации;
4) сопровождение.

Процесс управления конфигурацией предполагает применение административных и технических процедур на всем протяжении ЖЦ ПО для определения состояния компонентов ПО в системе, управления модификациями ПО, описания и подготовки отчетов о состоянии компонентов ПО и запросов на модификацию, обеспечения полноты, совместимости и корректности компонентов ПО, управления хранением и поставкой ПО. Согласно стандарту *IEEE*-90 под конфигурацией ПО понимается совокупность его функциональных и физических характеристик, установленных в технической документации и реализованных в ПО.

Процесс обеспечения качества предусматривает соответствующие гарантии того, что ПО и процессы его ЖЦ соответствуют заданным требованиям и утвержденным планам. Под качеством ПО понимается совокупность свойств, которые характеризуют способность ПО удовлетворять заданным требованиям.

Для получения достоверных оценок создаваемого ПО процесс обеспечения его качества должен происходить независимо от субъектов, непосредственно связанных с разработкой ПО. При этом могут использоваться результаты других вспомогательных процессов, таких, как верификация, аттестация, совместная оценка, аудит и разрешение проблем.

Процесс обеспечения качества включает следующие действия:

1) подготовительную работу;
2) обеспечение качества продукта;
3) обеспечение качества процесса;
4) обеспечение прочих показателей качества системы.

Процесс верификации состоит в определении того, что программные продукты, являющиеся результатами некоторого действия, полностью удовлетворяют требованиям или условиям, обусловленным предшествующими действиями (верификация в узком смысле означает формальное доказательство правильности ПО). Для повышения эффективности верификация должна как можно раньше интегрироваться с использующими ее

процессами (такими, как поставка, разработка, эксплуатация или сопровождение). Данный процесс может включать анализ, оценку и тестирование.

Верификация может проводиться с различными степенями независимости. Степень независимости может варьироваться от выполнения верификации самим исполнителем или другим специалистом данной организации до ее выполнения специалистом другой организации с различными вариациями. Если процесс верификации осуществляется организацией, не зависящей от поставщика, разработчика, оператора или службы сопровождения, то он называется процессом независимой верификации.

Процесс верификации включает следующие действия:

1) подготовительную работу;
2) верификацию.

Процесс аттестации предусматривает определение полноты соответствия заданных требований и созданной системы или программного продукта их конкретному функциональному назначению. Под аттестацией обычно понимается подтверждение и оценка достоверности проведенного тестирования ПО.

Процесс совместной оценки предназначен для оценки состояния работ по проекту и ПО, создаваемому при выполнении данных работ (действий). Он сосредоточен в основном на контроле планирования и управления ресурсами, персоналом, аппаратурой и инструментальными средствами проекта.

Оценка применяется как на уровне управления проектом, так и на уровне технической реализации проекта и проводится в течение всего срока действия договора. Данный процесс может выполняться двумя любыми сторонами, участвующими в договоре, при этом одна сторона проверяет другую.

Процесс совместной оценки включает следующие действия:

1) подготовительную работу;
2) оценку управления проектом;
3) техническую оценку.

Процесс аудита представляет собой определение соответствия требованиям, планам и условиям договора. Аудит может выполняться двумя любыми сторонами, участвующими в договоре, когда одна сторона проверяет другую.

Аудит - это ревизия (проверка), проводимая компетентным органом (лицом) в целях обеспечения независимой оценки степени соответствия

ПО или процессов установленным требованиям. Аудит служит для установления соответствия реальных работ и отчетов требованиям, планам и контракту. Аудиторы (ревизоры) не должны иметь прямой зависимости от разработчиков ПО. Они определяют состояние работ, использование ресурсов, соответствие документации спецификациям и стандартам, корректность тестирования.

Процесс разрешения проблем предусматривает анализ и решение проблем (включая обнаруженные несоответствия), независимо от их происхождения или источника, которые обнаружены в ходе разработки, эксплуатации, сопровождения или других процессов. Каждая обнаруженная проблема должна быть идентифицирована, описана, проанализирована и разрешена.

Процесс разрешения проблем включает следующие действия:

1) подготовительную работу;
2) разрешение проблем.

3. Организационные процессы ЖЦ ПО

Процесс управления состоит из действий и задач, которые могут выполняться любой стороной, управляющей своими процессами. Данная сторона (менеджер) отвечает за управление выпуском продукта, управление проектом и задачами соответствующих процессов, таких, как приобретение, поставка, разработка, эксплуатация, сопровождение и др.

Процесс управления включает следующие действия:

1) инициирование и определение области управления;
2) планирование;
3) выполнение и контроль;
4) проверку и оценку;
5) завершение.

Процесс создания инфраструктуры охватывает выбор и поддержку (сопровождение) технологии, стандартов и инструментальных средств, выбор и установку аппаратных и программных средств, используемых для разработки, эксплуатации или сопровождения ПО. Инфраструктура должна модифицироваться и сопровождаться в соответствии с изменениями требований к соответствующим процессам. Инфраструктура, в свою очередь, является одним из объектов управления конфигурацией.

Процесс создания инфраструктуры включает следующие действия:

1) подготовительную работу;
2) создание инфраструктуры;

3) сопровождение инфраструктуры.

Процесс усовершенствования предусматривает оценку, измерение, контроль и усовершенствование процессов ЖЦ ПО. Данный процесс включает следующие действия:

1) создание процесса;
2) оценку процесса;
3) усовершенствование процесса.

Усовершенствование процессов ЖЦ ПО направлено на повышение производительности труда всех участвующих в них специалистов за счет совершенствования используемой технологии, методов управления, выбора инструментальных средств и обучения персонала.

Процесс обучения охватывает первоначальное обучение и последующее постоянное повышение квалификации персонала. Приобретение, поставка, разработка, эксплуатация и сопровождение ПО в значительной степени зависят от уровня знаний и квалификации персонала. Должны быть разработаны и представлены методические материалы, необходимые для обучения пользователей в соответствии с учебным планом.

4. Модели ЖЦ ПО

Под моделью жизненного цикла ПО понимается структура, определяющая последовательность выполнения и взаимосвязи процессов, действий и задач на протяжении ЖЦ. Модель ЖЦ зависит от специфики, масштаба и сложности проекта и специфики условий, в которых система создается и функционирует.

Стандарт ГОСТ Р ИСО/МЭК 12207-99 не предлагает конкретную модель ЖЦ ПО. Его положения являются общими для любых моделей ЖЦ, методов и технологий создания ПО. Он описывает структуру процессов ЖЦ ПО, не конкретизируя в деталях, как реализовать, или выполнить действия и задачи, включенные в эти процессы.

4.1. Каскадная модель

В 1970 г. эксперт в области ПО Уинстон Ройс опубликовал получившую широкую известность статью, в которой он излагал свое мнение о методике, которая позднее получила название «модель водопада» (или «каскадная модель», рис. 6.2). Эта модель была разработана с учетом ограничений «тяжелых» процессов, налагавшихся на разработчиков государственными контрактами того времени. Многие ошибочно полагают, что в своей статье Ройс выступил в защиту однопроходной последовательной схемы. На самом же деле он рекомендовал подход, несколько отличный от того, который в конечном итоге вылился в концепцию «водопада» с ее строгой последовательностью стадий анализа требований, про-

ектирования и разработки. Принципиальными свойствами, так называемой, «чистой» каскадной модели являются следующие:

– фиксация требований к системе до ее сдачи заказчику;

– переход на очередную стадию проекта только после того, как будет полностью завершена работа на текущей стадии, без возвратов на пройденные стадии.

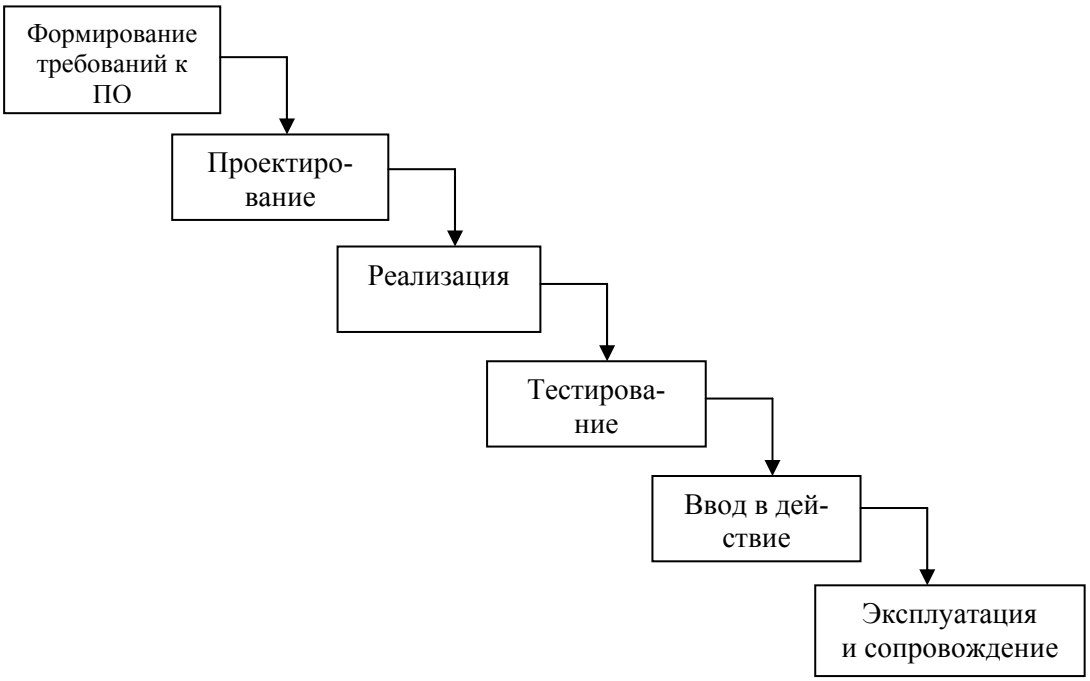


Рис. 6.2. Каскадная модель ЖЦПО

Каждая стадия заканчивается получением некоторых результатов, которые служат в качестве исходных данных для следующей стадии. Требования к разрабатываемому ПО, 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.

В то же время этот подход обладает рядом недостатков, вызванных, прежде всего тем, что реальный процесс создания ПО никогда полностью не укладывался в такую жесткую схему. Процесс создания ПО носит, как правило, итерационный характер: результаты очередной стадии часто вызывают изменения в проектных решениях, выработанных на более ранних стадиях. Таким образом, постоянно возникает потребность в возврате к предыдущим стадиям и уточнении или пересмотре ранее принятых решений. В результате реальный процесс создания ПО принимает вид, изображенный на рисунке 6.3.

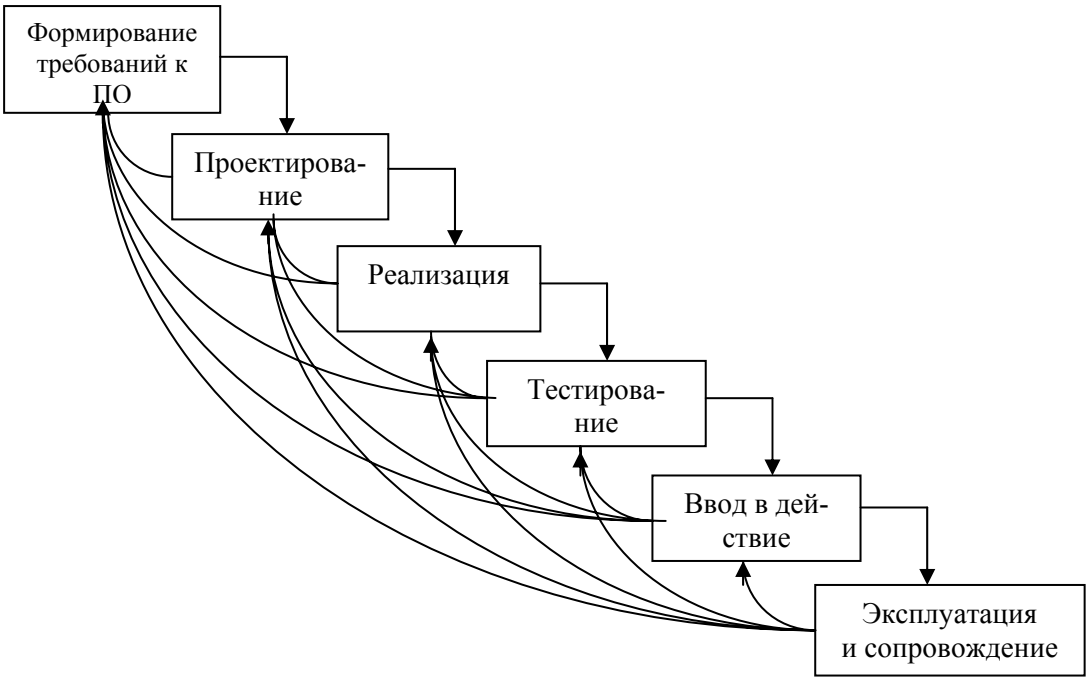


Рис. 6.3. Итерационная модель ЖЦПО

Основными недостатками каскадного подхода являются:

– позднее обнаружение проблем;

– выход из календарного графика, запаздывание с получением результатов;

– избыточное количество документации;

– невозможность разбить систему на части (весь продукт разрабатывается за один раз);

– высокий риск создания системы, не удовлетворяющей изменившимся потребностям пользователей.

4.2. Итерационная модель

В середине 1980-х годов Барри Боэм предложил вариант итерационной модели под названием «спиральная модель» (рис. 6.4).

Принципиальные особенности спиральной модели в следующем:

– отказ от фиксации требований и назначение приоритетов пользовательским требованиям;

– разработка последовательности прототипов, начиная с требований наивысшего приоритета;

– идентификация и анализ риска на каждой итерации;

– использование каскадной модели для реализации окончательного прототипа;

– оценка результатов по завершении каждой итерации и планирование следующей итерации.

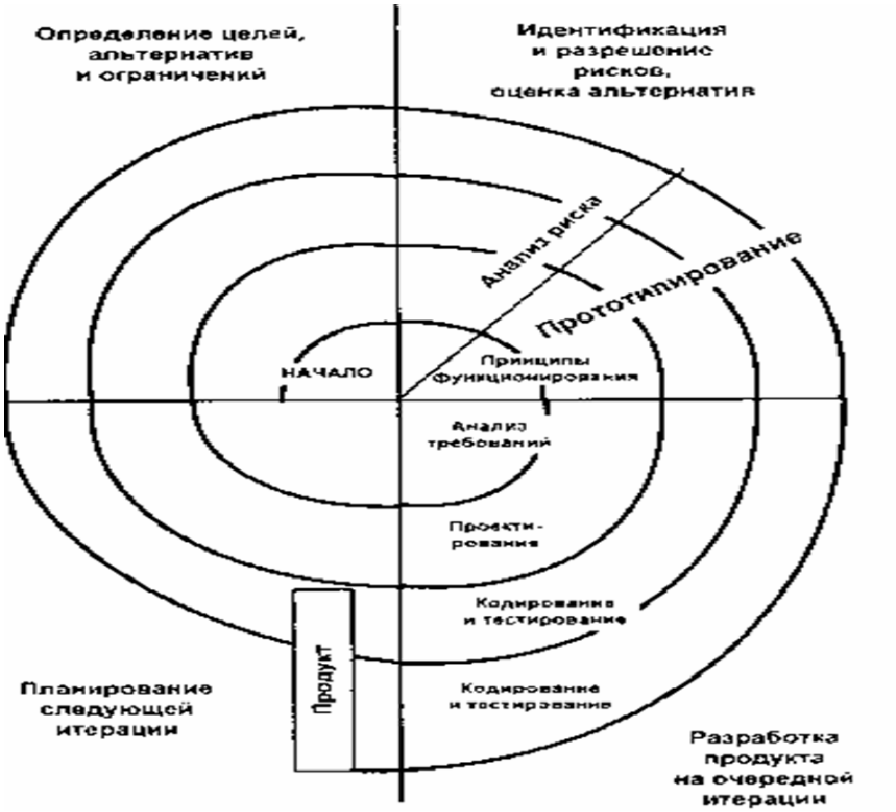


Рис. 6.4. Спиральная модель ЖЦПО

При использовании спиральной модели прикладное ПО создается в несколько итераций (витков спирали) методом прототипирования. Под прототипом понимается действующий программный компонент, реализующий отдельные функции и внешние интерфейсы разрабатываемого ПО. Создание прототипов осуществляется в несколько итераций, или витков спирали. 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 производится тщательная оценка риска превышения сроков и стоимости проекта, чтобы определить необходимость выполнения еще одной итерации, степень полноты и точности понимания требований к системе, а также целесообразность прекращения проекта.

Спиральная модель избавляет пользователей и разработчиков ПО от необходимости полного и точного формулирования требований к системе на начальной стадии, поскольку они уточняются на каждой итерации. Таким образом, углубляются и по-

следовательно конкретизируются детали проекта, и в результате выбирается обоснованный вариант, который доводится до реализации.

Достоинствами спиральной модели являются:

– ускорение разработки (раннее получение результата за счет прототипирования);

– постоянное участие заказчика в процессе разработки;

– разбиение большого объема работы на небольшие части;

– снижение риска (повышение вероятности предсказуемого поведения системы).

Спиральная модель не исключает использования каскадного подхода на завершающих стадиях проекта в тех случаях, когда требования к системе оказываются полностью определенными.

К недостаткам спиральной модели можно отнести:

– сложность планирования (определения количества и длительности итераций, оценки затрат и рисков);

– сложность применения модели с точки зрения менеджеров и заказчиков (из-за привычки к строгому и детальному планированию);

– напряженный режим работы для разработчиков (при краткосрочных итерациях).

6.2. Сертификация и оценка процессов создания программного обеспечения

1. Понятие зрелости процессов создания ПО. Модель оценки зрелости

Руководители организаций не всегда могут сформировать стратегию совершенствования и развития технологии деятельности своей компании; на рынке труда специалистов необходимой квалификации явно недостаточно. Вместе с тем в области совершенствования технологических процессов разработки и эксплуатации ПО международный опыт долгие годы был недостаточно обобщен и формализован [12]. Только в начале 1990-х годов американский Институт программной инженерии (*SEI*) сформировал модель технологической зрелости организаций *CMM* (*Capability Maturity Model*), определив уровни технологической зрелости и их отличительные черты. В течение десятилетия *CMM* прошла апробацию в целом ряде организаций, ее эффективность и достоверность проверили заказывающие организации, поставщики ПО, компании, осуществляющие разработку заказного ПО, занимающиеся оффшорным программированием.

Зрелость процессов (*software process maturity*) - это степень их управляемости, контролируемости и эффективности. Повышение технологической зрелости оз-

начает потенциальную возможность возрастания устойчивости процессов и указывает на степень эффективности и согласованности использования процессов создания и сопровождения ПО в рамках всей организации. Реальное использование процессов невозможно без их документирования и доведения до сведения персонала организации, без постоянного контроля и совершенствования их выполнения. Возможности хорошо продуманных процессов полностью определены. Повышение технологической зрелости процессов означает, что эффективность и качество результатов их выполнения могут постоянно возрастать. В организациях, достигших технологической зрелости, процессы создания и сопровождения ПО принимают статус стандарта, фиксируются в организационных структурах и определяют производственную тактику и стратегию. Введение их в статус закона влечет за собой необходимость построения инфраструктуры и создания требуемой корпоративной культуры производства, которые обеспечивают поддержку соответствующих методов, операций и процедур ведения дел даже после того, как из организации уйдут те, кто все это создал. *CMM* представляет собой методический материал, определяющий правила формирования системы управления созданием и сопровождением ПО и методы постепенного и непрерывного повышения культуры производства. Назначение *CMM* - предоставление организациям-разработчикам необходимых инструкций по выбору стратегии повышения качества процессов путем анализа степени их технологической зрелости и факторов, в наибольшей степени влияющих на качество выпускаемой продукции. На рисунке 6.5 показаны пять уровней технологической зрелости *CMM*. Надписи на стрелках определяют особенности совершенствования процессов при переходе с уровня на уровень.

На уровне 1 (начальном) основные процессы создания и сопровождения ПО носят случайный характер и выполняются хаотично. Успех выполнения проекта всецело зависит от индивидуальных усилий персонала. На этом уровне, как правило, в организации не существует стабильной среды, необходимой для создания и сопровождения ПО.

На уровне 2 (уровне повторяемых процессов) процессы управления проектом позволяют обеспечивать текущий контроль стоимости проекта, графика его выполнения и выполняемых функций. Дисциплина выполнения проекта такова, что существует возможность прогнозирования успешности выполнения проектов по созданию аналогичных программных продуктов. Каждый уровень *CMM*, начиная со второго, характеризуется наличием ряда так называемых основных групп процессов (*key process areas - KPA*). Модель *CMM* содержит 18 таких групп, последняя версия модели *CMMI* - 20 групп.

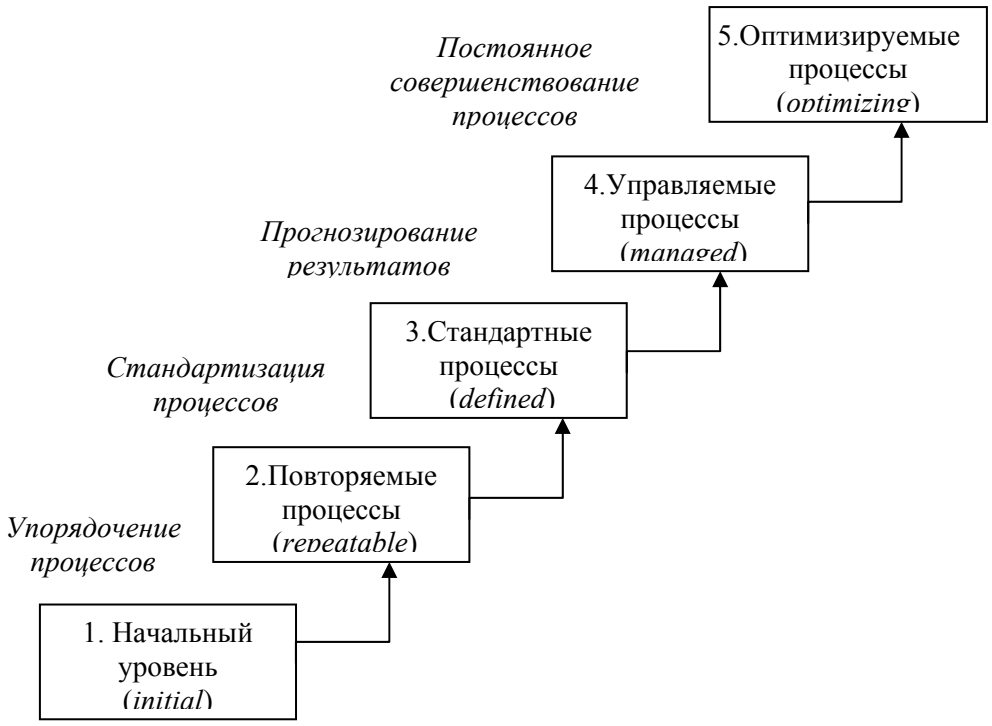


Рис. 6.5. Пять уровней технологической зрелости CCM.

Уровень 2 *CMM* характеризуется наличием в организации следующих процессов (их наименования соответствуют *CMMI*):

- управление требованиями;
- управление конфигурацией;
- планирование проекта;
- мониторинг и контроль проекта;
- управление контрактами;
- измерения и анализ;
- обеспечение качества процесса и продукта.

На уровне 3 (уровне стандартизованных процессов) процессы создания ПО документированы, стандартизованы и представляют собой единую технологическую систему, обязательную для всех подразделений организации. Во всех проектах используется опробованная, утвержденная и возведенная в статус стандарта единая технология создания и сопровождения ПО. На данном уровне к процессам уровня 2 добавляются следующие процессы:

- спецификация требований;
- интеграция продукта;
- верификация;
- аттестация;

– стандартизация процессов организации;
– обучение;
– интегрированное управление проектом;
– управление рисками;
– анализ и принятие решений.

Главное назначение уровня 4 (уровня управляемых процессов) - текущий контроль над процессами. Управление должно обеспечивать выполнение процессов в рамках заданного качества. Могут быть неизбежные потери и временные пики в измеряемых результатах, которые требуют вмешательства, но в целом система должна быть стабильна.

На уровне 4 добавляются следующие процессы:
– управление производительностью и продуктивностью;
– количественное управление проектом.

Основное назначение уровня 5 (уровня оптимизируемых процессов) - последовательное усовершенствование и модернизация процессов создания и сопровождения ПО. В целях плановой модернизации технологии создания ПО в организации создается специальное подразделение, основными обязанностями которого являются сбор данных по выполнению процессов, их анализ, модернизация имеющихся и создание новых процессов, их проверка на пилотных проектах и придание им статуса стандартов предприятия.

На уровне 5 добавляются следующие процессы:
– внедрение технологических и организационных инноваций;
– причинно-следственный анализ и разрешение проблем.

К недостаткам *CMM* относятся следующее:

1) Модель сосредоточена исключительно на управлении проектом, а не на процессе создания программного продукта. В модели не учтены такие важные факторы, как использование определенных методов, например прототипирования, формальных и структурных методов, средств статического анализа и т.п.

2) В модели отсутствует анализ рисков и решений, что не позволяет обнаруживать проблемы прежде, чем они окажут воздействие на процесс разработки.

3) Не определена область применения модели, хотя авторы признают, что она является универсальной и подходящей всем организациям. Однако авторы не дают четкого разграничения организаций, которые могут или не могут внедрять *CMM* в свою деятельность. Небольшие компании находят эту модель слишком бюрократичной. В ответ на эту критику были разработаны стратегии

совершенствования технологического процесса для малых организаций.

Главной целью создания модели *CMM* было намерение Министерства обороны США оценить возможности поставщиков ПО. На данный момент пока не существует четких требований к достижению определенного уровня развития организаций-разработчиков. Однако принято считать, что у организации, достигшей высокого уровня, больше шансов выиграть тендер на поставку ПО. В качестве альтернативы *CMM* предлагается обобщенная классификация процессов совершенствования технологической зрелости, которая подходит для большинства организаций и программных проектов. Можно выделить несколько общих типов процессов совершенствования.

1) Неформальный процесс. Не имеет четко выраженной модели совершенствования. Его с успехом может использовать отдельная команда разработчиков. Неформальность процесса не исключает таких формальных действий, как управление конфигурацией, однако при этом сами действия и их взаимосвязи не предопределены заранее.

2) Управляемый процесс. Имеет подготовленную модель, которая управляет процессом совершенствования. Модель определяет действия, их график и взаимосвязи между ними.

3) Методически обоснованный процесс. Подразумевается, что введены в действие определенные методы (например, систематически применяются методы объектно-ориентированного проектирования). Для процессов этого типа будут полезными инструментальные средства поддержки проектирования и анализа процессов (*CASE*-средства).

4) Процесс непосредственного совершенствования. Имеет четко поставленную цель совершенствования технологического процесса, для чего существует отдельная строка в бюджете организации и определены нормы и процедуры внедрения нововведений. Частью такого процесса является количественный анализ процесса совершенствования.

Многие технологические процессы в настоящее время имеют *CASE*-средства поддержки, поэтому их можно назвать поддерживаемыми процессами. Методически обоснованные процессы поддерживаются инструментальными средствами анализа и проектирования. Эффективность средства поддержки зависит от применяемого процесса совершенствования. Например, в неформальном процессе могут использоваться типовые средства поддержки (средства прототипирования, компиляторы, средства отладки, текстовые процессоры и т.п.). Вряд ли в неформальных процессах будут использоваться на постоянной основе более специализированные средства поддержки.

2. *Методика SPMN*

Большинство современных технологий разработки ПО основаны на принципе улучшения процесса разработки, когда наибольшую свободу действий в выборе способа реализации имеет руководитель проекта. В то же время признанные мировые лидеры, создающие качественное ПО, активно используют принцип лучшего практического навыка. Этот принцип ориентирован на улучшение деталей работы и быстрое достижение конечного результата. В нем нет абстрактного «улучшения процесса», а есть конкретные рекомендации, использующие числовые характеристики проекта. Другое преимущество принципа лучших практических навыков - возможность их немедленного применения в противовес «тяжелой» модели *CMM*, для сертификации по которой нужны годы труда. Несмотря на определенное число очень успешных результатов внедрения *CMM*, эта методика не получила массового признания среди небольших фирм в силу сложности и слишком больших усилий, требуемых для ее внедрения. Продолжающиеся неудачи в крупных программных проектах заставили Министерство обороны США сформировать подразделение *SPMN* (*Software Program Managers Network*), которое было призвано помочь военным быстро наладить эффективные процессы управления проектами в организациях-разработчиках ПО.

Подход, предложенный отделом *SPMN*, называется *CBP* (*Critical Best Practices* - критически важные практические навыки). Он позволяет тактическими изменениями в работе организации очень быстро (за полтора-два года) примерно на 80% достичь третьего уровня *CMM* (на что обычно требуется около десяти лет). При этом подход *CBP* проверен на сотнях реальных крупных программных проектов.

Рекомендации по применению практических навыков достаточно очевидны. В самом общем виде подход *CBP* предлагает:

- сфокусироваться на количественных параметрах завершения проекта (дате, бюджете, объеме);
- придумать быстро реализуемую стратегию выполнения проекта;
- измерять продвижение к цели;
- измерять активность разработки.

Результаты работы *SPMN* показали, что ход выполнения крупных проектов обычно находится на грани хаоса, и существует ряд факторов, от которых зависит, перейдет ли система в неуправляемое состояние. Чтобы правильно управлять проектом, надо придерживаться следующих принципов:

1) Ошибки и логические неувязки надо выявлять как можно раньше и

устранять сразу после обнаружения. Между внесением ошибки разработчиком и ее выявлением должно пройти минимальное время (в проектах Министерства обороны США среднее время между внесением ошибки и ее устранением составляло 9 месяцев). Практика почасовой оплаты программистов (имевшая место при выполнении госзаказов в США) совершенно недопустима. Надо также совершенствовать механизмы выявления типичных причин ошибок и способы их устранения.

2) Необходимо планировать работу на основе правильно выбранных показателей. Невозможно реализовать крупный проект, если не подготовить в его рамках максимально подробный план всех видов деятельности с учетом производительности сотрудников, объема проекта, бюджета и других ресурсов.

3) Надо минимизировать неконтролируемые изменения проекта с учетом того, что они вносятся разработчиками на всех этапах, начиная с требований к системе и заканчивая ее пользовательскими интерфейсами.

4) Необходимо эффективно использовать сотрудников. Знания, опыт и мотивация сотрудников - важнейшие факторы успеха. Акцент в управлении проектами должен быть смещен на производительность труда, качество работы, выполнение планов и удовлетворение пользователя. Для этого требуются большие усилия по подготовке профессиональных руководителей проектов и изменения текущих способов их подготовки.

Девять лучших навыков, рекомендованных SPMN.

Навык 1 - формальное управление рисками. Любой проект по разработке ПО - рискованный. Но отсутствие процедуры управления рисками в компании - это, пожалуй, самый показательный признак грядущей неудачи проекта. Поэтому необходимо уметь определять риск превышения бюджета и времени выполнения, неверного выбора и возможного отказа оборудования, ошибок программирования и плохого сопровождения. Риск оценивается по вероятности возникновения и его последствиям.

Навык 2 - соглашения об интерфейсах: пользовательских, внутренних (межмодульных) и внешних (для стыковки с другими компонентами и приложениями).

Интерфейсы программы - это необходимая часть системных требований и ее архитектуры, но руководители проектов часто забывают контролировать соответствие продукта этим соглашениям. Чем позже будут определены соглашения об интерфейсах, тем больше вероятность того, что систему придется заново проектировать, программировать и тестировать.

Навык 3 - формальные проверки проекта. Нередко устранение ошибок начинается, только когда проект переходит к этапу тестирования. Такие этапы были при-

думаны 30 лет назад для создания небольших по сегодняшним меркам систем, и хотя в тестировании нет ничего плохого, выделять его в отдельный этап методически неверно. Стоимость этапа тестирования может достигать 40-60% стоимости всего проекта.

Навык 4 - управление проектом на основе метрик. Этот навык нужен для раннего обнаружения потенциальных проблем. Как уже говорилось, стоимость устранения дефекта в проекте увеличивается в геометрической прогрессии по мере роста проекта.

Навык 5 - качество продукта должно контролироваться на глубоком уровне. Проблема реализации мелких деталей программного проекта очень важна при разработке ПО. Иногда мелкое на первый взгляд требование заказчика выливается в глобальную переделку проекта. Если же проект спланирован недостаточно подробно, обсуждать реальное положение дел в ходе его выполнения бессмысленно.

Навык 6 - информация о ходе проекта должна быть общедоступной. Чем больше сотрудников вовлечено в процесс контроля над ходом проекта, тем проще идентифицировать потенциальные проблемы и риски. Надо сделать показатели хода проекта доступными всем сотрудникам и заказчику и организовать канал приема анонимных сообщений о возникающих проблемах. Чаще всего такой канал используется для сведения личных счетов, но лучше получить ложный сигнал, чем не узнать о реальной проблеме. К тому же открытость проекта — это залог снижения числа ложных сообщений.

Навык 7 - чтобы добиться высокого качества, надо отслеживать причины возникновения ошибок. Эффективность работы компании непосредственно зависит от наличия ошибок в проекте. Большинство компаний не контролируют их реальные источники: ошибки программистов, отклонения в графиках выполнения работ, превышение планируемой стоимости, неверно сформулированные требования, неправильно подготовленную документацию и плохо обученный персонал. В каждой фазе проекта ошибки должны отслеживаться формально. Для этого желательно использовать средства конфигурационного управления. Каждый случай обнаружения и устранения ошибки обязательно надо документировать.

Навык 8 - управление конфигурацией. Неконтролируемые изменения в проекте могут быстро ввергнуть его в хаос. Поэтому на практике надо руководствоваться двумя простыми правилами:

– любую информацию, которую использует более чем один сотрудник, надо контролировать с помощью системы управления конфигурацией;

– любую информацию, учитываемую системой качества, надо контро-

лировать с помощью системы управления конфигурацией.

Навык 9 - управление персоналом. Главный фактор успеха проекта - качество, опыт и мотивация сотрудников. Не надо забывать, что с помощью различных методик производительность труда программистов можно значительно повысить. К тому же, как бы подробно ни документировался проект, некоторые детали его архитектуры всегда хранятся только в головах разработчиков, и руководитель проекта должен помогать сотрудникам проявлять индивидуальные творческие способности.

6.3. Общие принципы проектирования программного обеспечения

– Как было отмечено ранее, одной из основных проблем, которые приходится решать при создании больших и сложных систем любой природы, в том числе и ПО, является проблема сложности. Ни один разработчик не в состоянии выйти за пределы человеческих возможностей и понять всю систему в целом. Единственный эффективный подход к решению этой проблемы заключается в построении сложной системы из небольшого количества крупных частей, каждая из которых, в свою очередь, строится из частей меньшего размера, и т.д., до тех пор, пока самые небольшие части можно будет строить из имеющегося материала [12]. По отношению к проектированию сложной программной системы это означает, что ее необходимо разделить (декомпозировать) на небольшие подсистемы, каждую из которых можно разрабатывать независимо от других. Это позволяет при разработке подсистемы любого уровня иметь дело только с ней, а не со всеми остальными частями системы. Правильная декомпозиция является главным способом преодоления сложности разработки больших систем ПО. Понятие «правильная» по отношению к декомпозиции означает следующее:

– количество связей между отдельными подсистемами должно быть минимальным (принцип «слабой связанности»);

– связность отдельных частей внутри каждой подсистемы должна быть максимальной (принцип «сильного сцепления»).

Структура системы должна быть такой, чтобы все взаимодействия между ее подсистемами укладывались в ограниченные, стандартные рамки, т.е.:

– каждая подсистема должна инкапсулировать свое содержимое (скрывать его от других подсистем);

– каждая подсистема должна иметь четко определенный интерфейс с другими подсистемами.

Инкапсуляция (принцип «черного ящика») позволяет рассматривать структуру каждой подсистемы независимо от других подсистем. Интерфейсы позволяют строить систему более высокого уровня, рассматривая каждую подсистему как единое целое и игнорируя ее внутреннее устройство.

Существуют два основных подхода к декомпозиции систем. Первый подход называют функционально-модульным, он является частью более общего структурного подхода. В его основу положен принцип функциональной декомпозиции, при которой структура системы описывается в терминах иерархии ее функций и передачи информации между отдельными функциональными элементами. Второй, объектно-ориентированный подход, использует объектную декомпозицию. При этом структура системы описывается в терминах объектов и связей между ними, а поведение системы описывается в терминах обмена сообщениями между объектами.

В 1970-1980 годах при разработке ПО достаточно широко применялись структурные методы, предоставляющие в распоряжение разработчиков строгие формализованные методы описания проектных решений - спецификаций ПО (в настоящее время такое же распространение получают объектно-ориентированные методы). Эти методы основаны на использовании наглядных графических моделей: для описания архитектуры ПО с различных точек зрения (как статической структуры, так и динамики поведения системы) используются схемы и диаграммы. Наглядность и строгость средств структурного и объектно-ориентированного анализа позволяет разработчикам и будущим пользователям системы с самого начала неформально участвовать в ее создании, обсуждать и закреплять понимание основных технических решений. Однако широкое применение этих методов и следование их рекомендациям при разработке конкретных систем ПО сдерживалось отсутствием адекватных инструментальных средств, поскольку при неавтоматизированной (ручной) разработке все их преимущества практически сведены к нулю. Действительно, вручную очень трудно разработать и графически представить строгие формальные спецификации системы, проверить их на полноту и непротиворечивость и тем более изменить. Если все же удается создать строгую систему проектных документов, то ее переработка при появлении серьезных изменений практически неосуществима. Ручная разработка обычно порождала следующие проблемы:

- неадекватная спецификация требований;
- неспособность обнаруживать ошибки в проектных решениях;
- низкое качество документации, снижающее эксплуатационные характеристики;

– затяжной цикл и неудовлетворительные результаты тестирования.

Перечисленные факторы способствовали появлению программно-технологических средств специального класса - *CASE*-средств, реализующих *CASE*-технологию создания ПО. Понятие *CASE* (*Computer Aided Software Engineering*) используется в настоящее время в весьма широком смысле. Первоначальное значение этого понятия, ограниченное только задачами автоматизации разработки ПО, в настоящее время приобрело новый смысл, охватывающий большинство процессов жизненного цикла ПО.

Появлению *CASE*-технологии и *CASE*-средств предшествовали исследования в области программирования: разработка и внедрение 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, этому способствовали следующие факторы:

- подготовка аналитиков и программистов, восприимчивых к концепциям модульного и структурного программирования;
- широкое внедрение и постоянный рост производительности компьютеров, позволившие использовать эффективные графические средства и автоматизировать большинство этапов проектирования;
- внедрение сетевой технологии, предоставившей возможность объединения усилий отдельных исполнителей в единый процесс проектирования путем использования разделяемой базы данных, содержащей необходимую информацию о проекте.

CASE-технология представляет собой совокупность методов проектирования ПО, а также набор инструментальных средств, позволяющих в наглядной форме моделировать предметную область, анализировать эту модель на всех стадиях разработки и сопровождения ПО и разрабатывать приложения в соответствии с информационными потребностями пользователей. Большинство существующих *CASE*-средств основано на методах структурного или объектно-ориентированного анализа и проектирования, использующих спецификации в виде диаграмм или текстов для описания внешних требований, связей между моделями системы, динамики поведения системы и архитектуры программных средств.

1. Визуальное моделирование

Под моделью ПО в общем случае понимается формализованное описание системы ПО на определенном уровне абстракции. Каждая модель определяет конкретный аспект системы, использует набор диаграмм и документов заданно-

го формата, а также отражает точку зрения и является объектом деятельности различных людей с конкретными интересами, ролями или задачами.

Под термином «моделирование» понимается процесс создания формализованного описания системы в виде совокупности моделей. Особенно трудным оказывается описание систем средней сложности, таких, как система коммутаций в телефонных сетях, управление авиаперевозками или движением подводной лодки, сборка автомобилей, челночные космические рейсы, функционирование перерабатывающих предприятий (к таким системам относится и ПО). С точки зрения человека, эти системы описать достаточно трудно, потому что они настолько велики, что практически невозможно перечислить все их компоненты со своими взаимосвязями, и в то же время недостаточно велики для применения общих упрощающих предположений (как это принято в физике). Неспособность дать простое описание, а, следовательно, и обеспечить понимание таких систем делает их проектирование и создание трудоемким и дорогостоящим процессом и повышает степень их ненадежности. С ростом технического прогресса адекватное описание систем становится все более актуальной проблемой.

Модель должна давать полное, точное и адекватное описание системы, имеющее конкретное назначение. Таким образом, целью модели является получение ответов на некоторую совокупность вопросов. Эти вопросы неявно присутствуют (подразумеваются) в процессе анализа и, следовательно, они руководят созданием модели и направляют его. Это означает, что сама модель должна будет дать ответы на эти вопросы с заданной степенью точности. Если модель отвечает не на все вопросы или ее ответы недостаточно точны, то говорят, что модель не достигла своей цели.

По мнению авторитетных специалистов в области проектирования ПО, моделирование является центральным звеном всей деятельности по созданию систем ПО. Модели строятся для того, чтобы понять и осмыслить структуру и поведение будущей системы, облегчить управление процессом ее создания и уменьшить возможный риск, а также документировать принимаемые проектные решения.

Визуальное моделирование - это способ восприятия проблем с помощью зримых абстракций, воспроизводящих понятия и объекты реального мира. Модели служат полезным инструментом анализа проблем, обмена информацией между всеми заинтересованными сторонами (пользователями, специалистами в предметной области, аналитиками, проектировщиками и т.д.), проектирования ПО, а также подготовки документации. Моделирование способствует более пол-

ному усвоению требований, улучшению качества системы и повышению степени ее управляемости.

Графические (визуальные) модели представляют собой средства для визуализации, описания, проектирования и документирования архитектуры системы. Под архитектурой понимается набор основных правил, определяющих организацию системы:

– совокупность структурных элементов системы и связей между ними;
– поведение элементов системы в процессе их взаимодействия;
– иерархию подсистем, объединяющих структурные элементы;
– архитектурный стиль (используемые методы и средства описания архитектуры, а также архитектурные образцы).

Архитектура является многомерной, поскольку различные специалисты работают с различными ее аспектами. Например, при строительстве здания используются разные типы чертежей, представляющие различные аспекты архитектуры:

– планы этажей;
– вертикальные проекции;
– планы монтажа кабельной проводки;
– чертежи водопроводов, системы центрального отопления и вентиляции;
– вид здания на фоне местности (эскизы).

Архитектура ПО также предусматривает различные представления, служащие разным целям:

– представлению функциональных возможностей системы;
– отображению логической организации системы;
– описанию физической структуры программных компонентов в среде реализации;
– отображению структуры потоков управления и аспектов параллельной работы;
– описанию физического размещения программных компонентов на базовой платформе.

Архитектурное представление - это упрощенное описание (абстракция) системы с конкретной точки зрения, охватывающее определенный круг интересов и опускающее объекты, несущественные с данной точки зрения. Архитектурные представления концентрируют внимание только на элементах, значимых с точки зрения архитектуры. Архитектурно значимый элемент - это элемент,

имеющий значительное влияние на структуру системы и ее производительность, надежность и возможность развития. Это элемент, важный для понимания системы. Например, в состав архитектурно значимых элементов объектно-ориентированной архитектуры входят основные классы предметной области, подсистемы и их интерфейсы, основные процессы или потоки управления.

Архитектурные представления подобны сечениям различных моделей, выделяющим только важные, значимые элементы моделей.

Разработка модели архитектуры системы ПО промышленного характера на стадии, предшествующей ее реализации или обновлению, в такой же мере необходима, как и наличие проекта для строительства большого здания. Это утверждение справедливо как в случае разработки новой системы, так и при адаптации типовых продуктов класса *R*/3 или *BAAN*, в составе которых также имеются собственные средства моделирования. Хорошие модели являются основой взаимодействия участников проекта и гарантируют корректность архитектуры. Поскольку сложность систем повышается, важно располагать хорошими методами моделирования. Хотя имеется много других факторов, от которых зависит успех проекта, но наличие строгого стандарта языка моделирования является весьма существенным.

Язык моделирования включает:

– элементы модели - фундаментальные концепции моделирования и их семантику;

– нотацию (систему обозначений) - визуальное представление элементов моделирования;

– руководство по использованию - правила применения элементов в рамках построения тех или иных типов моделей ПО.

Очевидно, что конечная цель разработки ПО - это не моделирование, а получение работающих приложений (кода). Диаграммы, в конечном счете - это всего лишь наглядные изображения. Создание слишком большого количества диаграмм до начала программирования займет слишком много времени и не обеспечит построения готовой системы. Поэтому при использовании графических языков моделирования очень важно понимать, чем это поможет, когда дело дойдет до написания кода. Можно привести следующие причины, побуждающие прибегать к их использованию:

– получение общего представления о системе. Графические модели помогают быстро получить общее представление о системе, сказать о том, какого рода абстракции существуют в системе и какие ее части нуждаются в дальнейшем уточнении;

– общение с экспертами организации. Графические модели образуют внешнее представление системы и объясняют, что эта система будет делать;

– изучение методов проектирования.

Множество людей отмечает наличие серьезных трудностей, связанных, например, с освоением объектно-ориентированных методов, и, в первую очередь, смену парадигмы. В некоторых случаях переход к объектно-ориентированным методам происходит относительно безболезненно. В других случаях при работе с объектами приходится сталкиваться с рядом препятствий, особенно в части максимального использования их потенциальных возможностей. Графические средства позволяют облегчить решение этой проблемы.

В процессе создания ПО, автоматизирующего деятельность некоторой организации, используются следующие виды моделей:

1) модели деятельности организации (или модели бизнес-процессов);

2) модели «*AS-IS*» («как есть»), отражающие существующее на момент обследования положение дел в организации и позволяющие понять, каким образом функционирует данная организация, а также выявить узкие места и сформулировать предложения по улучшению ситуации;

3) модели «*AS-TO-BE*» («как должно быть»), отражающие представление о новых процессах и технологиях работы организации.

Переход от модели «*AS-1S*» к модели «*AS-TO-BE*» может выполняться двумя способами:

– совершенствованием существующих технологий на основе оценки их эффективности;

– радикальным изменением технологий и перепроектированием (реинжинирингом) бизнес-процессов.

4) модели проектируемого ПО, которые строятся на основе модели «*AS-TO-BE*», уточняются и детализируются до необходимого уровня.

Состав моделей, используемых в каждом конкретном проекте, и степень их детальности в общем случае (как для структурного, так и для объектно-ориентированного подхода) зависят от следующих факторов:

– сложности проектируемой системы;

– необходимой полноты ее описания;

– знаний и навыков участников проекта;

– времени, отведенного на проектирование.

2. Структурные методы анализа и проектирования

Структурные методы являются строгой дисциплиной системного анализа и проектирования. Методы структурного анализа и проектирования стремятся пре-

одолеть сложность больших систем путем расчленения их на части («черные ящики») и иерархической организации этих «черных ящиков». Выгода в использовании «черных ящиков» заключается в том, что их пользователю не требуется знать, как они работают, необходимо знать лишь их входы и выходы, а также назначение (т.е. функции, которые они выполняет).

Таким образом, первым шагом упрощения сложной системы является ее разбиение на «черные ящики», при этом такое разбиение должно удовлетворять следующим критериям:

– каждый «черный ящик» должен реализовывать единственную функцию системы;

– функция каждого «черного ящика» должна быть легко понимаема независимо от сложности ее реализации;

– связь между «черными ящиками» должна вводиться только при наличии связи между соответствующими функциями системы (например, в бухгалтерии один «черный ящик» не обходим для расчета общей заработной платы служащего, а другой для расчета налогов - необходима связь между этими «черными ящиками»: размер заработной платы требуется для расчета налогов);

– связи между «черными ящиками» должны быть простыми, насколько это возможно, для обеспечения независимости между ними.

Второй важной идеей, лежащей в основе структурных методов, является идея иерархии. Для понимания сложной системы недостаточно разбиения ее на части, необходимо эти части организовать определенным образом, а именно в виде иерархических структур. Все сложные системы Вселенной организованы в иерархии: от галактик до элементарных частиц. Человек при создании сложных систем также подражает природе. Любая организация имеет директора, заместителей по направлениям, иерархию руководителей подразделений, рядовых служащих.

Кроме того, структурные методы широко используют визуальное моделирование, служащее для облегчения понимания сложных систем.

Структурным анализом принято называть метод исследования системы, начинающий с ее общего обзора, который затем детализируется, приобретая иерархическую структуру с все большим числом уровней. Для таких методов характерно:

– разбиение системы на уровни абстракции с ограничением числа элементов на каждом из уровней (обычно от 3 до 6-7);

– ограниченный контекст, включающий лишь существенные на каждом уровне детали;

– использование строгих формальных правил записи;
– последовательное приближение к конечному результату.

В структурном анализе основным методом разбиения на уровни абстракции является функциональная декомпозиция, заключающаяся в декомпозиции (разбиении) системы на функциональные подсистемы, которые, в свою очередь, делятся на подфункции, те - на задачи и так далее до конкретных процедур. При этом система сохраняет целостное представление, в котором все составляющие компоненты взаимоувязаны. При разработке системы «снизу вверх» от отдельных задач ко всей системе целостность теряется, возникают проблемы при описании информационного взаимодействия отдельных компонентов.

Все наиболее распространенные методы структурного подхода базируются на ряде общих принципов. Базовыми принципами являются:

– принцип «разделяй и властвуй» - принцип решения трудных проблем путем разбиения их на множество меньших независимых задач, легких для понимания и решения;
– принцип иерархического упорядочения - принцип организации составных частей системы в иерархические древовидные структуры с добавлением новых деталей на каждом уровне.

Выделение двух базовых принципов не означает, что остальные принципы являются второстепенными, поскольку игнорирование любого из них может привести к нежелательным последствиям (вплоть до неудачного завершения проекта). Основными из этих принципов являются:

– принцип абстрагирования - выделение существенных аспектов системы и отвлечение от несущественных;
– принцип непротиворечивости - обоснованность и согласованность элементов системы;
– принцип структурирования данных - данные должны быть структурированы и иерархически организованы.

В структурном анализе и проектировании используются различные модели, описывающие:

1) функциональную структуру системы;
2) последовательность выполняемых действий;
3) передачу информации между функциональными процессами;
4) отношения между данными.

Наиболее распространенными моделями первых трех групп являются:

– функциональная модель *SADT*;

– модель *IDEF*3;

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

Модель «сущность - связь» (*ERM – Entity-Relationship Model*), описывающая отношения между данными, традиционно используется в структурном анализе и проектировании, однако, по существу, представляет собой подмножество объектной модели предметной области.

6.4. Объектно-ориентированные методы анализа и проектирования программного обеспечения

Целью объектно-ориентированного анализа является трансформация функциональных требований к ПО в предварительный системный проект и создание стабильной основы архитектуры системы. В процессе проектирования системный проект «погружается» в среду реализации с учетом всех нефункциональных требований.

В основе объектно-ориентированного подхода (ООП) лежит объектная декомпозиция, при этом статическая структура системы описывается в терминах объектов и связей между ними, а поведение системы описывается в терминах обмена сообщениями между объектами. Каждый объект системы обладает своим собственным поведением, моделирующим поведение объекта реального мира [12].

Проблемы, стимулировавшие развитие ООП**:**

– необходимость повышения производительности разработки за счет многократного (повторного) использования ПО;

– необходимость упрощения сопровождения и модификации разработанных систем (локализация вносимых изменений);

– облегчение проектирования систем (за счет сокращения семантического разрыва между структурой решаемых задач и структурой ПО).

Объектная модель является наиболее естественным способом представления реального мира. В разделе «Теория классификации» Британской энциклопедии сказано следующее:

«В постижении реального мира люди пользуются тремя методами, организующими их мышление:

1. разделение окружающей действительности на конкретные объекты и их атрибуты (например, когда явно различаются понятия дерева и его высоты или пространственного расположения по отношению к другим объектам);

2. различие между целыми объектами и их составными частями (например, ветви являются составными частями дерева);

3. формирование и выделение различий между различными классами объектов (например, между классом всех деревьев и классом всех камней.)».

Понятие «объект» впервые было использовано около 30 лет назад в технических средствах при попытках отойти от традиционной архитектуры фон Неймана и преодолеть барьер между высоким уровнем программных абстракций и низким уровнем абстрагирования на уровне компьютеров. С объектно-ориентированной архитектурой также тесно связаны объектно-ориентированные операционные системы. Однако наиболее значительный вклад в объектный подход был внесен объектными и объектно-ориентированными языками программирования: *Simula* (1967), *Smalltalk* (1970-е гг.), *C++* (1980-е гг.) и языком моделирования *UML* (1990-е гг.). На объектный подход оказали влияние также развивавшиеся достаточно независимо методы моделирования данных, в особенности модель «сущность-связь».

1. Основные принципы построения объектных моделей.

Концептуальной основой объектно-ориентированного подхода является объектная модель. Основными принципами ее построения являются:

- абстрагирование (*abstraction*);
- инкапсуляция (*encapsulation*);
- модульность (*modularity*);
- иерархия (*hierarchy*).

Абстрагирование - это выделение наиболее важных, существенных характеристик некоторого объекта, которые отличают от всех других видов объектов и, таким образом, четко определяют его концептуальные границы с точки зрения дальнейшего рассмотрения и анализа, и игнорирование менее важных или незначительных деталей. Абстрагирование позволяет управлять сложностью системы, концентрируясь на существенных свойствах объекта. Абстрагирование концентрирует внимание на внешних особенностях объекта и позволяет отделить самые существенные особенности его поведения от деталей их реализации. Выбор правильного набора абстракций для заданной предметной области представляет собой главную задачу объектно-ориентированного проектирования. Абстракция зависит от предметной области и точки зрения - то, что важно в одном контексте,

может быть неважно в другом. Объекты и классы - основные абстракции предметной области.

Инкапсуляция - физическая локализация свойств и поведения в рамках единственной абстракции (рассматриваемой как «черный ящик»), скрывающая их реализацию за общедоступным интерфейсом. Инкапсуляция - это процесс отделения друг от друга отдельных элементов объекта, определяющих его устройство и поведение. Инкапсуляция служит для того, чтобы изолировать интерфейс объекта, отражающий его внешнее поведение, от внутренней реализации объекта. Объектный подход предполагает, что собственные ресурсы, которыми могут манипулировать только операции самого объекта, скрыты от внешней среды. Абстрагирование и инкапсуляция являются взаимодополняющими: абстрагирование фокусирует внимание на внешних особенностях объекта, а инкапсуляция (или иначе ограничение доступа) не позволяет объектам-пользователям различать внутреннее устройство объекта.

По-другому инкапсуляцию можно описать, сказав, что приложение разделяется на небольшие фрагменты связанной функциональности. Допустим, в банковской системе имеется информация, касающаяся банковского счета, такая как номер счета, баланс, имя и адрес его владельца, тип счета, начисляемые на него проценты и дата открытия. Со счетом также связаны определенные действия: открыть, закрыть его, положить или снять некоторую сумму денег, а также изменить тип, владельца или адрес. Вся эта информация и действия (поведение) совместно инкапсулируются в объект счет.

В результате все изменения банковской системы связанные со счетами, могут быть реализованы в одном только объекте «счет».

Еще одним преимуществом инкапсуляции является ограничение последствий изменений, вносимых в систему. Применим принцип инкапсуляции к банковской системе. Допустим, управление банка постановило, что если клиент имеет кредитный счет, то этот кредит может быть использован как овердрафт на его счете «до востребования». В неинкапсулированной системе модификация начинается с узконаправленного анализа изменений, которые необходимо будет внести в систему. Как правило, неизвестно, где в системе находятся все обращения к функции снятия со счета, поэтому приходится искать их везде. После того, как они найдены, нужно осуществить в них некоторые изменения, чтобы реализовать новые требования. Если работать тщательно, то, вероятно, можно будет обнаружить

около 80% случаев использования данной функции. В инкапсулированной системе не требуется осуществлять такой детальный анализ. Достаточно посмотреть на модель системы и определить, где инкапсулировано соответствующее поведение (действие снятия со счета). После его локализации остается внести требуемые поправки один раз только в этом объекте, и задача выполнена.

Инкапсуляция подобна понятию сокрытия информации (*information hiding*). Это возможность скрывать многочисленные детали объекта от внешнего мира. Внешний мир объекта - это все, что находится вне его, включая остальную часть системы. Сокрытие информации предоставляет то же преимущество, что и инкапсуляция, - гибкость.

Модульность - это свойство системы, связанное с возможностью ее декомпозиции на ряд внутренне сильно сцепленных, но слабо связанных между собой подсистем (модулей).

Модульность снижает сложность системы, позволяя выполнять независимую разработку отдельных модулей. Инкапсуляция и модульность создают барьеры между абстракциями.

Иерархия - это ранжированная или упорядоченная система абстракций, расположение их по уровням. Основными видами иерархических структур применительно к сложным системам являются структура классов (иерархия по номенклатуре) и структура объектов (иерархия по составу). Примерами иерархии классов являются простое и множественное наследование (один класс использует структурную или функциональную часть соответственно одного или нескольких других классов), а иерархии объектов - агрегация.

2. Основные элементы объектной модели

К основным понятиям объектно-ориентированного подхода (элементам объектной модели) относятся:

- -объект;
- класс;
- атрибут;
- операция;
- полиморфизм (интерфейс);
- компонент;
- связи.

Объект определяется как осязаемая сущность (*tangible entity*) - предмет или явление (процесс), имеющие четко определяемое поведение.

Объект может представлять собой абстракцию некоторой сущности предметной области (объект реального мира) или программной системы (архитектурный объект). Любой объект обладает состоянием (*state*), поведением (*behavior*) и индивидуальностью (*identity*).

Состояние объекта - одно из возможных условий, в которых он может существовать, оно изменяется со временем. Состояние объекта характеризуется перечнем всех возможных (статических) свойств данного объекта и текущими значениями (динамическими) каждого из этих свойств. Состояние объекта определяется значениями его свойств (атрибутов) и связями с другими объектами.

Поведение объекта определяет действия объекта и его реакцию на запросы от других объектов. Поведение характеризует воздействием объекта на другие объекты, изменяющее их состояние. Иначе говоря, поведение объекта полностью определяется его действиями. Поведение представляется с помощью набора сообщений, воспринимаемых объектом (операций, которые может выполнять объект).

Индивидуальность объект. Каждый объект обладает уникальной индивидуальностью Индивидуальность - это свойства объекта, отличающие его от других объектов. Структура и поведение схожих объектов определяют для них общий класс. Термины «экземпляр класса» и «объект» являются эквивалентными.

Класс - это множество объектов, связанных общностью свойств, поведения, связей и семантики. Класс инкапсулирует (объединяет) в себе данные (атрибуты) и поведение (операции). Класс является абстрактным определением объекта и служит в качестве шаблона для создания объектов. Класс изображается в виде прямоугольника, разделенного на три части (рис.6.6). В первой содержится имя класса, во второй - его атрибуты. В последней части содержатся операции класса, отражающие его поведение (действия, выполняемые классом). Любой объект является экземпляром (*instance*) класса. Определение классов и объектов - одна из самых сложных задач объектно-ориентированного проектирования.

Атрибут - поименованное свойство класса, определяющее диапазон допустимых значений, которые могут принимать экземпляры данного свойства. Атрибут - это элемент информации, связанный с классом.

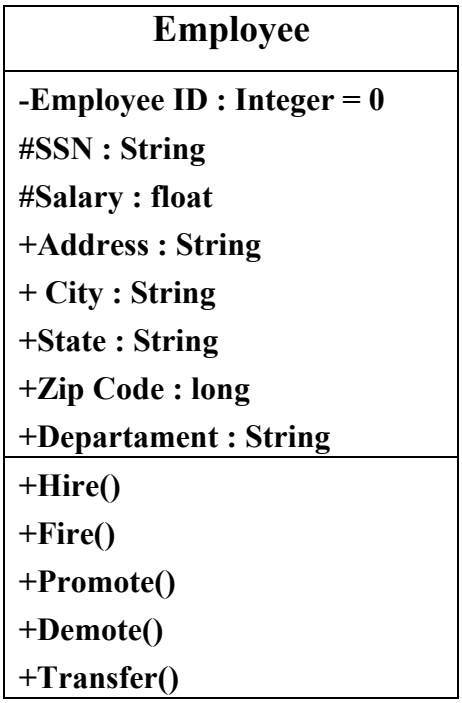


Рис. 6.6. Графическое представление класса

Например, у класса *Company* (Компания) могут быть атрибуты *Name* (Название), *Address* (Адрес) и *NumberOfEmployees* (Число служащих).

Так как атрибуты содержатся внутри класса, они скрыты от других классов. В связи с этим может понадобиться указать, какие классы имеют право читать и изменять атрибуты. Это свойство называется видимостью атрибута (*attribute visibility*).

У атрибута можно определить три возможных значения этого параметра. Рассмотрим каждый из них в контексте примера (рис.6.6). Пусть имеется класс *Employee c* атрибутом *Address* и класс *Company*.

Видимость атрибута *Public* (общий, открытый). Это значение видимости предполагает, что атрибут будет виден всеми остальными классами. Любой класс может просмотреть или изменить значение атрибута. В таком случае класс *Company* может изменить значение атрибута *Address* класса *Employee*. В соответствии с нотацией *UML* общему атрибуту предшествует знак «+».

Private (закрытый, секретный). Соответствующий атрибут не виден никаким другим классом. Класс *Employee* будет знать значение атрибута *Address* и сможет изменять его, но класс *Company* не сможет его, ни увидеть, ни редактировать. Если это понадобится, он должен попросить класс *Employee* просмотреть или изменить значение этого атрибута, что обычно

делается с помощью операции. Закрытый атрибут обозначается знаком «-» в соответствии с нотацией *UML*.

Protected (защищенный). Такой атрибут доступен только самому и его потомкам в иерархии наследования. Допустим, имеется два различных типа сотрудников - с почасовой оплатой и на окладе. Таким образом, потомками класса *Employee* будут два класса - *HourlyEmp и SalariedEmp*. Защищенный атрибут *Address* можно просмотреть или изменить из классов *Employee, HourlyEmp и SalariedEmp*, но не из класса *Company.* Нотация *UML* для защищенного атрибута - это знак «#».

В общем случае атрибуты рекомендуется делать закрытыми или защищенными. При этом удается избежать ситуации, когда значение атрибута изменяется всеми классами системы. Вместо этого логика изменения атрибута будет заключена в том же классе, что и сам этот атрибут.

Определенное воздействие одного объекта на другой с целью вызвать соответствующую реакцию называется операцией. Операция - это реализация услуги, которую можно запросить у любого объекта данного класса.

Операции отражают поведение объекта. Операция-запрос не изменяет состояния объекта. Операция-команда может изменить состояние объекта. Результат операции зависит от текущего состояния объекта.

Как правило, в объектных и объектно-ориентированных языках программирования операции, выполняемые над данным объектом, называются методами и являются составной частью определения класса.

Операции реализуют связанное с классом поведение (иначе говоря, реализуют обязанности класса - *responsibilities*). В широком смысле обязанности класса делятся на две категории.

Знание (определяется атрибутами класса):

– наличие информации о данных или вычисляемых величинах;
– наличие информации о связанных объектах.

Действие (определяется операциями класса):

– выполнение некоторых действий самим объектом;
– инициация действий других объектов;
– координация действий других объектов.

Операция включает три части - имя, параметры и тип возвращаемого значения. Параметры - это аргументы, получаемые операцией «на входе». Тип возвращаемого значения относится к результату действия операции.

В языке *UML* операции имеют следующую нотацию: Имя Операции (аргументе1: тип данных аргумента1, аргумент2: тип данных аргумента2,...): тип возвращаемого значения.

Существуют четыре различных типа операций.

– Операции реализации (*implementor operations*) реализуют некоторые функции (процедуры). Такие операции выявляются путем анализа диаграмм взаимодействия *UML*.

– Операции управления (*manager operations*) управляют созданием и уничтожением объектов. В эту категорию попадают конструкторы и деструкторы классов.

– Атрибуты обычно бывают закрытыми или защищенными. Тем не менее, другие классы иногда должны просматривать или изменять их значения. Для этого существуют операции доступа (*access operations*).

– Вспомогательными (*helper operations*) называются такие операции класса, которые необходимы ему для выполнения его обязанностей, но о которых другие классы не должны ничего знать. Это закрытые и защищенные операции класса.

Понятие полиморфизма может быть интерпретировано, как способность класса принадлежать более чем одному типу.

Полиморфизм - это способность скрывать множество различных реализаций под единственным общим интерфейсом.

Интерфейс - совокупность операций, определяющих набор услуг класса или компонента. Интерфейс не определяет внутреннюю структуру, все его операции имеют открытую видимость.

Полиморфизм тесно связан с наследованием. Наследование означает построение новых классов на основе существующих с возможностью добавления или переопределения свойств (атрибутов) и поведения (операций).

Объектно-ориентированная система изначально строится с учетом ее эволюции. Наследование и полиморфизм обеспечивают возможность определения новой функциональности классов с помощью создания производных классов - потомков базовых классов. Потомки наследуют характеристики родительских классов, изменения их первоначального описания и добавляют при необходимости собственные структуры данных и методы. Определение производных классов, при котором задаются только различия или уточнения, в огромной степени экономит время и усилия при производстве и использовании спецификаций и программного кода.

Компонент – относительно независимая и замещаемая часть системы, выполняющая четко определенную функцию в контексте заданной архитектуры. Компонент представляет собой физическую реализацию проектной абстракции и может быть:

- компонентом исходного кода;
- компонентом времени выполнения (*runtime*);
- исполняемым компонентом.

Компонент обеспечивает физическую реализацию набора интерфейсов.

Между элементами объектной модели существуют различные виды связей. К основным типам связей относятся связи ассоциации, зависимости и обобщения.

Ассоциация (*association*) - это семантическая связь между классами. Ее изображают на диаграмме классов в виде обыкновенной линии (рис.6.7). Ассоциация отражает структурные связи между объектами различных классов.

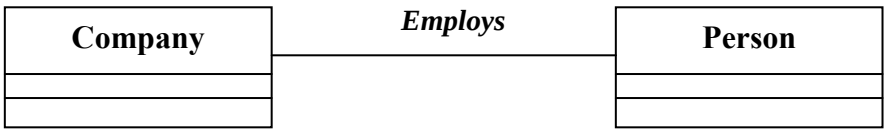


Рис. 6.7. Ассоциация

Агрегация (*aggregation*) представляет собой форму ассоциации - более сильный тип связи между целым (составным) объектом и его частями (компонентными объектами).

Существуют четыре возможных семантики для агрегации:

1) агрегация типа «Безраздельно обладает»;
2) агрегация типа «Обладает»;
3) агрегация типа «Включает»;
4) агрегация типа «Участник».

Агрегация типа «Безраздельно обладает» устанавливает следующее:

- между компонентными объектами и их составными объектами установлено отношение зависимости по существованию (следовательно, удаление составного объекта распространяется вниз по иерархии отношения, так что связанные компонентные объекты также удаляются);
- агрегация транзитивна (если объект C1 является частью объекта B1, а B1 - частью A1, тогда C1 является частью A1);

– агрегация асимметрична (нерефлексивная) (если объект B1 является частью объекта A1, то объект A1 не является частью B1);

– агрегация стационарна, если объект B1 является частью объекта A1, то он не может быть частью объекта C1.

Агрегация типа «Обладает» поддерживает первые три свойства агрегации «Безраздельно обладает», к которым относятся:

– зависимость по существованию;

– транзитивность;

– асимметричность.

Агрегация типа «Включает» слабее, чем агрегация типа «Обладает», она поддерживает транзитивность и асимметричность.

Агрегация типа «Участник» обладает свойством целенаправленного группирования независимых объектов - группирования, при котором не делается предположений относительно свойства зависимости по существованию, транзитивности, асимметричности или стационарности. Компонентный объект в агрегации типа «Участник» может одновременно принадлежать более чем одному составному объекту.

Язык *UML* обеспечивает ограниченную поддержку агрегации. Сильная форма агрегации является в *UML* композицией. В композиции составной объект может физически содержать компонентные объекты. Компонентный объект может принадлежать только одному составному объекту. Композиция языка *UML* в большей или меньшей степени соответствует агрегациям типа «Безраздельно обладает» и «Обладает».

Слабая форма агрегации в *UML* называется просто агрегацией. При этом составной объект физически не содержит компонентный объект. Один компонентный объект может обладать несколькими связями ассоциации или агрегации. Иначе говоря, агрегация в языке *UML* соответствует агрегациям типа «Включает» и «Участник».

Агрегация изображается линией между классами с ромбом на стороне целого объекта (рис.6.8). Сплошной ромб представляет композицию (рис. 6.9).

Рис. 6.8. Агрегация

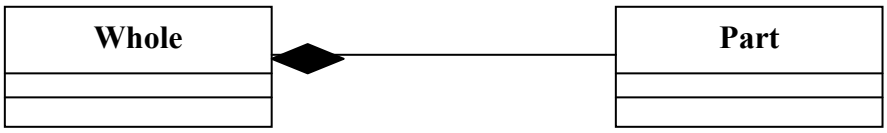


Рис. 6.9. Композиция

Хотя связи ассоциации и агрегации двунаправленные по умолчанию, часто накладываются ограничения на направление навигации (только в одном направлении). Если введено ограничение по направлению, то добавляется стрелка на конце связи Направление ассоциации можно определить, изучая сообщения между классами. Если все сообщения на них отправляются только одним классом и принимаются только другим классом, но не наоборот, между этими классами имеет место однонаправленная связь. Если хотя бы одно сообщение отправляется в обратную сторону, ассоциация должна быть двунаправленной.

Ассоциации могут быть рефлексивными. Рефлексивная ассоциация предполагает, что один экземпляр класса взаимодействует с другими экземплярами этого же класса.

Связи можно уточнить с помощью имен связей или ролевых имен. Имя связи - это обычно глагол или глагольная фраза, описывающая, зачем она нужна. Например, между классом *Person* (человек) и классом *Company* (компания) может существовать ассоциация. Если человек является сотрудником компании, ассоциацию можно назвать «*employs*» (нанимает) (рис. 6.10).

Имена у связей определять не обязательно. Обычно это делают, если причина создания связи не очевидна. Имя показывают около линии соответствующей связи.

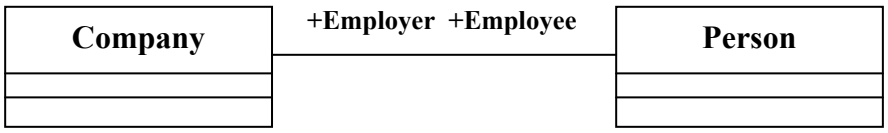


Рис. 6.10. Ролевые имена

Для уточнения роли, которую играет каждый класс в связях ассоциации или агрегации, применяют ролевые имена. Возвращаясь к примеру с классами *Person* и *Company*, можно сказать, что класс *Person* играет роль сотрудника класса *Company*. Ролевые имена - это обычно имена существительные или основанные на них фразы, их показывают на диаграмме рядом с классом, играющим соответствующую роль. Как правило, пользуются или ролевым именем, или именем связи, но не обоими сразу. Как и имена

связей, ролевые имена не обязательны, их дают, только если смысл связи не очевиден.

Мощность (*multiplicity*) показывает, как много объектов участвует в связи. Мощность - это число объектов одного класса, связанных с одним объектом другого класса. Понятие мощности связи в объектной модели аналогично понятиям мощности и класса принадлежности связи в модели «сущность-связь» (с точностью до расположения показателя мощности на диаграмме). Для каждой связи можно обозначить два показателя мощности - по одному на каждом конце связи.

В языке *UML* приняты следующие нотации для обозначения мощности (табл.6.1.).

Таблица 6.1. Значения мощности

Мощность	Значения
*	Много
0	Нуль
1	Один
0..*	Нуль или больше
1..*	Один или больше
1..1	Ровно один

Например, при разработке системы регистрации курсов в университете можно определить классы *Course* (учебный курс) и *Student* (студент). Между ними установлена связь, означающая посещение курсов студентами. Если один студент может посещать от нуля до четырех курсов, а на одном курсе могут заниматься от 10 до 20 студентов, то на диаграмме классов это можно изобразить следующим образом (рис. 6.11).

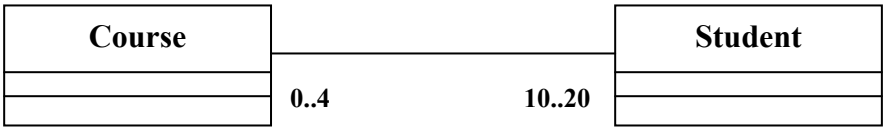


Рис 6.11. Мощность связи

Частным случаем ассоциации является ассоциация-класс (*Association class*), которая обладает как свойствами класса, так и свойствами ассоциации. Ассоциация-класс - это место, где хранятся относящиеся к ассоциации атрибуты и операции. Экземплярами ассоциации-класса являются свя-

зи, у которых есть не только ссылки на объекты, но и значения атрибутов. Ассоциация класс подобна связи с атрибутами в модели «сущность связь».

Допустим, что имеются два класса, *Student* и *Course* и возникла необходимость добавить атрибут *Grade* (оценка) В таком случае возникает вопрос, в какой класс надо его добавить. Если поместить его внутри класса *Student*, то придется вводить его для каждого посещаемого студентом курса, что слишком сильно увеличит размер этого класса. Если же поместить его внутри класса *Course*, то придется задавать его для каждого посещающего этот курс студента.

Чтобы решить эту проблему, можно создать ассоциацию-класс. В этот класс следует поместить атрибут *Grade*, относящийся к связи между курсом и студентом. Нотация *UML* для ассоциации-класса представлена на рис. 6.12.

Ассоциация-класс определяет дополнительное ограничение, согласно которому двум участвующим в ассоциации объектам может соответствовать только один экземпляр ассоциации-класса. Диаграмма на рис. 6.13 не допускает, чтобы студент мог получать по курсу более чем одну оценку. Если необходимо, чтобы такое допускалось, то ассоциацию-класс *Grade* следует преобразовать в обычный класс, связанный ассоциациями с классами *Student* и *Course*.

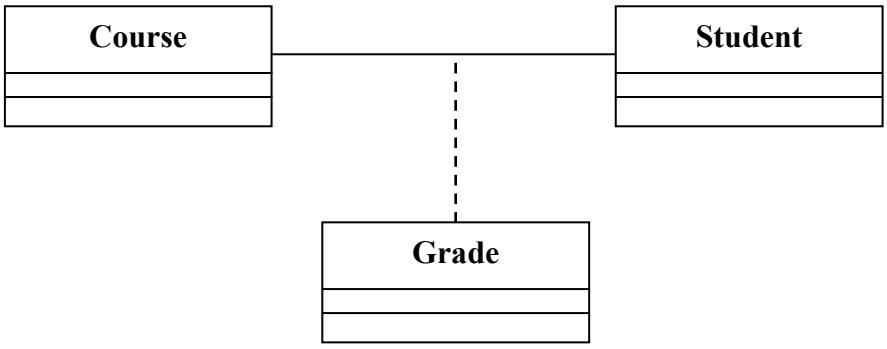


Рис. 6.12.Нотация UML для ассоциации-класса

Зависимость (*dependency*) - связь между двумя элементами модели, при которой изменения в спецификации одного элементе могут повлечь за собой изменения в другом элементе. Зависимость - слабая форма связи между клиентом и сервером (клиент зависит от сервера и не имеет знаний о сервере). Зависимость изображается пунктирной линией, направленной от клиента к серверу (6.13).

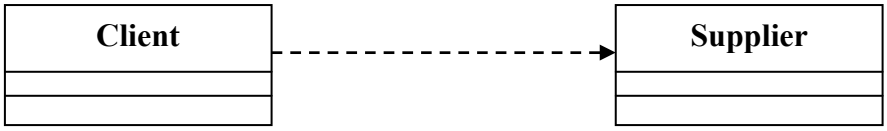


Рис 6.13. Зависимость

Зависимость между двумя элементами имеет место в том случае, если изменения в определении одного элемента могут повлечь за собой изменения в другом. Причины для зависимостей могут быть самыми разными: один класс посылает сообщение другому; один класс включает часть данных другого класса; один класс использует другой в качестве параметра операции. Если класс меняет свой интерфейс, то любое сообщение, которое он посылает, может утратить свою силу.

Обобщение (*generalization*) - связь «тип-подтип» реализует механизм наследования (*inheritance*). Большинство объектно-ориентированных языков непосредственно поддерживают концепцию наследования. Она позволяет одному классу наследовать все атрибуты, операции и связи другого. В языке *UML* связи наследования называют обобщениями и изображают в виде стрелок от класса-потомка к классу-предку (рис. 6.14).

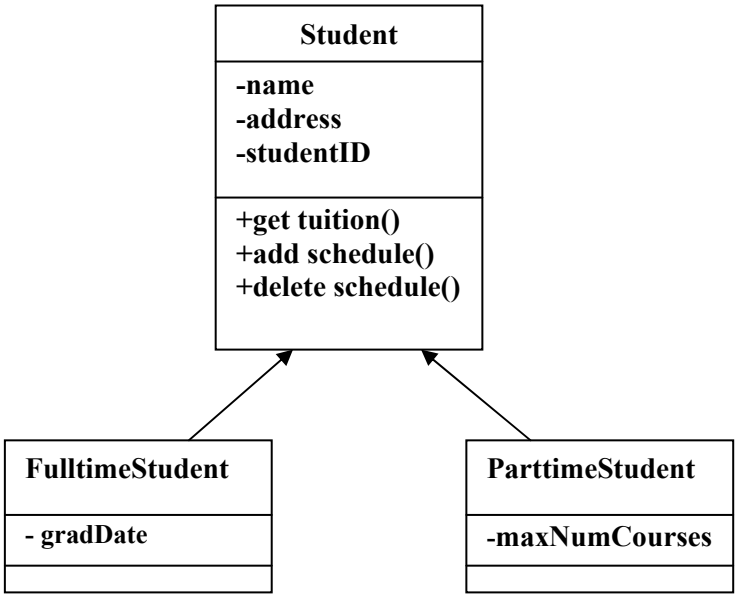


Рис. 6.14. Обобщение

Общие атрибуты, операции и/или связи отображаются на верхнем уровне иерархии.

Помимо наследуемых, каждый подкласс имеет свои собственные уникальные атрибуты, операции и связи.

6.5. Унифицированный язык моделирования *UML*

Унифицированный язык моделирования *UML* (*Unified Modeling Language*) является преемником методов объектно-ориентированного анализа и проектирования (*OOA&D*), которые появились в конце 80-х и начале 90-х годов. Он непосредственно унифицирует методы Буча, Рамбо (*OMT*) и Джекобсона, однако обладает большими возможностями. Язык *UML* прошел процесс стандартизации в рамках консорциума *OMG* (*Object Management Group*) и в настоящее время является стандартом *OMG*.

UML - это название языка моделирования, но не метода. Большинство методов включают в себя, по крайней мере, в принципе, язык моделирования и процесс. Язык моделирования - это нотация (главным образом, графическая), которую используют методы для описания проектов. Процесс - это рекомендация относительно этапов, которые необходимо выполнить при разработке проекта.

Система объектно-ориентированных моделей в соответствии с нотациями *UML* включает в себя следующие диаграммы:

1) диаграмму прецедентов использования (*Use-case diagram*), которая отображает функциональность ЭИС в виде совокупности выполняющихся последовательностей транзакций;

2) диаграмму классов объектов (*Class diagram*), которая отображает структуру совокупности взаимосвязанных классов объектов аналогично *ER*-диаграмме функционально-ориентированного подхода;

3) диаграммы состояний (*Statechart diagram*), каждая из которых отображает динамику состояний объектов одного класса и связанных с ними событий;

4) диаграммы взаимодействия объектов (*Interaction diagram*), каждая из которых отображает динамическое взаимодействие объектов в рамках одного прецедента использования;

5) диаграммы деятельностей (*Activity diagram*), которые отображают потоки работ во взаимосвязанных вариантах использования (могут декомпозироваться на более детальные диаграммы);

6) диаграммы пакетов (*Package diagram*), которые отображают распределение объектов по функциональным или обеспечивающим подсистемам (могут декомпозироваться на более детальные диаграммы);

7) диаграмму компонентов (*Component diagram*), которая отображает физические модули программного кода;

8) диаграмму размещения/развертывания (*Deployment diagram*),

которая отображает распределение объектов по узлам вычислительной сети.

1. Диаграммы прецедентов использования

Диаграмма прецедентов использования выявляет основные бизнес-процессы как последовательности транзакций, которые должны выполняться целиком, когда выполнение обособленного подмножества действий не имеет значения без выполнения всей последовательности. Прецеденты использования инициируются из внешней среды пользователями ЭИС, называемыми актерами. На этом уровне моделирования не раскрывается механизм реализации процессов. Представленные сущности имеют следующие графические обозначения:

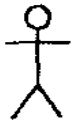


Рис.6.15. Актер - внешний пользователь процесса

Рис.6.16. Прецедент использования (бизнес-процесс)

Актер инициирует выполнение прецедента использования и получает от него результаты. Взаимодействие (ассоциация) актера с прецедентом использования осуществляется в результате события, которое обозначается поименованной стрелкой (рис. 6.17). Один актер может участвовать в нескольких прецедентах использования, а в одном прецеденте использования может быть занято несколько актеров.

В реализации прецедента использования возможно выделение нескольких потоков событий:

– основной поток, событий, который приводит к требуемому результату наиболее коротким путем, например выполнение заказа без задержек;

– альтернативные потоки событий, например временное откладывание или полный отказ от выполнения заказов. Основной и альтернативный потоки событий в модели прецедентов использования описываются в виде неформальных текстовых комментариев.

Рис. 6.17. Диаграмма вариантов использования

Несколько прецедентов использования может иметь общую часть, выделяемую в самостоятельный прецедент использования, с которым устанавливаются отношения использования (*uses*). С другой стороны, некоторые прецеденты использования могут быть расширены деталями. В таком случае создается дополнительный прецедент использования, с которым устанавливаются отношения расширения (*extends*). Пример применения такого рода отношений показан на рисунке 6.18.

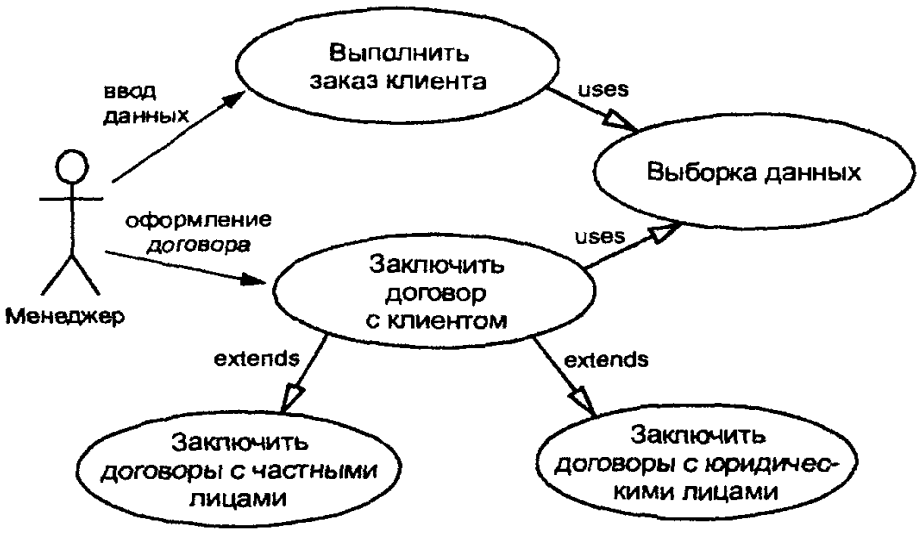


Рис. 6.18. Пример применения отношений использования и расширения.

2. Диаграммы взаимодействия

Для каждого прецедента использования может быть построена модель динамического взаимодействия объектов, которая представляется в одной из двух форм:

– в форме диаграммы последовательностей (*sequence diagram*), показывающей последовательность взаимодействий на графе;

– в форме кооперативной диаграммы (*collaboration diagram*), показывающей взаимодействие объектов в табличной форме.

В диаграмме последовательностей взаимодействие объектов отображается в виде стрелки между объектами, которая соответствует событию или сообщению от одного объекта к другому, вызывающему выполнение метода, реагирующего на событие (сообщение) объекта. Номер стрелки соответствует номеру события в последовательности. Пример диаграммы последовательностей представлен на рисунке 6.19.

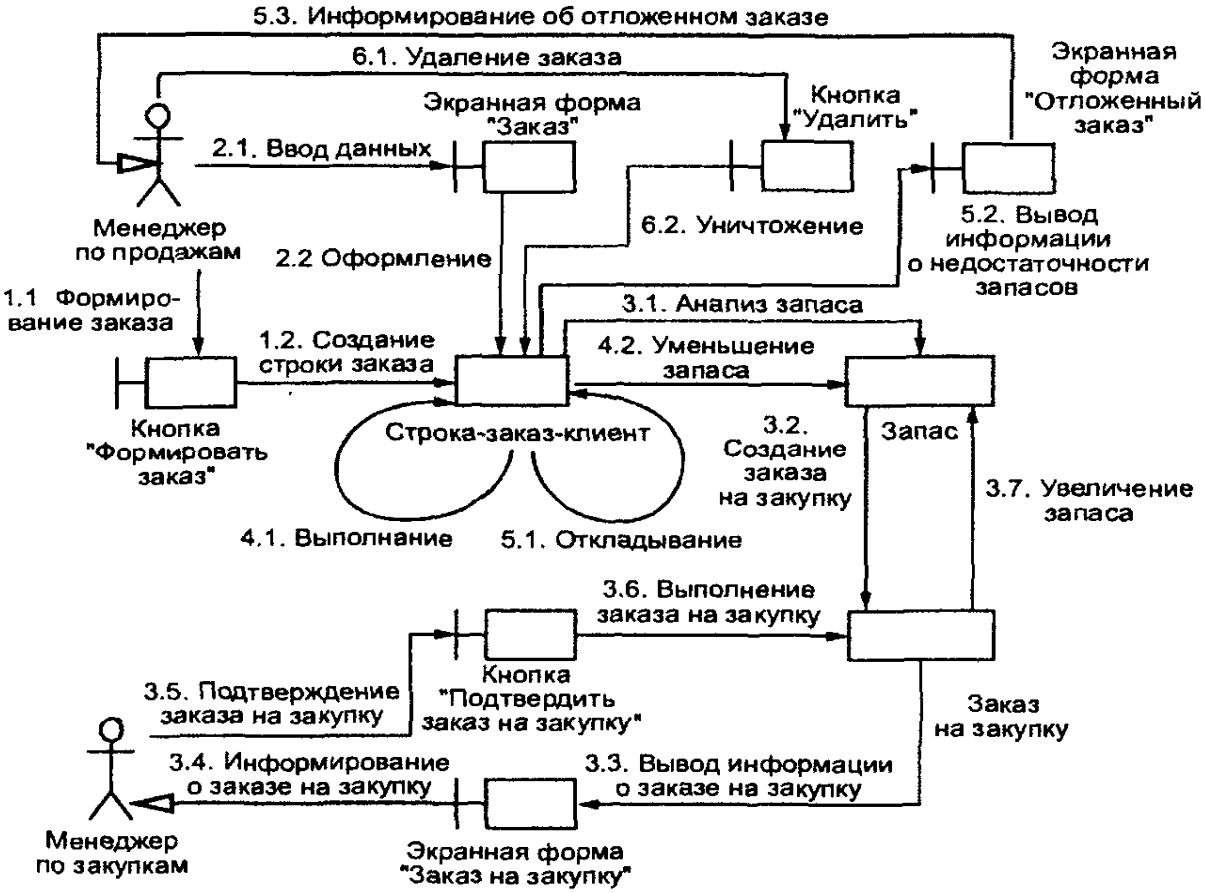


Рис. 6.19. Диаграмма последовательностей для прецедента «Выполнение заказа клиента»

3. Диаграммы классов

Диаграммы классов объектов (*Class diagram*) отображают статическую структуру классов объектов. Эта диаграмма рассматривает внутреннюю структуру проблемной области, иерархию классов объектов, статические связи объектов.

Классы объектов могут иметь различные стереотипы поведения: объекты-сущности, управляющие объекты, интерфейсные объекты (рис. 6.20).

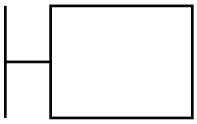


Рис. 6.20. Интерфейсный объект

Интерфейсный объект (*Interface Object*) - активный объект, форма взаимодействия информационной системы с пользователем (экранная форма, меню, командная строка, кнопка).

Управляющий объект (*Control Object*) - активный объект, координирующий выполнение функций (рис. 6.21).

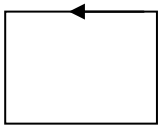


Рис. 6.21. Управляющий объект

Сущность (*Entity Object*) - пассивный объект, над которым выполняются операции обработки процесса (рис. 6.22).

Рис.6.22 . Сущность

Объекты, отражаемые в диаграмме классов объектов, связываются статическими отношениями, которые отражают постоянные связи между объектами независимо от выполнения конкретного бизнес-процесса. К статическим отношениям относятся обобщение, агрегация, ассоциация объектов:

0..1 *****

Рис. 6.23. Отношения ассоциации

Отношения ассоциации 0..1:1; 0..1:M, M:N могут быть поименованы; 0..1 - необязательность связи; * - множественность.

Рис. 6.24. Отношения обобщения (наследования)

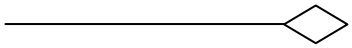


Рис. 6.25. Отношения агрегации (целое - часть)

Пример использования статических отношений представлен на рисунке 6.30.

В прямоугольниках в верхней части даны имена классов объектов, в средней части - имена атрибутов, в нижней части - имена методов.

4. Диаграммы состояний

Диаграмма состояний отображает поведение объектов одного класса в динамике, связь состояний объектов с событиями и определяет:

- какие типичные состояния проходит объект;
- какие события ведут к изменению состояния объекта;
- какие действия объект выполняет, когда он получает сообщение об изменении состояния;
- как объекты создаются и уничтожаются (входные и выходные точки диаграммы).

Ниже представлены используемые в диаграмме состояний понятия и их графическое обозначение:

Рис. 6.26. Входная точка

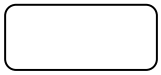


Рис. 6.27.Состояние

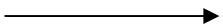


Рис. 6.28. Переход состояний

Рис. 6.29 Выходная точка

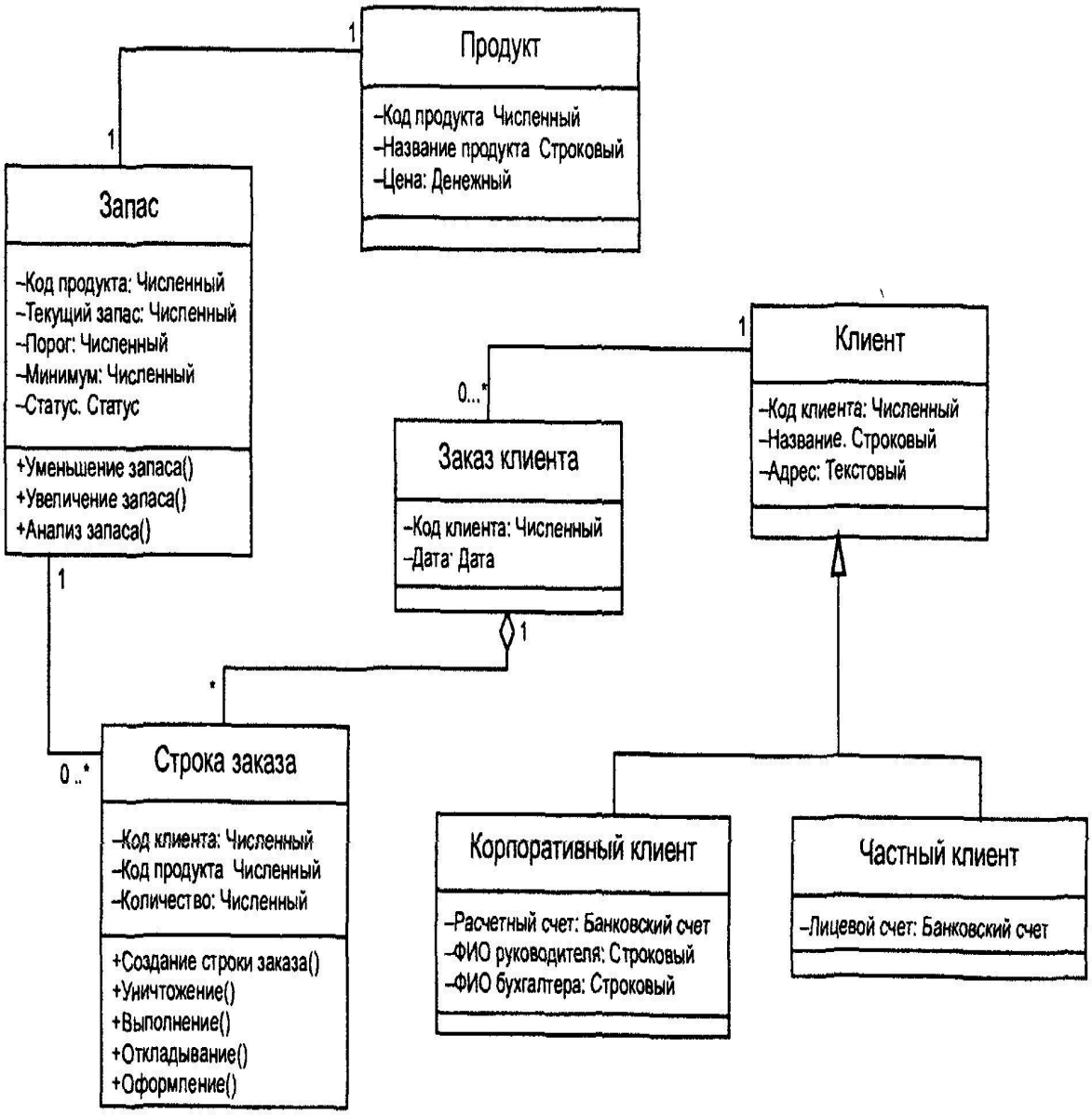


Рис. 6.30. Фрагмент диаграммы классов объектов

Входная точка определяет событие, которое образует начальное состояние объекта. В точку входа нельзя перейти из состояния объекта.

Выходная точка определяет завершение существования объекта. Из точки выхода нет перехода состояния.

Состояние представляет ситуацию, в течение которой выполняется непрерывная деятельность или объект находится в стационарном положении. Состояние определяется как набор значений атрибутов и отношений,

связанных с объектом. Имя состояния должно быть уникальным только внутри класса объекта, для которого оно определяется.

С каждым состоянием связано одно событие или более, которые могут его изменить. Для состояния задаются имена всех связанных с ним переходов в другие состояния.

Переход состояний определяет изменение в состоянии объекта, которое происходит в результате события, возникшего в то время, когда объект находился в данном состоянии. Каждый переход состояний должен иметь уникальное имя.

Переход состояний описывается следующими атрибутами.

Назначение - состояние объекта, в которое перейдет объект после перехода состояния.

Вызов - имя события, которое вызывает переход состояний. Имена событий должны быть идентичными в определении класса и состояния. Вызываемые события могут быть либо внешними, осуществляемыми актерами, либо внутренними, связанными с поведением других объектов, либо временными, связанными с истечением заданного интервала времени.

Условие перехода - это логическое выражение, связанное с атрибутами объекта, которое должно быть проверено для выбора перехода состояния. Условие перехода задается в том случае, если происходит событие, в результате которого может произойти неоднозначный переход состояний. Условия переходов для одного исходного состояния должны быть взаимоисключающими.

Действие - атрибут, информационно описывающий сущность действия, которое должно выполняться при переходе состояний. Этому действию будет соответствовать некоторая процедура, реализующая метод класса объектов.

Переход состояний графически помечается меткой линии, на которой задается, по крайней мере, один из следующих атрибутов: Вызов, Условие перехода, Действие.

Пример модели перехода состояний представлен на рисунке 6.31.

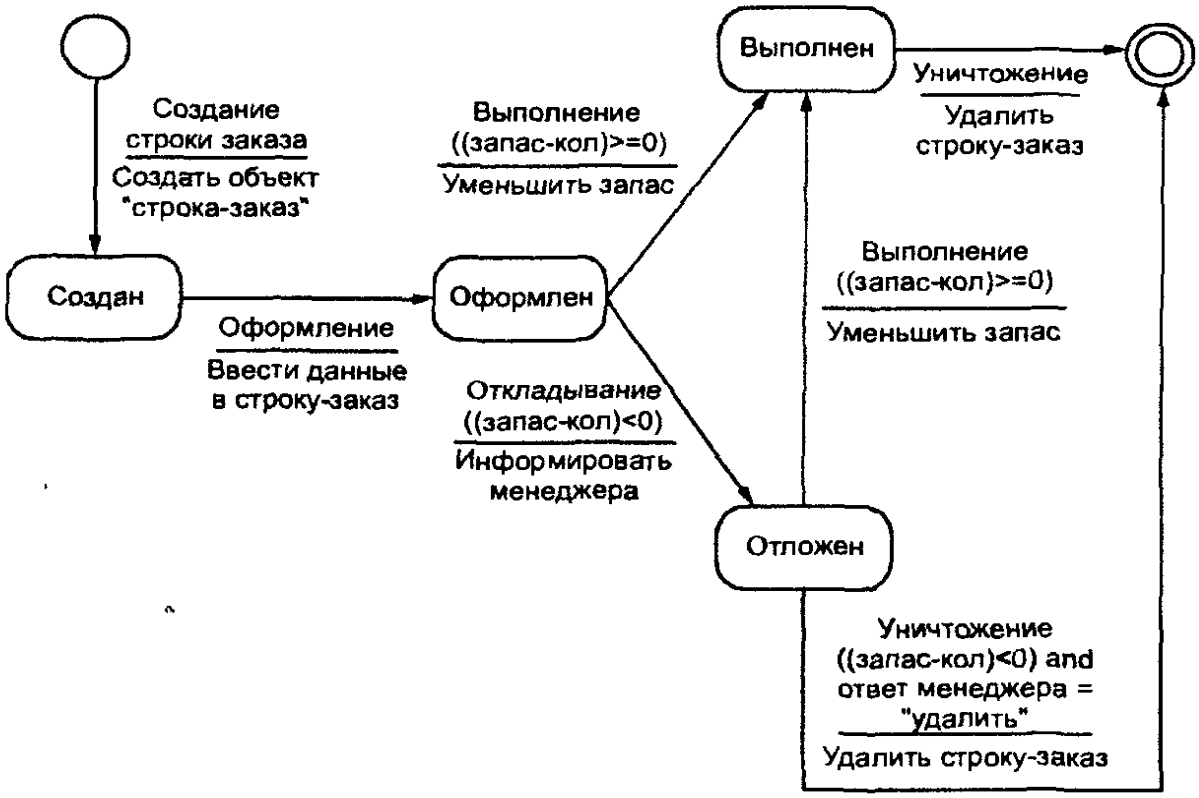


Рис.6.31. Пример диаграммы состояний для объекта

5. Диаграммы деятельности

Диаграммы взаимодействий не отражают детально порядок выполнения операций в части разветвлений, циклических повторений, параллельности/произвольности действий. Диаграмма деятельностей исправляет данные недостатки. Под деятельностью будем понимать некоторую работу, которая может быть декомпозирована на совокупность действий.

Диаграмма деятельностей может отражать взаимодействие объектов из нескольких прецедентов использования, в частности реализующих отдельно стандартные и альтернативные пути обработки объектов.

Блок, соответствующий одной деятельности, может отражать несколько событий и быть декомпозирован аналогично блоку функционально-ориентированного подхода.

Ниже представлены используемые в диаграмме деятельностей понятия и их графическое обозначение.

Рис. 6.32. Деятельность (activity)

Рис.6.33.Поток от деятельности к деятельности

Рис. 6.34. Разделение потока на деятельности, выполняемые параллельно или произвольно

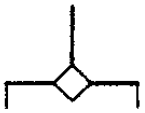


Рис.6.35 . Решение

Рис.6.36. Синхронизация

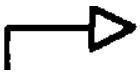


Рис. 6.37..Итерация

Рис.6.38. Выход

Пример диаграммы деятельностей представлен на рисунке 6.39.

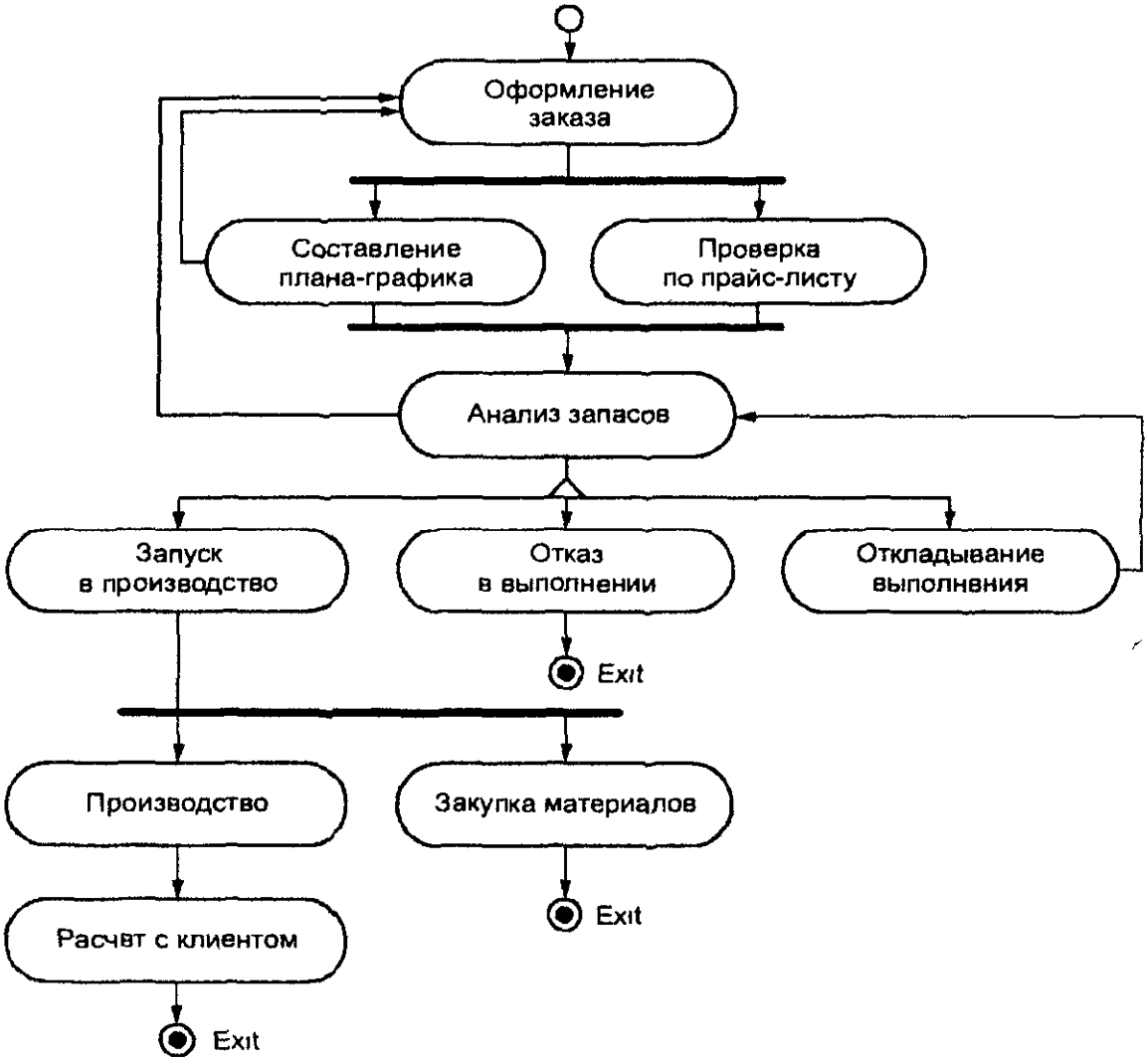


Рис.6.39. Диаграмма деятельностей процесса выполнения заказа

6. Диаграммы компонентов и диаграммы размещения

Диаграмма компонентов отображает зависимости программных компонентов, которые представляются в виде исходных, откомпилированных и исполняемых программных кодов объектов. Один компонент, как правило, соответствует программному коду одного пакета классов объектов.

Компонент в своем составе имеет интерфейсный класс объектов, через который осуществляется доступ к остальным классам объектов компонента. На рисунке 6.40 интерфейс обозначен маленьким кружком, присоединенным к пиктограмме компонента. С помощью интерфейса объекты других компонентов обращаются не к конкретным объектам рассматриваемого компонента, а к его интерфейсному объекту. Таким образом, уп-

рощается взаимодействие компонентов между собой, когда при доступе к компоненту из других компонентов не требуется знать внутреннюю структуру этого компонента. Компонент, к которому осуществляется обращение, может быть необъектно-ориентированным. Достаточно, чтобы у такого компонента был только один интерфейсный класс объектов, который транслирует запросы к компоненту в вызовы обычных процедур. У компонентов может быть несколько интерфейсов.

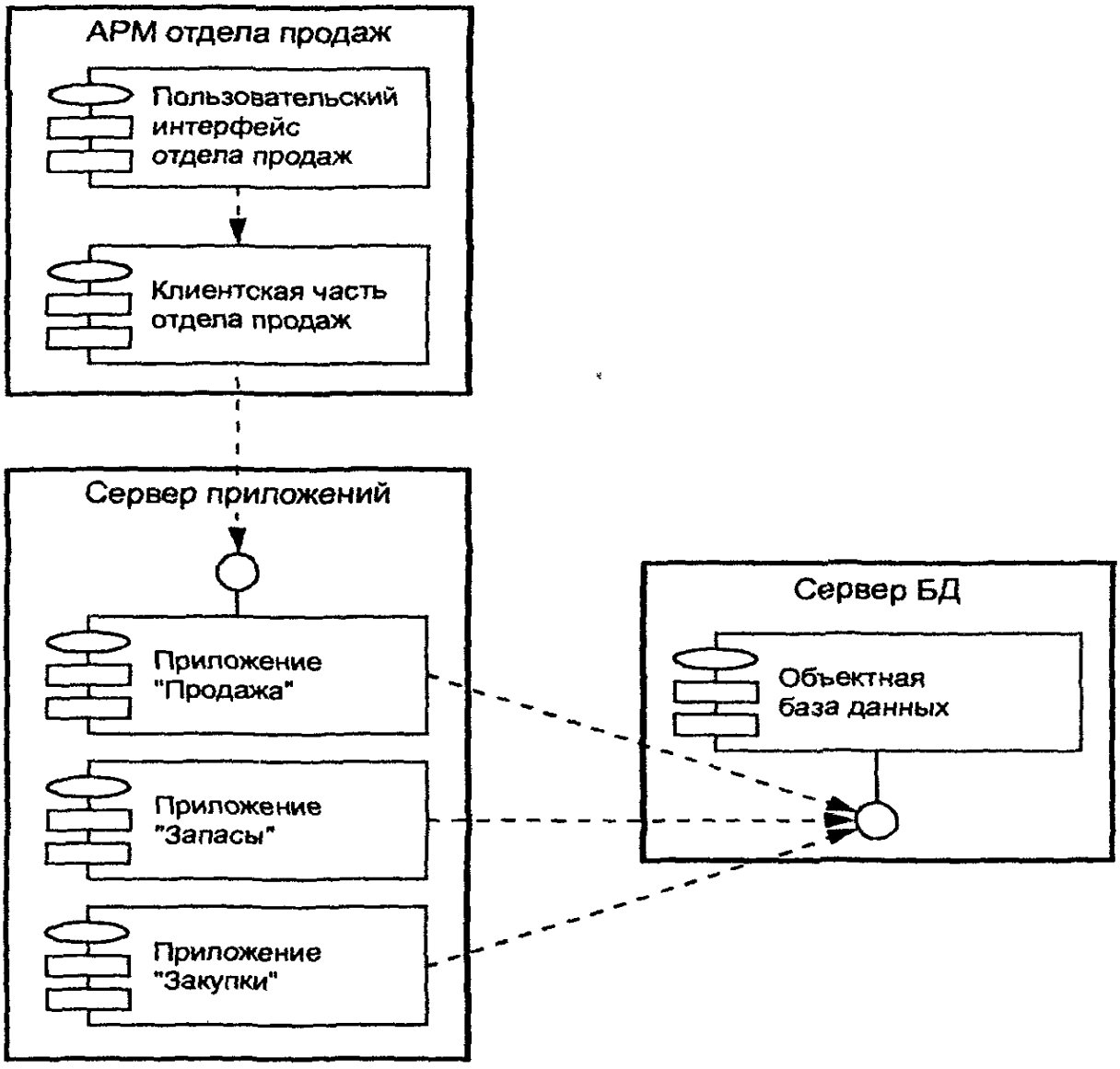


Рис. 6.40. Пример диаграммы компонентов и размещения

В модели размещения отображается топология расположения компонентов по узлам вычислительной сети. Отдельный компонент всегда

располагается на одном компьютере-сервере. На одном компьютере-сервере может располагаться несколько компонентов (рис. 6.40).

Рассмотрим технологическую сеть проектирования ИС на основе использования объектно-ориентированной *CASE*-технологии, для которой характерны последовательное расширение и уточнение моделей на различных стадиях жизненного цикла ИС: анализа системных требований, логического и физического проектирования, реализации.

Технологическая сеть объектно-ориентированного проектирования ИС представляет собой обобщение методологий *Objectory и Natural Engineering Workbench*

7. Диаграммы пакетов

В объектно-ориентированном подходе пакет содержит множество взаимосвязанных классов объектов и соответствует понятию «подсистема функционально-ориентированного подхода». Один прецедент использования может требовать классы объектов из разных пакетов. Класс объектов обычно назначается одному пакету, но с позиции достижения разных подцелей может входить в состав разных пакетов.

Пакетная технология группирования классов объектов позволяет упростить:

– разработку и эксплуатацию ЭИС;

– гибкую адаптацию типовых компонентов с позиции их повторного использования;

– оптимизацию клиент-серверной архитектуры ИС.

Обычно ИС разбивается на функциональные и обеспечивающие пакеты. Функциональные пакеты, соответствующие решаемым проблемам (задачам), объединяются в один общий пакет «Проблемная область». Каждый пакет, в свою очередь, может быть разбит на подпакеты в соответствии с семантической близостью и теснотой взаимодействия классов объектов. Обычно пакеты проблемной области содержат иерархии обобщения и агрегации. Классы объектов, требуемые в нескольких подсистемах, выделяются в самостоятельные пакеты. В одном пакете, как правило, определяется не более 20 компонентов, обычно 5-15.

С обеспечивающей точки зрения ИС разбивают на пять основных пакетов:

1) «Интерфейс», объекты которого реализуют функции взаимодействия пользователей с ИС по вводу-выводу информации и обмен сообщениями между подсистемами;

2) «База данных», объекты которого выполняют доступ к данным во внешней памяти;

3) «Управление задачами», объекты которого осуществляют функции диспетчеризации и маршрутизации обработки объектов, например, в системе управления рабочими потоками;

4) «Утилиты», объекты которого осуществляют вспомогательные функции, например преобразование форматов данных;

5) обеспечивающие пакеты, т.е. работающие по принципу «клиент-серверной» архитектуры, выполняющие серверные функции для функциональных объектов-клиентов. Таким образом, обеспечивающие пакеты освобождают пользователя от знания деталей программно-технической реализации ИС.

6.6. Механизмы расширения *UML*

Механизм расширения специфицирует порядок включения в модель элементов с уточненной семантикой, а также модификацию отдельных компонентов языка *UML* для более точного отражения специфики моделируемых систем [12]. Механизм расширения определяет семантику для стереотипов, ограничений и помеченных значений. Хотя язык *UML* обладает богатым множеством понятий и нотаций для моделирования типичных программных систем, реально разработчик может столкнуться с необходимостью включить в модель дополнительные свойства или нотации, которые не определены явно в языке *UML*. При этом разработчики часто сталкиваются с необходимостью включения в модель графической информации, такой, например, как дополнительные значки и украшения.

Для этой цели в языке *UML* предусмотрены три механизма расширения, которые могут использоваться совместно или раздельно для определения новых элементов модели с отличающимися семантикой, нотацией и свойствами от специфицированных в метамодели языка *UML* элементов. Такими механизмами являются: ограничение (*Constraint*), стереотип (*Stereotype*) и помеченное значение (*TaggedValue*).

Таким образом, механизмы расширения языка *UML* предназначены для выполнения следующих задач:

1) Уточнения существующих модельных элементов при разработке моделей на языке *UML*.

2) В спецификации самого языка *UML* для определения

стандартных компонентов, которые либо не являются достаточно интересными, либо сложны для непосредственного определения в качестве элементов метамодели *UML*.

3) Определения таких расширений языка *UML*, которые зависят от специфики моделируемого процесса или от языка реализации программного кода.

4) Присоединения произвольной семантической или несемантической информации к элементам модели.

Наиболее важные из встроенных механизмов расширения основываются на понятии стереотип. Стереотипы обеспечивают некоторый способ классификации модельных элементов на уровне объектной модели и возможность добавления в язык *UML* "виртуальных" метаклассов с новыми атрибутами и семантикой. Другие встроенные механизмы расширения основываются на понятии списка свойств, содержащего помеченные значения и ограничения. Эти механизмы обеспечивают пользователю возможность включения дополнительных свойств и семантики непосредственно в отдельные элементы модели.

1. Профили языка (наборы стереотипов)

Профили *UML*, позволяют разработчикам моделей представлять различные виды элементов *web*-приложения – клиентские и серверные страницы, формы, кадры и т.д. Профиль содержит набор стереотипов для различных элементов и их отношений. Этот подход описан в книге *Building Web Applications with UML* (*Addison Wesley Longman*, 2000). Профили *UML*, входят в состав последних версий *Rational Rose.*

В мире существуют группы, создающие профили на основе *UML* для различных прикладных областей.

UML обладает механизмами расширения, предназначенными для того, чтобы разработчики могли адаптировать язык моделирования к своим конкретным нуждам, не меняя при этом его метамодель. Наличие механизмов расширения принципиально отличает *UML* от таких средств моделирования, как *IDEF*0, *IDEF*1*X*, *IDEF*3, *DFD* и *ERM*. Перечисленные языки моделирования можно определить как сильно типизированные (по аналогии с языками программирования), поскольку они не допускают произвольной интерпретации семантики элементов моделей. *UML*, допуская такую интерпретацию (в основном за счет стереотипов), является слабо типизированным языком. К его механизмам расширения относятся:

– стереотипы;

– тегированные (именованные) значения;
– ограничения.

Стереотип - это новый тип элемента модели, который определяется на основе уже существующего элемента. Стереотипы расширяют нотацию модели и могут применяться к любым элементам модели. Стереотипы классов - это механизм, позволяющий разделять классы на категории. Разработчики ПО могут создавать свои собственные наборы стереотипов, формируя тем самым специализированные подмножества *UML* (например, для описания бизнес-процессов, *Web*-приложений, баз данных и т.д.). Такие подмножества (наборы стереотипов) в стандарте языка *UML* носят название профилей языка.

Именованное значение - это пара строк «тег = значение», или «имя = содержимое», в которых хранится дополнительная информация о каком-либо элементе системы, например, время создания, статус разработки или тестирования, время окончания работы над ним и т.п.

Ограничение - это семантическое ограничение, имеющее вид текстового выражения на естественном или формальном языке (*OCL - Object Constraint Language)*, которое невозможно выразить с помощью нотации *UML*.

Профили *UML* позволяют вам настраивать язык на соответствующую предметную область или метод. Профили *UML* предлагают набор стереотипов, которые расширяют существующие элементы *UML*, чтобы их можно было использовать в данном контексте. Эта техника применяется в *MDD*, чтобы дизайнеры могли вести моделирование с использованием концепций предметной области.

Сторонники *UML* 2.0 утверждают, что этот язык отражает передовой опыт и лучшие методы моделирования. На первый план выдвигаются следующие его достоинства:

– расширение представления об *UML* как о семействе языков с помощью профилей и точек семантических вариаций; они помечают преднамеренно оставленные без семантики части *UML*, чтобы облегчить добавление семантики, определенной пользователем;
– большая выразительность моделирования, в том числе улучшенное моделирование бизнес-процессов, поддержка моделирования классификаторов многократного использования и архитектур распределенных гетерогенных систем;
– интеграция семантики действий (*Action Semantics*), которую

разработчики могут использовать для определения семантики среды выполнения моделей и обеспечения семантической точности, необходимой для анализа моделей и их преобразования в конкретные реализации.

Точки семантических вариаций обеспечивают малый уровень адаптации, а профили - высокий уровень. Профиль *UML* описывает расширение элементов модели *UML* для использования в конкретном контексте моделирования. Например, с помощью профиля можно определить вариант *UML*, адаптированный для моделирования прикладных компонентов *Enterprise JavaBeans*.

В профиле элементы модели *UML* расширяются с помощью стереотипов, определяющих дополнительные свойства элемента. Свойства, которые определены расширением, введенным стереотипом, не должны противоречить свойствам, ассоциированным с элементом модели. С помощью профиля можно ввести новые ограничения и определить дополнительные атрибуты в классах, но нельзя ввести новые или удалить существующие классы элементов модели. По этой причине профили рассматриваются как облегченные механизмы расширения. В качестве примеров профилей можно назвать *UML Profile for Schedulability*, *OMG Performance and Time* и профиль *SysML,* предназначенный для моделирования систем

2. Ограничение и язык описания ограничений OCL

Язык ограничения объектов (*OCL, Object Constraint Language*) является описательным текстовым языком для создания ограничений. Он используется в спецификации *UML*, выражая правила для корректно построенных моделей.

Язык *Object Constraint Language* 2.0 (*OCL*) является языком ограничений, дополняющим язык *UML* 2.1. Этот язык может быть использован как для верификации *UML* моделей, так и для спецификации запросов к *UML* модели.

Язык *OCL* текстовый язык, разработанный как дополнение к графическим языкам. Хотя графическая нотация языка *UML* позволяет точно и компактно представить многие аспекты *UML*-модели, часто средств графической нотации бывает недостаточно. Дополнения к графической нотации на естественном языке могут иметь неоднозначное толкование и не могут быть использованы для автоматической верификации моделей. Текстовый язык *OCL* позволяет накладывать дополнительные ограничения на модель, используя для этого классы, отношения ассоциации между клас-

сами, атрибуты и операции классов, которые уже представлены в *UML* модели и показаны на *UML* диаграммах.

3. Языка OCL 2.0 как средство спецификации метамодели языка UML 2.0

В спецификации метамодели языка *UML* 2.0 выражения на языке *OCL* используются для нескольких целей. Первое назначение языка *OCL* - описание операций запроса к классам метамодели. Данная операция не может модифицировать *UML*-модель, а только лишь возвращает запрашиваемое значение. Таким образом, язык *OCL* специфицирует важную функциональность классов метамодели, которая затем должна быть представлена на языке реализации метамодели. В нашем случае таким языком реализации является язык *C#*.

На рисунке 6.41. показан небольшой фрагмент диаграммы *UML* из спецификации метамодели.

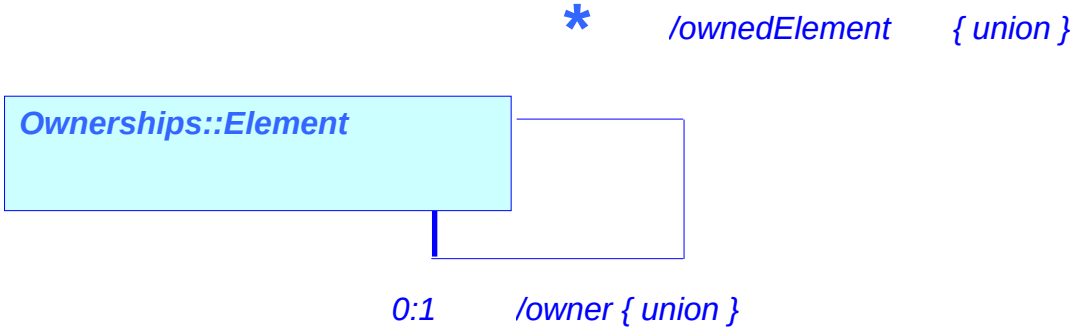


Рис.6.41. Описание отношения собственности классом Element.

На рисунке представлен абстрактный класс *Element* из пространства имен *Ownership*. Класс *Element* вступает в рекурсивное отношение ассоциации с самим собой, что показано ребром графа с прямой линией. Этот класс является предком классов метамодели, вследствие чего показанное на диаграмме отношение ассоциации применимо и к классам-потомкам класса *Element*. Из имен ролей и обозначений множественности на окончаниях ассоциации следует, что у элемента модели может быть неограниченное (*) количество элементов модели, которым он владеет (*ownedElement*). С другой стороны, из вида другого окончания ассоциации следует, что у элемента модели может быть не более одного (0:1) элемента-собственника (*owner*).

Рассмотрим дополнение к диаграмме, показанной на рисунке. Для запроса всех элементов метамодели, которыми прямо или косвенно владеет данный элемент модели, в стандарте на языке *OCL* специфицирована следующая операция *allOwnedElements:*

– Все элементы, которыми прямо или косвенно владеет данный элемент.

```
Element::allOwnedElements(): Set(Element);
allOwnedElements = ownedElement->union(
 ownedElement->collect(e|
 e.allOwnedElements())
```

Второе назначение *OCL* в спецификации метамодели - с помощью логических выражений описывать ограничения, налагаемые на элементы модели, которые должны выполняться все время существования модели. Таким образом, семантика языка описывается более формально и точно. Выражения на *OCL* возможно использовать также и для верификации *UML*-моделей построенных из классов метамодели.

Третий способ использования *OCL* в спецификации метамодели - описание алгоритма вычисления порожденных (*derived*) атрибутов и окончаний ассоциации. Для этого может быть использован синтаксис аналогичный описанию операций на языке *OCL*.

4. Реализация языка OCL 2.0

Особенностью языка *OCL* является то, что выражения на этом языке имеют смысл в контексте *UML* модели. Для трансляции выражений необходимо, что бы уже существовала *UML*-модель, которая содержит имена атрибутов классов и имена ролей окончаний отношений ассоциации используемых при описании выражений на *OCL*.

Наиболее полная реализация метамодели языка *UML* 2.0 в открытых исходных кодах была сделана в рамках проекта *Eclipse на языке Java.* Для данной реализации языка *UML* была адаптирована реализация языка *OCL*, сделанная в университете *Canterbury*. Дальнейшее развитие эта реализация получила в подпроекте *The Eclipse Modeling Framework Technology* проекта *Eclipse.*

5. Ограничения и язык OCL

Как уже отмечалось, в диаграммах классов могут указываться ограничения целостности, которые должны поддерживаться в проектируемой БД. Имеются два способа определения ограничений: на естественном язы-

ке и на языке *OCL*. На рисунке 6.42 показана простая диаграмма классов Студент и Университет с ограничением, выраженном на естественном языке.

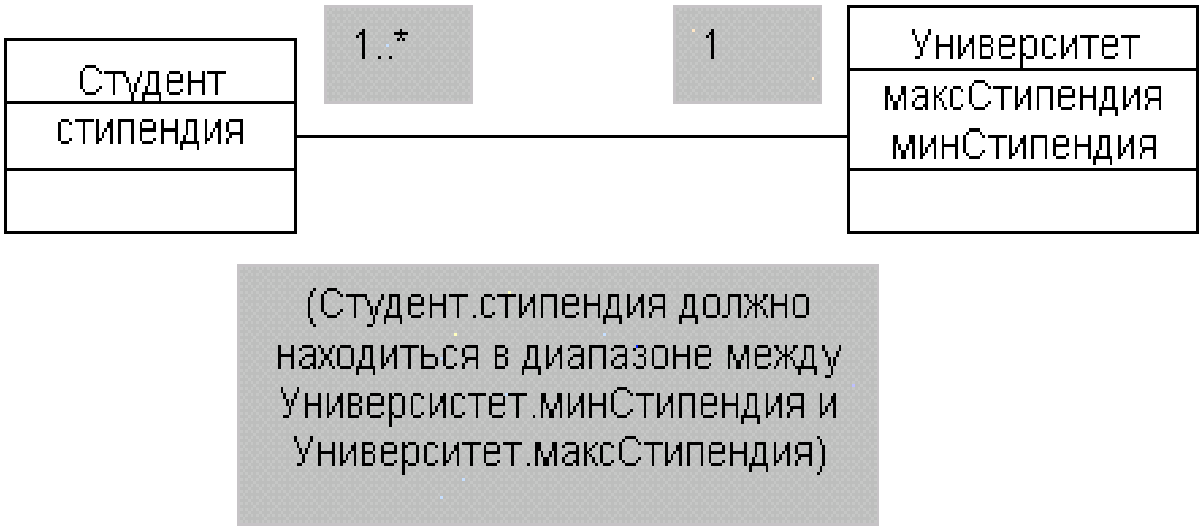


Рис. 6.42. Диаграмма классов Студент-Университет

В данном случае накладывается ограничение на состояние объектов классов Студент и Университет, входящих в один экземпляр ассоциации. Объект класса Студент может входить в экземпляр связи с объектом класса Университет только при том условии, что размер стипендии данного студента находится в диапазоне, допустимом в данном университете.

Более точный и лаконичный способ формулировки ограничений обеспечивает язык *OCL* (*Object Constraint Language*). Приведем его сжатое описание.

К заимствованным из *UML* концепциям относятся, в первую очередь, следующие:

- Класс, операция, атрибут
- Объект (экземпляр класса)
- Ассоциация
- Тип данных (включая набор предопределенных типов *Boolean, Integer, Real и String*)
- Значение (экземпляр типа данных).

Существенными для понимания языка *OCL* являются определенные в *UML* отличия между объектом некоторого класса и значением некоторого типа, обычные для объектных моделей данных:

1) Объект имеет уникальный идентификатор и может сравниваться с другими объектам только по значению идентификатора, следствием чего является возможность определения множественных операций над объектами в терминах их идентификаторов.

2) Объект может быть ассоциирован через бинарную связь с другими объектами, что позволяет определить в *OCL* операцию перехода от объекта к связанным с ним объектам.

3) В то же время значение является “чистым значением” в том смысле, что при сравнении двух значений проверяются сами эти значения, и, кроме того, значения никак не могут участвовать в связи, поскольку это понятие определено только для объектов классов.

В дополнение к скалярным типам данных, заимствованным из *UML*, в *OCL* предопределены структурные типы, которые являются разновидностями коллекций (*collection*):

1) Математическое множество (*set*), т.е. неупорядоченная коллекция, не содержащая одинаковых элементов.

2) Мультимножество (*bag*), т.е. коллекция, которая может содержать повторяющиеся элементы.

3) Последовательность (*sequence*), т.е. упорядоченная коллекция, которая может содержать повторяющиеся элементы.

В *OCL* элементами каждого из трех типов коллекций могут быть либо объекты, либо значения.

Основной задачей, которую призван решить язык *OCL*, является определение ограничений на данные, соответствующие модели, которая представлена в терминах диаграммы классов. *OCL* может применяться для определения ограничений, описывающих пред- и постусловия операций классов, и ограничений, представляющих собой инварианты классов. При проектировании реляционных баз данных возможность определения пред- и постусловий операций вряд ли может оказаться существенной. С точки зрения определения ограничений целостности баз данных более важны средства определения инвариантов классов.

Под инвариантом класса в *OCL* понимается условие, которому должны удовлетворять все объекты данного класса. Более точно, инвариант класса - это логическое выражение, вычисление которого должно да-

вать *true* при создании каждого объекта данного класса и сохранять это значение в течение всего времени существования объекта. При определении инварианта требуется указать имя класса и выражение, определяющее инвариант указанного класса.

Отметим, что *OCL* является типизированным языком, поэтому у каждого выражения имеется некоторый тип. Естественно, что *OCL*-выражение в инварианте класса должно быть логического типа.

В общем случае *OCL*-выражение в определении инварианта основывается на композиции операций, которым посвящена большая часть определения языка. В спецификации языка эти операции условно разделены на следующие группы: операции над значениями предопределенных в *UML* (скалярных) типов данных, операции над объектами, операции над множествами, операции над мультимножествами, операции над последовательностями.

В заключение обзора языка *OCL* приведем пример инварианта, выраженного на этом языке. Будем основываться на диаграмме классов, показанной на рисунке 6.43.

Пример:
context Сотрудник *inv*:
self.возраст >18 *and self*.возраст < 100

Этот инвариант указывает, что возраст сотрудников должен быть больше 18 и меньше 100, что выражается в виде ограничений на значения атрибута возраст, определенного в классе Сотрудник.

6. Применение образцов при проектировании ПО

Шаблоны проектирования (паттерн, *pattern*) - это эффективные способы решения характерных задач проектирования, в частности проектирования компьютерных программ. Паттерн не является законченным образцом проекта, который может быть прямо преобразован в код, скорее это описание или образец для того, как решить задачу, таким образом, чтобы это можно было использовать в различных ситуациях. Объектно-ориентированные шаблоны зачастую показывают отношения и взаимодействия между классами или объектами, без определения того, какие конечные классы или объекты приложения будут использоваться. Алгоритмы не рассматриваются как шаблоны, так как они решают задачи вычисления, а не проектирования.

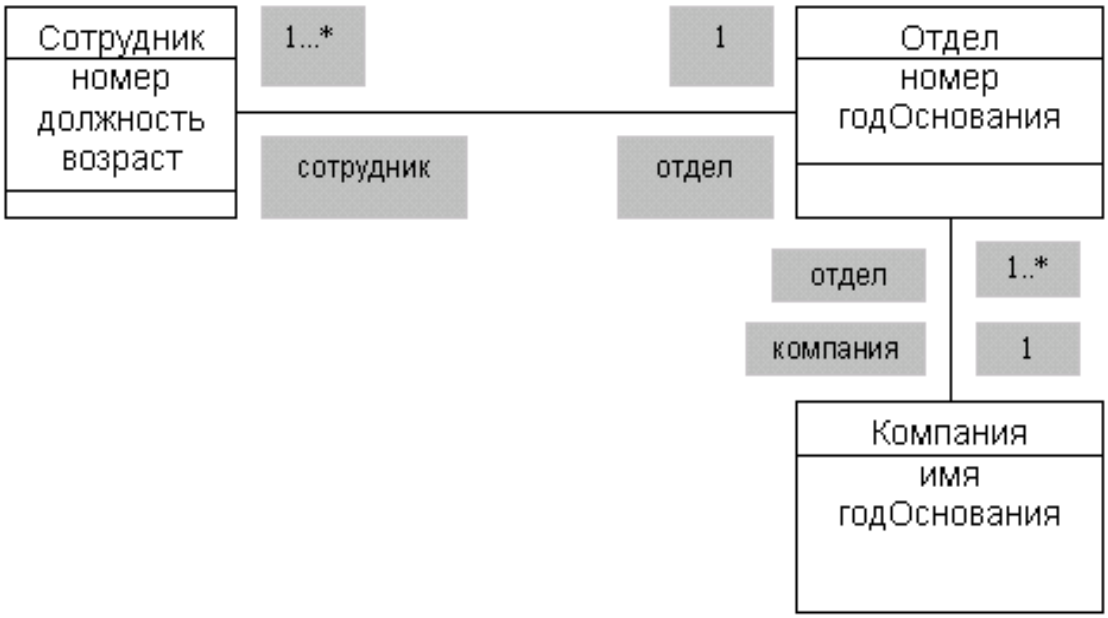


Рис.6.43. Диаграмма классов, используемая для примеров на языке OCL

Главная польза каждого отдельного шаблона состоит в том, что он описывает решение целого класса абстрактных проблем. Также тот факт, что каждый шаблон имеет свое имя, облегчает дискуссию об абстрактных структурах данных (*ADT*) между разработчиками, так как они могут ссылаться на известные шаблоны. Таким образом, за счёт шаблонов производится унификация терминологии, названий модулей и элементов проекта.

Правильно сформулированный паттерн проектирования позволяет, отыскав удачное решение, пользоваться им снова и снова.

В отличие от идиом, шаблоны независимы от применяемого языка программирования.

Иногда шаблоны консервируют громоздкую и малоэффективную систему понятий, разработанную узкой группой. Когда количество шаблонов возрастает, превышая критическую сложность, исполнители начинают игнорировать шаблоны и всю систему, с ними связанную.

Нередко шаблонами заменяется отсутствие или недостаточность документации в сложной программной среде.

Есть мнение, что слепое применение шаблонов из справочника, без осмысления причин и предпосылок выделения каждого отдельного шаблона, замедляет профессиональный рост программиста, так как подменяет творческую работу механическим подставлением шаблонов. Люди, придерживающиеся данного мнения, считают, что знакомиться со списками

шаблонов надо тогда, когда "дорос" до них в профессиональном плане - и не раньше. Хороший критерий нужной степени профессионализма - выделение шаблонов самостоятельно, на основании собственного опыта. При этом разумеется, знакомство с теорией, связанной с шаблонами, полезно на любом уровне профессионализма и направляет развитие программиста в правильную сторону. Сомнению подвергается только использование шаблонов "по справочнику".

7. Основные типы шаблонов проектирования

Основные шаблоны (*Fundamental*):

- *Delegation pattern*/шаблон делегирования;
- *Functional design/*Шаблон функционального дизайна;
- *Interface*;
- *Proxy*;
- *Immutable*;
- *Marker interface.*

Порождающие шаблоны проектирования:

- *Factory Method*/Фабричный метод;
- *Virtual Constructor*;
- *Abstract Factory*/Абстрактная фабрика, *Kit*;
- *Builder/*Строитель;
- *Prototype*/Прототип;
- *Singleton*/Одиночка;
- *Anonymous subroutine objects*;
- - *Lazy initializatio*n.

Структурные шаблоны (*Structural*):

- *Adapter*/Адаптер;
- *WrapperBridge/*Мост;
- *Handle/Body*;
- *Composite*/Компоновщик;
- *Decorator*/Декоратор;
- *Wrapper Facade*/Фасад;
- *Flyweight*/Приспособленец;
- *Proxy*/Заместитель;
- *Surrogate*;
- *Container*;
- *Extensibility*;
- *Pipes and filters*;

- *Private class data.*

Поведенческие шаблоны (*Behavioral*):

– *Interpreter*/Интерпретатор;
– *Template Method*/Шаблонный метод;
– *Chain of Responsibility*/Цепочка обязанностей;
– *Command*/Команда, *Action, Transaction*;
– Iterator/Итератор, *Cursor*;
– *Mediator*/Посредник;
– *Memento*/Хранитель, *Token;*
– *Observe*r/Наблюдатель, *Dependents, Publish-Subscribe, Listener*;
– *State*/Состояние, *Objects for States*;
– *Strategy*/Стратегия, *Policy*;
– *Visitor/*Посетитель;
– *Event listener*;
– *Single-serving visitor*;
– *Hierarchical visitor.*

Шаблоны параллельного программирования (*Concurrency*):

– *Active Object*;
– *Balking*;
– *Double checked locking*;
– *Guarded suspension*;
– *Half-Sync/Half-Async*;
– *Leaders/followers*;
– *Monitor Object*;
– *Read write lock*;
– *Scheduler*;
– *Thread pool*;
– *Thread-Specific Storage.*

Другие типы шаблонов. На сегодняшний день существует ряд других шаблонов:

– *Model View Controller* (*MVC*);
– *Carrier Rider Mapper*, предоставление доступа к хранимой информации;
– аналитические шаблоны, описывают основной подход для составления требований для программного обеспечения (*requirement analysis*) до начала самого процесса программной разработки;
– коммуникационные шаблоны, описывают процесс общения

между отдельными участниками/сотрудниками организации;

– организационные шаблоны, описывают организационную иерархию предприятия/фирмы;

– Анти-паттерны (*Anti-Design-Patterns*) описывают, как не следует поступать при разработке программ, показывая характерные ошибки в дизайне и в реализации.

6.7. Технологии создания программного обеспечения информационных систем

Технологии создания (ТС) ПО в общем случае можно описать следующей системой понятий [12]:

Технология создания ПО - упорядоченная совокупность взаимосвязанных технологических процессов в рамках ЖЦ ПО.

Технологический процесс - совокупность взаимосвязанных технологических операций.

Технологическая операция - основная единица работы, выполняемая определенной ролью, которая:

– подразумевает четко определенную ответственность роли;

– дает четко определенный результат (набор рабочих продуктов), базирующийся на определенных исходных данных (другом наборе рабочих продуктов);

– представляет собой единицу работы с жестко определенными границами, которые устанавливаются при планировании проекта.

Рабочий продукт - информационная или материальная сущность, которая создается, модифицируется или используется в некоторой технологической операции (модель, документ, код, тест и т.п.). Рабочий продукт определяет область ответственности роли и является объектом управления конфигурацией.

Роль - определение поведения и обязанностей отдельного лица или группы лиц в среде организации-разработчика ПО, осуществляющих деятельность в рамках некоторого технологического процесса и ответственных за определенные рабочие продукты.

Руководство - практическое руководство по выполнению одной или совокупности технологических операций. Руководства включают методические материалы, инструкции, нормативы, стандарты и критерии оценки качества рабочих продуктов.

Инструментальное средство (*CASE*-средство) - программное средство, обеспечивающее автоматизированную поддержку деятельности, выполняемой в рамках технологических операций.

Основным требованием, предъявляемым к современным ТС ПО, является их соответствие стандартам и нормативным документам, связанным с процессами ЖЦ ПО и оценкой технологической зрелости организаций-разработчиков (*ISO* 12207, *ISO* 9000, *CMM* и др.). Согласно этим нормативам, ТС ПО должна поддерживать следующие процессы:

- управление требованиями;
- анализ и проектирование ПО;
- разработка ПО;
- эксплуатация;
- сопровождение;
- документирование;
- управление конфигурацией и изменениями;
- тестирование;
- управление проектом.

Полнота поддержки процессов ЖЦ ПО должна поддерживаться комплексом инструментальных средств (*CASE*-средств).

Соответствие стандартам означает также, в частности, использование общепринятых, стандартных нотаций и соглашений. Для того чтобы проект мог выполняться разными коллективами разработчиков, необходимо использование стандартных методов моделирования и стандартных нотаций, которые должны быть оформлены в виде нормативов до начала процесса проектирования. Несоблюдение проектных стандартов ставит разработчиков в зависимость от фирмы-производителя данного средства, делает затруднительным формальный контроль корректности проектных решений и снижает возможности привлечения дополнительных коллективов разработчиков, смены исполнителей и отчуждения проекта, поскольку число специалистов, знакомых с данным методом (нотацией), может быть ограниченным.

Другим важным требованием является адаптируемость к условиям применения, которая достигается за счет поставки технологии в электронном виде вместе с *CASE*-средствами и библиотеками процессов, шаблонов, методов, моделей и других компонентов, предназначенных для построения ПО того класса систем, на который ориентирована технология. Электронные технологии должны включать средства, обеспечивающие их адапта-

цию и развитие по результатам выполнения конкретных проектов. Процесс адаптации заключается в удалении ненужных процессов и действий ЖЦ ПО, в изменении неподходящих или в добавлении собственных процессов и действий, а также методик, стандартов и руководств.

1. Оценка и выбор ТС ПО

Цель процесса оценки - определение функциональности и качества ТС ПО для последующего выбора. Оценка выполняется в соответствии с конкретными критериями, ее результаты включают как объективные, так и субъективные данные по каждой ТС ПО.

Процессы оценки и выбора тесно взаимосвязаны. По результатам оценки цели выбора и/или критерии выбора и их веса могут потребовать модификации. В таких случаях может понадобиться повторная оценка. Когда анализируются окончательные результаты оценки и к ним применяются критерии выбора, может быть рекомендовано приобретение технологии. Альтернативой может быть отсутствие адекватных технологий, в этом случае рекомендуется разработать новую технологию, модифицировать существующую или отказаться от внедрения.

Процесс выбора включает в себя следующие действия:

– формулировка задач выбора, включая цели, предположения и ограничения;

– выполнение всех необходимых действий по выбору, включая определение и ранжирование критериев, определение технологий-кандидатов, сбор необходимых данных и применение ранжированных критериев к результатам оценки для определения средств с наилучшими показателями;

– выполнение необходимого количества итераций с тем, чтобы выбрать (или отвергнуть) технологии, имеющие сходные показатели.

Типичный процесс оценки и/или выбора может использовать набор критериев различных типов. Каждый критерий должен быть выбран и адаптирован экспертом с учетом особенностей конкретного процесса.

Исходные данные для оценки и выбора - набор параметров (технико-экономических характеристик) ТС ПО:

1) Функциональные характеристики, ориентированные на процессы жизненного цикла ПО (управление проектом, управление требованиями, управление конфигурацией и изменениями, анализ и проектирование ПО и др.).

2) Функциональные характеристики применения (среда функционирования, совместимость с другими ТС ПО, соответствие технологическим стандартам).

3) Характеристики качества (надежность, удобство использования, эффективность, сопровождаемость, переносимость).

4) Общие характеристики (затраты на технологию, лицензионная политика, оценочный эффект от внедрения ТС ПО, инфраструктура, требуемая для внедрения ТС ПО, доступность и качество обучения, сертификация поставщика, поддержка поставщика).

На основе данного набора параметров анализируются и классифицируются существующие ТС ПО. Общий набор критериев, применяемых для оценки ТС ПО, приведен в таблице 6.2.

Таблица 6.2. Критерии, применяемые для оценки ТС ПО

Критерий	Определение
Минимум трудоемкости создания ПО	Количество человеко-месяцев, затрачиваемых на создание ПО с использованием ТС ПО
Максимум продуктивности	Объем работы (измеряемый в количестве строк кода или функциональных точек), приходящийся на единицу трудоемкости (человеко-месяц) при использовании данной ТС ПО
Максимум качества создаваемого ПО	Количество дефектов в рабочих продуктах при использовании данной ТС ПО
Возврат инвестиций	Доход от использования ПО - Затраты на создание и сопровождение ПО/ Затраты на создание и сопровождение ПО
Минимум затрат на сопровождение ПО	Отношение стоимости сопровождения ПО при использовании данной ТС ПО к совокупным затратам на информационные ТС ПО в организации
Минимум времени внедрения ТС ПО	Временной интервал от начала внедрения ТС ПО до выхода на безубыточный уровень (начало возврата инвестиций в ТС ПО)
Минимум затрат на внедрение ТС ПО	Суммарная стоимость приобретения, обучения и сопровождения ТС ПО
Минимальный срок окупаемости затрат на внедрение ТС ПО	Временной интервал от начала внедрения ТС ПО до полной окупаемости затрат на ее внедрение

В результате выполненной оценки может оказаться, что ни одна доступная технология не удовлетворяет в нужной мере всем критериям и не покрывает все потребности проекта. В этом случае может применяться набор средств, позволяющий построить на их базе единую технологическую среду.

2. Технология Rational Unified Process (IBM Rational Software)

На сегодняшний день практически все ведущие компании - разработчики технологий и программных продуктов (*IBM, Oracle, Borland, Computer Associates* и др.) располагают развитыми ТС ПО, которые создавались как собственными силами, так и за счет приобретения продуктов и технологий, созданных небольшими специализированными компаниями. Выбор в качестве примера четырех перечисленных компаний объясняется их ведущими позициями на мировом рынке ТС ПО и присутствием на российском рынке.

Одна из наиболее совершенных технологий, претендующих на роль мирового корпоративного стандарта - *Rational Unified Process* (*RUP*). *RUP* представляет собой программный продукт, разработанный компанией *Rational Software*, которая в настоящее время входит в состав *IBM*.

RUP в значительной степени соответствует стандартам и нормативным документам, связанным с процессами ЖЦ ПО и оценкой технологической зрелости организаций-разработчиков (*ISO* 12207, *ISO* 9000, *CMM* и др.). Ее основными принципами являются:

1) Итерационный и инкрементный (наращиваемый) подход к созданию ПО.

2) Планирование и управление проектом на основе функциональных требований к системе - вариантов использования.

3) Построение системы на базе архитектуры ПО.

Первый принцип является определяющим. В соответствии с ним разработка системы выполняется в виде нескольких краткосрочных мини-проектов фиксированной длительности (от 2 до 6 недель), называемых итерациями. Каждая итерация включает свои собственные этапы анализа требований, проектирования, реализации, тестирования, интеграции и завершается созданием работающей системы.

Итерационный цикл основывается на постоянном расширении и дополнении системы в процессе нескольких итераций с периодической обратной связью и адаптацией добавляемых модулей к существующему ядру

системы. Система постоянно разрастается шаг за шагом, поэтому такой подход называют итерационным и инкрементным.

На рисунке 6.44 показано общее представление *RUP* в двух измерениях. Горизонтальное измерение представляет время, отражает динамические аспекты процессов и оперирует такими понятиями, как стадии, итерации и контрольные точки. Вертикальное измерение отражает статические аспекты процессов и оперирует такими понятиями, как виды деятельности (технологические операции), рабочие продукты, исполнители и дисциплины (технологические процессы).

Согласно *RUP*, ЖЦ ПО разбивается на отдельные циклы, в каждом из которых создается новое поколение продукта. Каждый цикл, в свою очередь, разбивается на четыре последовательные стадии:

1) начальная стадия (*inception*);
2) стадия разработки (*elaboration*);
3) стадия конструирования (*construction*);
4) стадия ввода в действие (*transition*).

Каждая стадия завершается в четко определенной контрольной точке (*milestone*). В этот момент времени должны достигаться важные результаты и приниматься критически важные решения о дальнейшей разработке.

Статический аспект *RUP* представлен четырьмя основными элементами:

1) роли;
2) виды деятельности;
3) рабочие продукты;
4) дисциплины.

Понятие "роль" (*role*) определяет поведение и ответственность личности или группы личностей, составляющих проектную команду. Одна личность может играть в проекте много различных ролей.

Под видом деятельности конкретного исполнителя понимается единица выполняемой им работы. Вид деятельности (*activity*) соответствует понятию технологической операции. Он имеет четко определенную цель, обычно выражаемую в терминах получения или модификации некоторых рабочих продуктов (*artifacts*), таких, как модель, элемент модели, документ, исходный код или план.

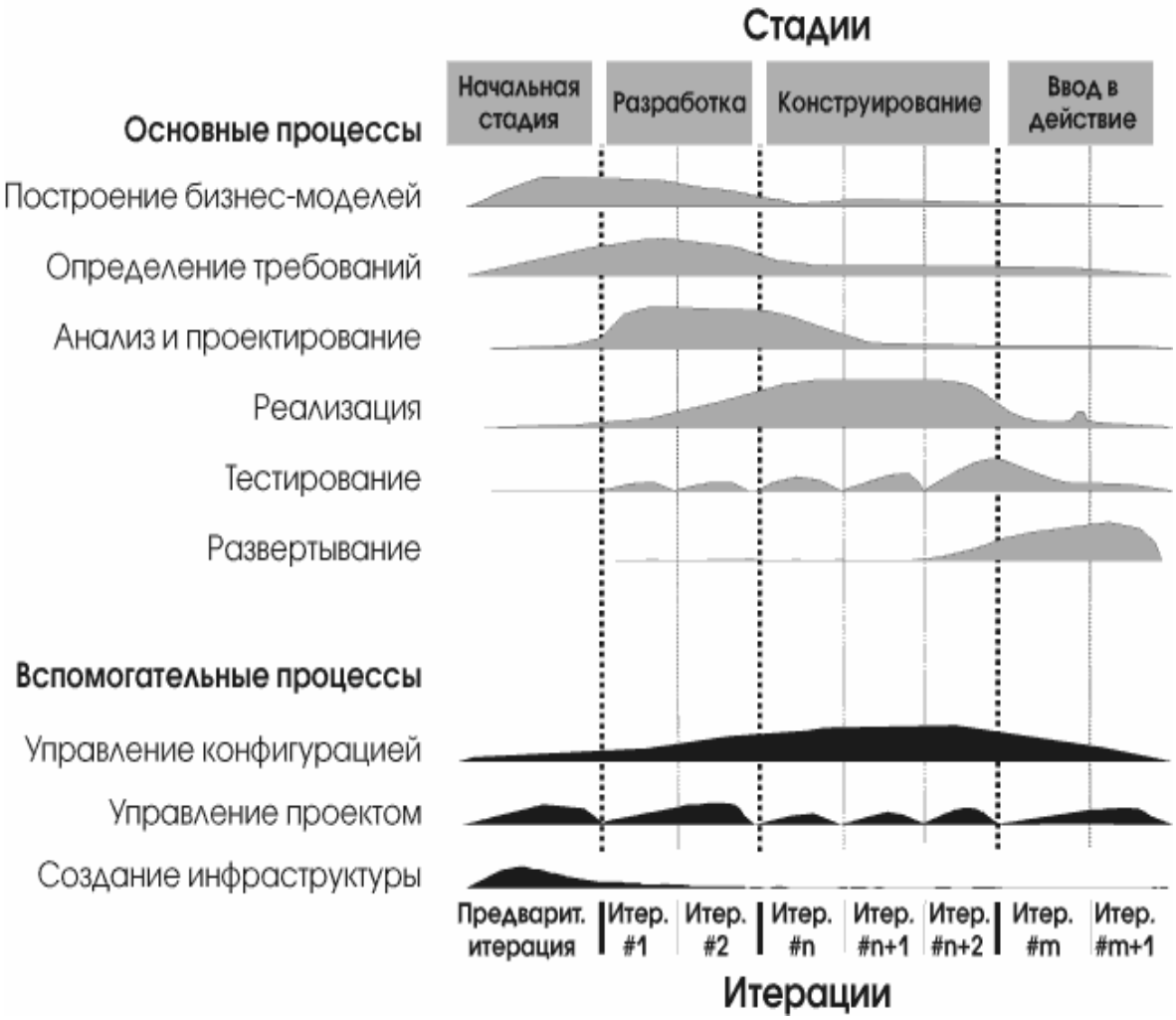


Рис. 6.44. Общее представление RUP

Каждый вид деятельности связан с конкретной ролью. Продолжительность вида деятельности составляет от нескольких часов до нескольких дней, он обычно выполняется одним исполнителем и порождает только один или весьма небольшое количество рабочих продуктов. Любой вид деятельности должен являться элементом процесса планирования. Примерами видов деятельности могут быть планирование итерации, определение вариантов использования и действующих лиц, выполнение теста на производительность. Каждый вид деятельности сопровождается набором руководств (*guidelines*), представляющих собой методики выполнения технологических операций.

Дисциплина (*discipline*) соответствует понятию технологического процесса и представляет собой последовательность действий, приводящую к получению значимого результата.

В рамках *RUP* определены шесть основных дисциплин:

– построение бизнес-моделей;
– определение требований;
– анализ и проектирование;
– реализация;
– тестирование;
– развертывание;

и три вспомогательных:

– управление конфигурацией и изменениями;
– управление проектом;
– создание инфраструктуры.

3. Технология Oracle

Методическую основу ТС ПО корпорации *Oracle* составляет метод *Oracle* (*Oracle Method*) - комплекс методов, охватывающий большинство процессов ЖЦ ПО. В состав комплекса входят:

1) *CDM* (*Custom Development Method*) - разработка прикладного ПО;
2) *PJM* (*Project Management Method*) - управление проектом;
3) *AIM* (*Application Implementation Method*) - внедрение прикладного ПО;
4) *BPR* (*Business Process Reengineering*) - реинжиниринг бизнес-процессов;
5) *OCM* (*Organizational Change Management*) - управление изменениями.

Метод *CDM* оформлен в виде консалтингового продукта *CDM Advantage* - библиотеки стандартов и руководств (включающего также *PJM*). Он представляет собой развитие достаточно давно созданного *Oracle CASE-Method*, известного по использованию *CASE*-средств фирмы *Oracle* и книгам Р. Баркера. По существу, *CDM* является методическим руководством по разработке прикладного ПО с использованием инструментального комплекса *Oracle Developer Suite*, а сам процесс проектирования и разработки тесно связан с *Oracle Designer* и *Oracle Forms*.

В соответствии с *CDM* ЖЦ ПО формируется из определенных этапов (фаз) проекта и процессов, каждый из которых выполняется в течение нескольких этапов (рис. 6.45):

– стратегия (определение требований);
– анализ (формулирование детальных требований к системе);

– проектирование (преобразование требований в детальные спецификации системы);

– реализация (написание и тестирование приложений);

– внедрение (установка новой прикладной системы, подготовка к началу эксплуатации);

– эксплуатация.

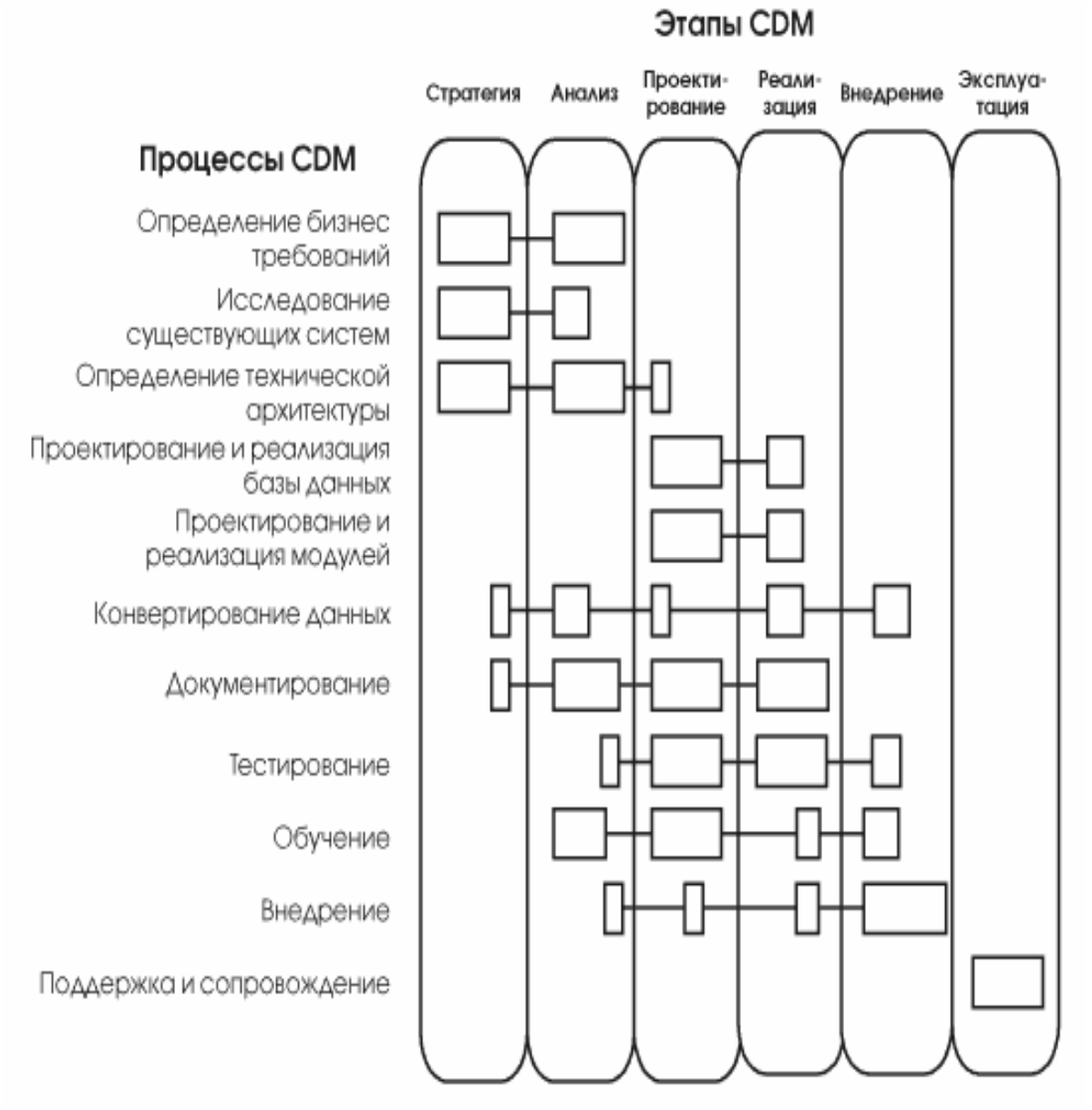


Рис. 6.45. Этапы и процессы CDM

На этапе стратегии определяются цели создания системы, приоритеты и ограничения, разрабатывается системная архитектура и составляется план разработки. На этапе анализа строятся модель информационных по-

требностей (диаграмма "сущность-связь"), диаграмма функциональной иерархии (на основе функциональной декомпозиции системы), матрица перекрестных ссылок и диаграмма потоков данных.

На этапе проектирования разрабатывается подробная архитектура системы, проектируются схема реляционной БД и программные модули, устанавливаются перекрестные ссылки между компонентами системы для анализа их взаимного влияния и контроля над изменениями.

На этапе реализации создается БД, строятся прикладные системы, производится их тестирование, проверка качества и соответствия требованиям пользователей. Создается системная документация, материалы для обучения и руководства пользователей.

На этапах внедрения и эксплуатации анализируются производительность и целостность системы, выполняется поддержка и, при необходимости, модификация системы.

В соответствии с этими факторами в *CDM* выделяются два основных подхода к разработке:

1) Классический подход (*Classic*). Этапы данного подхода представлены на рис. 6.45. Классический подход применяется для наиболее сложных и масштабных проектов, он предусматривает последовательный и детерминированный порядок выполнения задач. Для таких проектов характерно большое количество реализуемых бизнес-правил, распределенная архитектура, критичность приложения. Применение классического подхода также рекомендуется при нехватке опыта у разработчиков, неподготовленности пользователей, нечетко определенной задаче. Продолжительность таких проектов от 8 до 36 месяцев

2) Подход быстрой разработки (*Fast Track*). Данный подход, в отличие от каскадного классического, является итерационным и основан на методе *DSDM* (*Dynamic Systems Development Method*). В этом подходе четыре этапа - стратегия, моделирование требований, проектирование и генерация системы и внедрение в эксплуатацию. Подход используется для реализации небольших и средних проектов с несложной архитектурой системы, гибкими сроками и четкой постановкой задач. Продолжительность проекта от 4 до 16 месяцев.

4. Технология Borland

Компания *Borland* в результате развития собственных разработок и приобретения целого ряда компаний представила интегрированный комплекс инструментальных средств, реализующих управление полным жиз-

ненным циклом приложений (*Application Life Cycle Management, ALM*). В соответствии с технологией *Borland* процесс создания ПО включает в себя пять основных этапов:

1) определение требований;
2) анализ и проектирование;
3) разработка;
4) тестирование и профилирование;
5) развертывание.

Выполнение всех этапов координируется процессом управления конфигурацией и изменениями.

В технологии *Borland* выделяется три уровня интеграции. Функциональная (*touch-point*) интеграция позволяет обратиться из одной системы к функциям другой, выбрав соответствующий пункт меню. Например, интерфейс управления изменениями *StarTeam* непосредственно отображается в системах *Together, C#Builder* и *Visual Studi. Net*. Такая интеграция дает возможность разделять информацию между системами, но не обеспечивает единого рабочего пространства, вынуждает пользователя переключать окна и приводит к дублированию процессов управления структурой проекта. Встроенная (*embedded*) интеграция обеспечивает работу с одной системой непосредственно в среде другой. Например, не выходя из среды разработки *Jbuilder*, можно просматривать графики производительности, которые создает система *Optimizeit*. Самый высокий уровень интеграции - синергетический (*synergistic*), позволяющий сочетать функции двух различных продуктов незаметно для разработчиков. Для большинства продуктов *Borland* и других поставщиков синергетическая интеграция пока остается делом будущего, однако ее принципы уже начинают реализовываться.

5. Технология Computer Associates

Компания *Computer Associates* (*www.ca.com*) предлагает комплексы инструментальных средств поддержки различных процессов ЖЦ ПО:

AllFusion Modeling Suite - интегрированный комплекс *CASE*-средств, включающий следующие продукты:

- *AllFusion Process Modeler* (*BPwin*) - функциональное моделирование;
- *AllFusion ERwin Data Modeler* (*ERwin*) - моделирование данных;

- *AllFusion Component Modeler* (*Paradigm Plus*) - объектно-ориентированный анализ и проектирование с использованием *UML* и возможностью генерации кода;
- *AllFusion Model Manager* (*Model Mart*) - организация совместной работы команды разработчиков;
- *AllFusion Data Model Validator* (*ERwin Examiner*) - проверка структуры и качества моделей данных.
- *AllFusion Change Management Suite* - комплекс средств управления конфигурацией и изменениями.
- *AllFusion Process Management Suite* - средства управления процессами и проектами для различных типов приложений.

BPwin - средство моделирования бизнес-процессов, реализующее метод *IDEF*0, а также поддерживающее диаграммы потоков данных и *IDEF*3. В процессе моделирования *BPwin* позволяет переключиться с нотации *IDEF*0 на любой ветви модели на нотацию *IDEF*3 или *DFD* и создать смешанную модель. *BPwin* поддерживает функционально-стоимостной анализ (*ABC*).

Семейство продуктов *ERwin* представляет собой набор средств концептуального моделирования данных, использующих метод *IDEF*1*X*. *ERwin* реализует проектирование схемы БД, генерацию ее описания на языке целевой СУБД (*Oracle, Sybase, DB*2, *Microsoft SQL Server* и др.) и реверсный инжиниринг существующей БД. *ERwin* выпускается в нескольких конфигурациях, ориентированных на наиболее распространенные средства разработки приложений.

Для управления групповой разработкой используется средство *Model Mart*, обеспечивающее многопользовательский доступ к моделям, созданным с помощью *ERwin* и *BPwin*. Модели хранятся на центральном сервере и доступны для всех участников группы проектирования.

Model Mart удовлетворяет ряду требований, предъявляемым к средствам управления разработкой крупных систем, а именно:

Совместное моделирование. Каждый участник проекта имеет инструмент поиска и доступа к интересующей его модели в любое время. При совместной работе используются три режима: незащищенный, защищенный и режим просмотра. В режиме просмотра запрещается любое изменение моделей. В защищенном режиме модель, с которой работает один пользователь, не может быть изменена другими пользователями. В незащищенном режиме пользователи могут работать с общими моделями в реальном масштабе времени.

Model Mart позволяет формировать библиотеки стандартных решений, включающие наиболее удачные фрагменты реализованных проектов, накапливать и использовать типовые модели, объединяя их при необходимости в "сборки" больших систем. На основе существующих баз данных с помощью *ERwin* возможно восстановление моделей (реверсный инжиниринг), которые в процессе анализа пригодности их для новой системы могут объединяться с типовыми моделями из библиотек моделей.

Управление доступом. Для каждого участника проекта определяются права доступа, в соответствии с которыми, они получают возможность работать только с определенными моделями. Права доступа могут быть определены как для групп, так и для отдельных участников проекта. Роль специалистов, участвующих в различных проектах может меняться, поэтому *в Model Mart* можно определять и управлять правами доступа участников проекта к библиотекам, моделям и даже к специфическим областям модели.

6.8. Методы оценки трудоемкости создания программного обеспечения

Оценка трудоемкости создания ПО является одним из наиболее важных видов деятельности в процессе создания ПО, хотя она и не выделена в стандарте *ISO* 12207 как отдельный процесс.

Модели и методы оценки трудоемкости используются для решения многих задач, среди которых можно выделить следующие:

- разработка бюджета проекта (здесь, прежде всего, требуется точность общей оценки);
- анализ степени риска и выбор компромиссного решения (решение данной задачи позволяет уточнить такие характеристики проекта, как масштабы, возможность повторного использования, количество разработчиков, используемое оборудование и т. д.);
- планирование и управление проектом (полученные результаты обеспечивают распределение и классификацию расходов по компонентам, этапам и операциям);
- анализ затрат на улучшение качества ПО (это позволяет оценить затраты и прибыль от стратегии инвестирования в совершенствование аппаратных средств, технологий и возможности повторного использования).

Недооценка стоимости, времени и ресурсов, требуемых для создания ПО, влечет за собой недостаточную численность проектной команды, чрезмерно сжатые сроки разработки и, как результат, утрату доверия к разработчикам в случае нарушения графика. С другой стороны, перестраховка и переоценка могут оказаться ничуть не лучше. Если для проекта выделено больше ресурсов, чем реально необходимо, причем без должного контроля над их использованием, то ни о какой экономии ресурсов говорить не приходится. Такой проект окажется более дорогостоящим, чем должен был быть при грамотной оценке, и приведет к запаздыванию с началом следующего проекта.

1. Методы оценки и их классификация

Методы оценки трудоемкости создания программного обеспечения можно классифицировать на следующие виды:

1) Алгоритмическое моделирование. Метод основан на анализе статистических данных о ранее выполненных проектах, при этом определяется зависимость трудоемкости проекта от какого-нибудь количественного показателя программного продукта (обычно это размер программного кода). Проводится оценка этого показателя для данного проекта, после чего с помощью модели прогнозируются будущие затраты.

2) Экспертные оценки. Проводится опрос нескольких экспертов по технологии разработки ПО, знающих область применения создаваемого программного продукта. Каждый из них дает свою оценку трудоемкости проекта. Потом все оценки сравниваются и обсуждаются. Этот процесс повторяется до тех пор, пока не будет достигнуто согласие по окончательному варианту предварительной трудоемкости.

3) Оценка по аналогии. Этот метод используется в том случае, если в данной области применения создаваемого ПО уже реализованы аналогичные проекты. Метод основан на сравнении планируемого проекта с предыдущими проектами, имеющими подобные характеристики. Он использует экспертные данные или сохраненные данные о проекте. Эксперты вычисляют высокую, низкую и наиболее вероятную оценку трудоемкости, основываясь на различиях между новым и предыдущими проектами. Самый лучший вариант - это использование накопленных в организации исторических данных, позволяющих сопоставить трудоемкость вашего проекта с трудоемкостью предыдущих проектов аналогичного размера.

4) Закон Паркинсона. Согласно этому закону усилия, затраченные на работу, распределяются равномерно по выделенному на проект време-

ни. Здесь критерием для оценки затрат по проекту являются человеческие ресурсы, а не целевая оценка самого программного продукта. Если проект, над которым работают пять человек, должен быть закончен в течение 12 месяцев, то затраты на его выполнение исчисляются в 60 человеко-месяцев.

5) Оценка с целью выиграть контракт. Затраты на проект определяются наличием тех средств, которые имеются у заказчика. Поэтому трудоемкость проекта зависит от бюджета заказчика, а не от функциональных характеристик создаваемого продукта. Требования приходится изменить так, чтобы не выходить за рамки принятого бюджета.

Каждый метод оценки имеет слабые и сильные стороны. Для работы над большими проектами необходимо применить несколько методов оценки для их последующего сравнения. Если при этом получаются совершенно разные результаты, значит, информации для получения более точной оценки недостаточно. В этом случае необходимо воспользоваться дополнительной информацией, после чего повторить оценку, и так до тех пор, пока результаты разных методов не станут достаточно близкими.

Хорошая оценка трудоемкости разработки ПО:

– создается и поддерживается менеджером проекта и командами архитекторов, разработчиков и тестировщиков, ответственными за выполнение работы;

– воспринимается всеми исполнителями как амбициозная, но выполнимая;

– основывается на подробно описанной и обоснованной модели оценки;

– основывается на данных по аналогичным проектам, которые включают в себя аналогичные процессы, технологии, среду, требования к качеству и квалификации работников;

– подробно описывается таким образом, чтобы все ключевые области риска были хорошо видны, а вероятность успеха оценивалась объективно.

Идеальную оценку можно получить путем экстраполяции хорошей оценки, полученной на основе устоявшейся модели трудоемкости и использующей опыт выполнения множества аналогичных проектов, подготовленных той же командой, которая использовала те же зрелые процессы и инструментарий. Хотя такая ситуация на практике встречается редко, когда команда приступает к осуществлению нового проекта, хорошие оценки могут быть получены обычным путем на более поздних этапах жизненного цикла проекта.

В алгоритмическом моделировании трудоемкости разработки ПО существует в основном два подхода к моделированию: теоретические модели и статистические модели, которые будут рассмотрены ниже. Большинство моделей для определения трудоемкости разработки ПО могут быть сведены к функции пяти основных параметров:

1) размера конечного продукта (для компонентов, написанных вручную), который обычно измеряется числом строк исходного кода или количеством функциональных точек, необходимых для реализации данной функциональности;

2) особенностей процесса, используемого для получения конечного продукта, в частности его способность избегать непроизводительных видов деятельности (переделок, бюрократических проволочек, затрат на взаимодействие);

3) возможностей персонала, участвующего в разработке ПО, в особенности его профессионального опыта и знания предметной области проекта;

4) среды, которая состоит из инструментов и методов, используемых для эффективного выполнения разработки ПО и автоматизации процесса;

5) требуемого качества продукта, включающего в себя его функциональные возможности, производительность, надежность и адаптируемость.

Соотношение между этими параметрами, с одной стороны, и рассчитываемой трудоемкостью с другой, может быть записано следующим образом:

Трудоемкость = (Персонал) • (Среда) • (Качество) • (Размер $^{\text{Процесс}}$).

Наиболее влиятельный фактор оценки трудоемкости в этих моделях - размер программного продукта. Процедура оценки трудоемкости разработки ПО состоит из следующих действий:

1) оценка размера разрабатываемого продукта;

2) оценка трудоемкости в человеко-месяцах или человеко-часах;

3) оценка продолжительности проекта в календарных месяцах;

4) оценка стоимости проекта.

Оценка размера продукта базируется на знании требований к системе. Для такой оценки существуют два основных способа:

1) по аналогии. Если в прошлом приходилось иметь дело с подобным проектом, и его оценки известны, то можно, отталкиваясь от них, приблизительно оценить свой проект;

2) путем подсчета размера по определенным алгоритмам на основе исходных данных - требований к системе.

Проблемы, возникающие при оценке размера ПО:

– Проблема может быть недостаточно хорошо понята разработчиками и (или) заказчиками из-за того, что некоторые факты были упущены или искажены.

– Недостаток или полное отсутствие исторических данных не позволяет создать базу для оценок в будущем.

– Проектирующая организация не располагает стандартами, с помощью которых можно выполнять процесс оценивания (либо в случае наличия стандартов их никто не придерживается); в результате наблюдается недостаток совместимости при выполнении процесса оценивания.

– Менеджеры проектов полагают, что было бы неплохо фиксировать требования в начале проекта, заказчики же считают, что не стоит тратить время на разработку спецификации требований.

– Ошибки, как правило, скрываются, вместо того, чтобы оцениваться и отображаться, в результате чего создается ложное впечатление о фактической производительности.

– Возможности оценивания существенно зависят от субъектов, вовлеченных в процесс оценивания.

– Менеджеры, аналитики, разработчики, тестеры и те, кто внедряет продукт, могут иметь разные представления о процессах оценивания и о возможностях совершенствования продукта.

Точность оценки стоимости и размера ПО в зависимости от стадии проекта определяется графиком, приведенном на рисунке 6.46.

Основными единицами измерения размера ПО являются:

– количество строк кода (*LOC - Lines of Code*);

– функциональные точки (*FP - Function Points*).

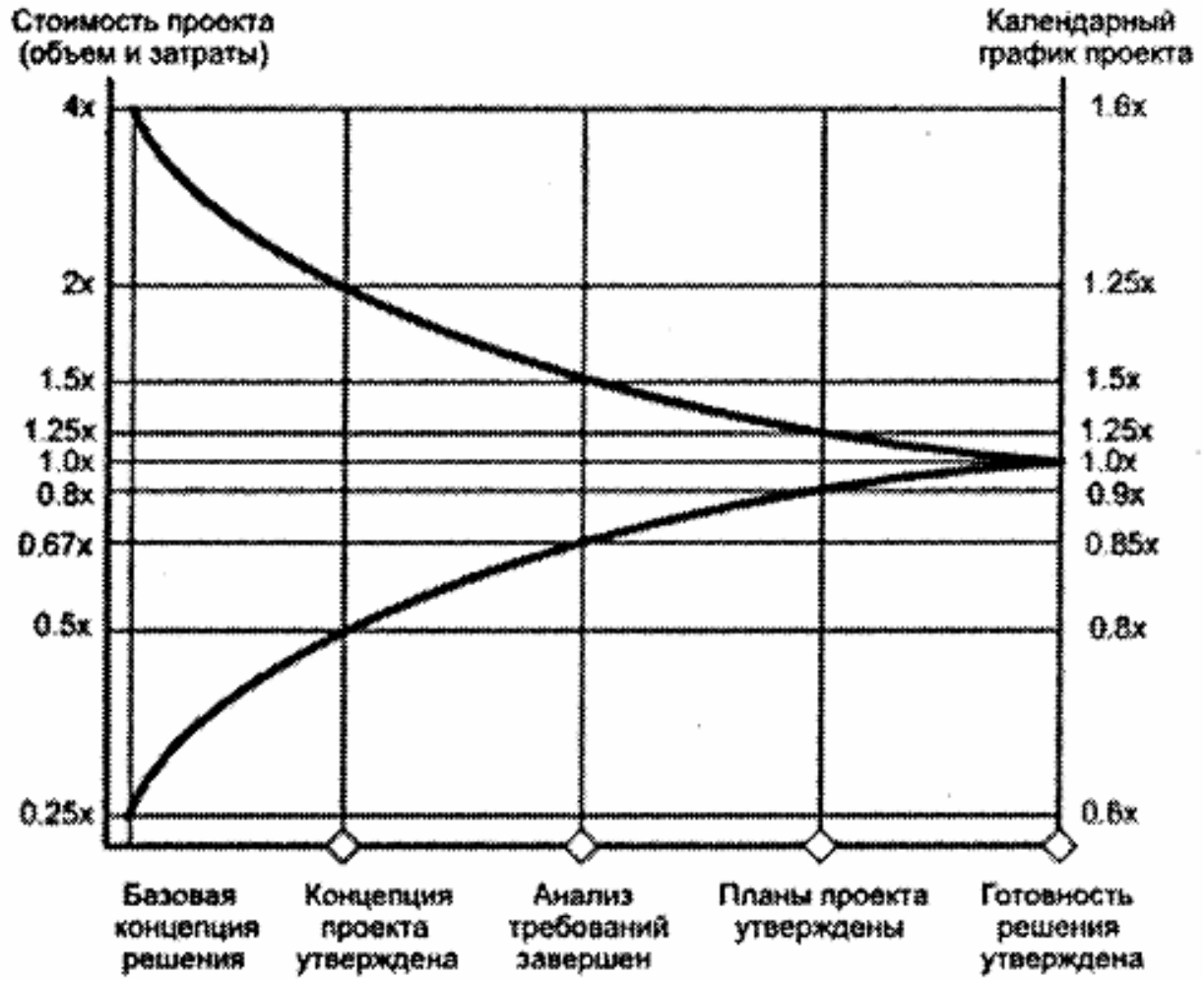


Рис. 6.46. Точность оценки стоимости и размера ПО в зависимости от стадии проекта

2. Методика оценки трудоемкости разработки ПО на основе подсчета количества строк

Количество строк кода - исторически самая известная и до недавнего времени распространенная единица измерения. Однако при ее использовании возникает ряд вопросов:

– Каким образом можно определить количество строк кода до того, как они фактически будут написаны либо просто спроектированы?

– Как показатель количества строк кода может отражать величину трудозатрат, если не будет учитываться сложность продукта, способности (стиль) программиста, либо возможности применяемого языка программирования?

– Каким образом разница в количестве строк кода может быть трансформирована в объем эквивалентной работы?

Эти и другие вопросы привели к тому, что строки кода как единицы измерения получили «дурную репутацию», хотя они по-прежнему остаются наиболее широко используемыми.

Взаимосвязь между *LOC* и затрачиваемыми усилиями не является линейной. Несмотря на появление новых языков программирования, средняя производительность работы программистов за двадцать лет осталась неизменной и составляет около 3000 строк кода на одного программиста в год. Это говорит о том, что уменьшение времени, затрачиваемого на цикл разработки, не может быть достигнуто за счет значительного повышения производительности труда программистов. Причем это не зависит от усовершенствований языка программирования, усилий со стороны менеджеров или сверхурочных работ. На самом деле первостепенное значение имеет набор функциональных свойств и качество ПО, а не количество строк кода.

Преимуществами использования *LOC* в качестве единиц измерения являются:

– широкое распространение и легкая адаптируемость;

– возможность сопоставления методов измерения размеров и производительности в различных группах разработчиков;

– непосредственная связь с конечным продуктом;

– легкая оценка до завершения проекта;

– оценка размеров ПО на основе точки зрения разработчика - физическая оценка созданного продукта (количество написанных строк кода).

Недостатками применения *LOC* являются:

– *LOC* затруднительны в применении при оценке размера ПО на ранних стадиях разработки;

– строки исходного кода могут различаться в зависимости от типов языков программирования, методов проектирования, стиля и способностей программиста;

– применение методов оценки с помощью подсчета количества строк кода не регламентируется промышленными стандартами (например, *ISO*);

– разработка ПО может быть связана с большими затратами, которые прямо не зависят от размеров программного кода - «фиксированными затратами», такими, как спецификации требований и пользовательские документы, не включенными в прямые затраты на кодирование;

– программисты могут быть незаслуженно премированы за достижение высоких показателей *LOC* в случае, если руководство по ошибке посчитает это признаком высокой продуктивности, но при этом будет отсутствовать тщательно разработанный проект; исходный код не является самоцелью при создании продукта - главную роль играют функциональные свойства и показатели производительности;

– при подсчете количества *LOC* следует различать автоматически и вручную созданный код - эта задача является более сложной, чем простой подсчет, который может быть выполнен на основе листинга, сгенерированного компилятором, либо с помощью утилиты, выполняющей подсчет строк кода;

– показатели *LOC* не могут применяться при осуществлении нормализации в случае, если применяемые платформы разработки или языки являются различными;

– единственный способ учета с помощью *LOC* по отношению к разрабатываемому ПО заключается в использовании метода аналогии на основе сравнения функциональных свойств у подобных программных продуктов, либо в использовании мнений экспертов (однако эти методы не относятся к числу точных);

– генераторы кода зачастую продуцируют чрезмерный объем кода, в результате чего искажаются показатели *LOC*.

Результатом этих соображений явилось осознание необходимости другой единицы измерения, в качестве которой стали выступать функциональные точки.

3. Методика оценки трудоемкости разработки ПО на основе функциональных точек

Определение числа функциональных точек является методом количественной оценки ПО, применяемым для измерения функциональных характеристик процессов его разработки и сопровождения независимо от технологии, использованной для его реализации.

Подсчет функциональных точек, помимо средства для объективной оценки ресурсов, необходимых для разработки и сопровождения ПО, применяется также в качестве средства для определения сложности приобретаемого продукта с целью принятия решения о покупке или собственной разработке.

Метод разработан на основе опыта реализации множества проектов создания ПО и поддерживается международной организацией *IFPUG* (*International Function Point User Group*).

Согласно данной методике трудоемкость вычисляется на основе функциональности разрабатываемой системы, которая, в свою очередь, определяется на основе выявления функциональных типов - логических групп взаимосвязанных данных, используемых и поддерживаемых приложением, а также элементарных процессов, связанных с вводом и выводом информации (рис. 6.47).

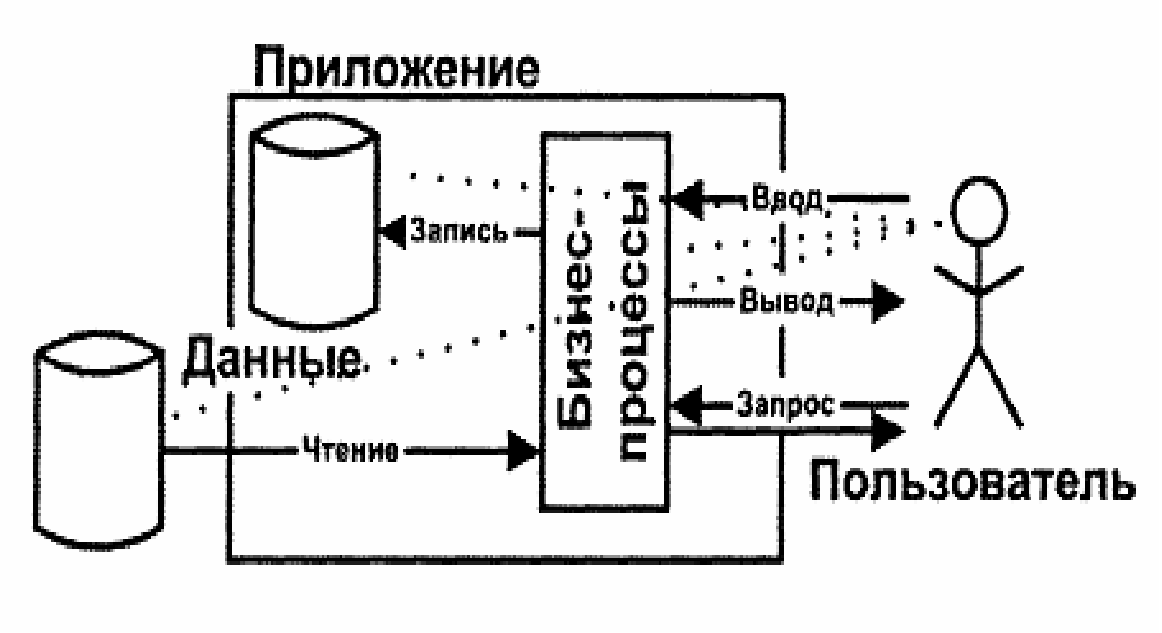


Рис. 6.47. Выявление функциональных типов

Порядок расчета трудоемкости разработки ПО следующий:

- определение количества и сложности функциональных типов приложения;
- определение количества связанных с каждым функциональным типом элементарных данных (*DET*), элементарных записей (*RET*) и файлов типа ссылок (*FTR*);
- определение сложности (в зависимости от количества *DET, RET* и *FTR*);
- подсчет количества функциональных точек приложения;
- подсчет количества функциональных точек с учетом общих характеристик системы (рис.6.48);

– оценка трудоемкости разработки (с использованием различных статистических данных).

Рис. 6.48 Определение количества функциональных точек

4. Методы, основанные на экспертных оценках

4.1. Метод Дельфи

Метод Дельфи был разработан в корпорации «Рэнд» в конце 1940-х годов и использовался первоначально для прогнозирования будущих событий (отсюда метод и получил свое название по сходству с предсказаниями Дельфийского оракула в Древней Греции). Позднее метод использовался для принятия решений по спорным вопросам. На предварительном этапе участники дискуссии должны без обсуждения с другими ответить на ряд вопросов, относительно их мнения по спорному вопросу. Затем ответы обобщаются, табулируются и возвращаются каждому участнику дискуссии для проведения второго этапа, на котором участникам снова предстоит дать свою оценку спорного вопроса, но на этот раз, располагая мнениями других участников, полученными на первом этапе. Второй этап завершается сужением и выделением круга мнений, отражающих некоторую общую оценку проблемы.

Изначально в методе Дельфи коллективное обсуждение не использовалось; обсуждение между этапами метода было впервые применено в обобщенном методе Дельфи. Метод достаточно эффективен в том случае, если необходимо сделать заключение по некоторой проблеме, а доступная информация состоит больше из «мнений экспертов», чем из строго определенных эмпирических данных.

4.2. Метод декомпозиции работ

Долгое время, являясь фактическим стандартом в практике разработки как программного, так и аппаратного обеспечения, метод декомпозиции работ (МДР) представляет собой способ иерархической организации элементов проекта, упрощающий задачу составления бюджета проекта и контроля над расходованием средств. Он позволяет определить, на что именно расходуются средства. Если с каждой категорией расходов, связанной с тем или иным элементом иерархии проекта, сопоставить некоторую вероятность, можно определить ожидаемую сумму расходов на разработку, начиная с некоторых структурных элементов проекта и заканчивая совокупными затратами на выполнение всего проекта. В рамках данного метода экспертные оценки применяются для составления наиболее нужных спецификаций элементов структуры проекта и для сопоставления каждому элементу определенной степени вероятности категории расходов, связанной с ним. Методы, основанные на экспертных оценках, пригодны для проектов, связанных с разработкой принципиально нового ПО, и для совокупной оценки проектов, но содержат ряд «узких мест», упомянутых выше. Кроме того, методы, основанные на экспертных оценках, плохо масштабируемы, что затрудняет масштабный анализ чувствительности. Подходы, в основу которых положен МДР, хорошо пригодны для планирования и управления.

Метод декомпозиции работ для ПО предполагает существование двух иерархий элементов проекта. Одна из них отражает структуру ПО, другая представляет собой упорядоченные стадии разработки ПО. Иерархия структуры ПО отражает фундаментальную структуру ПО и показывает функции и местоположение каждого элемента в рамках ПО. Иерархия стадий разработки показывает основные этапы, связанные с разработкой данного компонента ПО.

Наряду с процессом оценки МДР широко применяется для расчета себестоимости разработки ПО. Каждому элементу иерархий МДР можно приписать свой бюджет и уровень издержек, что позволяет персоналу составлять отчеты о количестве рабочего времени, которое было затрачено на выполнение заданной задачи в рамках проекта или какого-либо компонента проекта, которые затем могут быть обобщены для административного контроля над использованием средств.

Если организация применяет МДР как стандарт для всех проектов, то с течением времени формируется ценная база данных, отражающая распределение расходов на разработку ПО. На основе этих данных может быть разработана собственная модель оценки расходов, подстроенная под практический опыт организации.

УПРАВЛЕНИЕ ПРОЕКТИРОВАНИЕМ ИНФОРМАЦИОННЫХ СИСТЕМ. ПЛАНИРОВАНИЕ И КОНТРОЛЬ ПРОЕКТНЫХ РАБОТ

7.1. Организация работ по проектированию информационных систем

Процесс проектирования ИС включает в себя большое количество взаимосвязанных между собой разнообразных элементов и предполагает построение соответствующей системы управления. В качестве объекта разработки проекта могут выступать либо вся ИС для предприятия заказчика, либо только отдельная подсистема или совокупность подсистем, либо отдельные работы, например установка вычислительной сети, проведение консалтинговых работ по оценке эффективности информационной системы и т.д.

Проект как вид деятельности проектирующей организации отличается следующими особенностями:

- направлен на достижение конкретных целей;
- включает в себя координированное выполнение взаимосвязанных действий;
- имеет ограниченную протяженность во времени с определенным началом и концом;
- все проекты в определенной степени неповторимы и уникальны.

Организация процессов разработки проекта ИС отличается значительной сложностью. К причинам, обусловливающим сложность данных процессов, следует отнести, прежде всего:

- масштабы разработки ИС;
- взаимосвязь различных по своей природе элементов проекта ИС (информационные, программные и технические средства обработки информации; математические модели; методы и средства проектирования; специалисты-разработчики; элементы проекта системы и др.);
- различные факторы старения указанных элементов;
- разный временной цикл существования и темпов обновления элементов;
- длительность процесса проектирования системы;
- индивидуальность проекта, обусловленную спецификой объекта

проектирования;

– коллективный характер труда многих специалистов различной квалификации.

Под управлением проектом подразумевается деятельность, направленная на реализацию проекта с максимально возможной эффективностью при заданных ограничениях по времени, в денежных средствах и материальных ресурсах, а также по качеству конечных результатов проекта (документированных, например, в техническом задании). Управление как процесс характеризуется следующими компонентами: целью управления, ограничениями, объектом и субъектом управления, контуром управления, методами и средствами управления.

Глобальной целью управления проектированием ИС является получение проекта с заданными пользователем параметрами. Ограничениями могут выступать сроки проектирования, требуемые ресурсы. Объектом управления является процесс проектирования ИС как деятельность коллектива разработчиков системы, а также состояние используемых ресурсов.

Процесс проектирования ИС имеет специфические особенности, которые, в свою очередь, определяют специфику управления проектированием:

1) Процесс проектирования ИС по своему характеру является творческим. Поэтому при отсутствии достаточно полного формализованного перечня операций проектирования и состояний проекта в процессе его разработки управление проектированием носит ситуационный характер.

2) Пользователь на этапе разработки системы может изменять требования к качеству системы, срокам и затратам проектирования. В связи с отсутствием общепринятых надежных способов оценки качества проектных решений затруднен его контроль.

3) Стремление разработчиков к индивидуальному характеру труда приводит к невысокой степени организации контроля и координации деятельности отдельных разработчиков проекта.

Выделение субъекта управления связано с разделением труда в группе специалистов в процессе проектирования ИС. Управление проектными работами в этом случае может осуществляться на нескольких уровнях:

– руководства проектной организации;

– руководства обеспечивающих подразделений (например, планово-производственного отдела и т.п.);

– руководства функциональными подразделениями;

– руководителей проектов (главных конструкторов);

– руководителей проектных групп (ответственных исполнителей).

На каждом уровне управления проектными работами существует определенное представление о процессе проектирования, частных целях и задачах управления процессом проектирования ИС, что определяется кругом должностных обязанностей, характером выполняемых функций субъектов управления каждым уровнем, набором используемых методов и средств управления.

Управление проектированием, как правило, рассматривают в двух аспектах: организационном и функциональном.

В организационном аспекте управление проектированием рассматривается по уровням организационно-административной структуры с соответствующими правами и обязанностями субъектов процесса проектирования.

В функциональном аспекте управление проектированием рассматривается как применение соответствующих методов и средств организации и ведения проектных работ.

В данной главе будет рассмотрен организационный аспект управления проектированием. Организация работ по проектированию определяется порядком взаимодействия между несколькими сторонами, участвующими в этом процессе: пользователем, заказчиком, администратором и разработчиком.

Пользователь - это организация или группа подразделений, которые используют результаты обработки информации на ЭВМ. Для ИС под пользователем понимают, прежде всего, административно-управленческий аппарат, для которого создается эту система. Пользователь выполняет следующие функции:

– формирует исходные данные для проектирования и обработки;

– определяет состав задач для автоматизации;

– определяет основные требования к задачам и режим функционирования системы.

Заказчик - это ответственное лицо, под которым понимается организация или подразделение и которое выполняет функции:

– формирует требования к системе и ее частям;

– выдает техническое задание, финансирует разработку ИС;

– обеспечивает проведение комплекса мероприятий по ее созданию;

– проводит внедрение и прием проекта ИС.

При этом заказчик несет ответственность перед пользователем за соответствие состава и характеристик решаемых задач, режима функциони-

рования ИС исходным данным пользователя, за сроки создания системы, правильность использования ресурсов в процессе проектирования.

Администратор - ответственное лицо, которое выполняет эксплуатацию программно-технических средств и информационного и методологического обеспечения ИС (технологические и инструкционные карты).

Администратор несет ответственность перед пользователем за правильность результатов работы ИС и их своевременность, а перед заказчиком и разработчиком – за соблюдением условий эксплуатации, требовании к технической документации.

Разработчик - это ответственное лицо (организация или подразделение), которое выполняет следующие функции:

- разрабатывает ИС по техническому заданию заказчика;
- принимает участие во внедрении;
- осуществляет сдачу проекта заказчику;
- осуществляет авторское сопровождение проекта.

Разработчик несет ответственность перед заказчиком за правильность реализации требований ТЗ на ИС, научно-технический уровень разработки, сроки проведения работ, качество проектной документации, правильность расхода денежных ресурсов.

Под разработчиком понимается как одна организация, так и некоторая совокупность организаций, в которую входят головная организация и организации-соисполнители.

Существует несколько типов схем организации работ с участием четырех сторон, выбор которых зависит от объема заказа.

1. Если заказ имеет небольшие размеры по стоимости и по продолжительности работ, то принимают; первую схему, в которой в одном лице выступают заказчик, разработчик и администратор (рис. 7.1).

К преимуществу данной схемы можно отнести минимальное количество организаций - участников процесса, минимальные сроки и стоимость разработки. Однако совмещение в одной организации функций разрабатывающей стороны и принимающей стороны имеет ряд существенных недостатков:

- отсутствует действенный контроль за научно-техническим уровнем разработки, сроками выполнения работ;
- не достигается высокого профессионального уровня разработчиков.

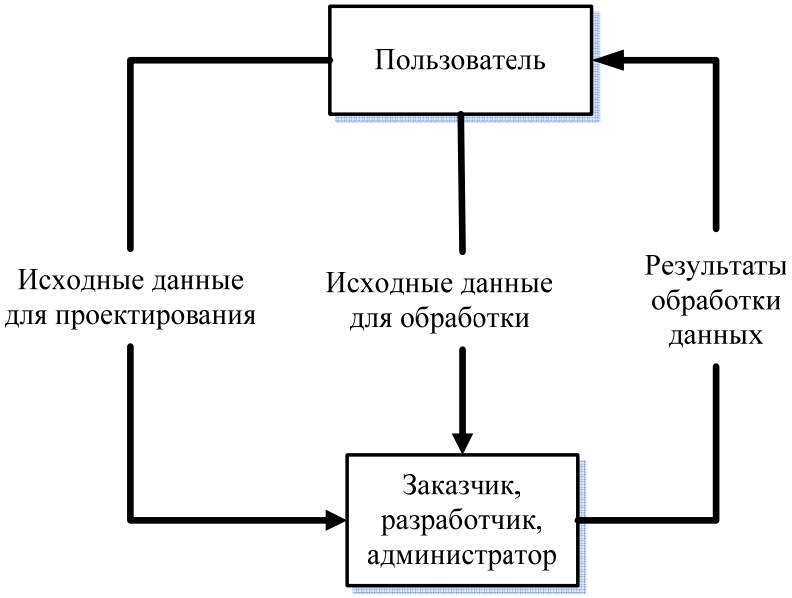


Рис. 7.1. Схема организации работ для небольших заказов

2. Для больших и сложных заказов применяют схему, согласно которой функции разработчика отделяются от функций заказчика и администратора и выполняются другой организацией (рис. 7.2).

К преимуществам данной схемы можно отнести:

– рациональное распределение функций между сторонами, участвующими в создании и эксплуатации ИС;

– возможность привлечения к разработке ИС специализированных организаций (НИИ, СКВ).

Однако и эта схема имеет недостатки:

– отсутствие прямой связи между разработчиком и пользователем, что создает трудности в своевременном получении и детализации исходных данных для проектирования;

– определенные трудности при приеме проекта в эксплуатацию из-за желания администраторов получить методологическое обеспечение задач, максимально соответствующее идеальным условиям эксплуатации, что, в свою очередь, требует больших сроков и объемов по доработке проекта.

Рис.7.2. Схема организации работ при наличии сложного заказа

3. Отделение заказчика от разработчика позволяет последнему привлекать к своей работе организации-соисполнителей разных уровней иерархии (рис. 7.3), что, в свою очередь, позволяет использовать труд специализированных и профессиональных организаций.

Основными документами, регулирующими отношения заказчика и проектировщика, являются техническое задание и договор на проведение работ. Иногда заказчик курирует частные ТЗ, если организациям выделены важные функции, которые имеют достаточно сложную структуру.

Переход экономики страны на рыночные отношения привел к тому, что в области проектирования ИС появится самостоятельный рынок услуг по проектированию, покупке и установке вычислительной техники, разработке локальных сетей, прокладке сетевого оборудования и обучению пользователей, выполняемых компаниями, называемыми «системными интеграторами».

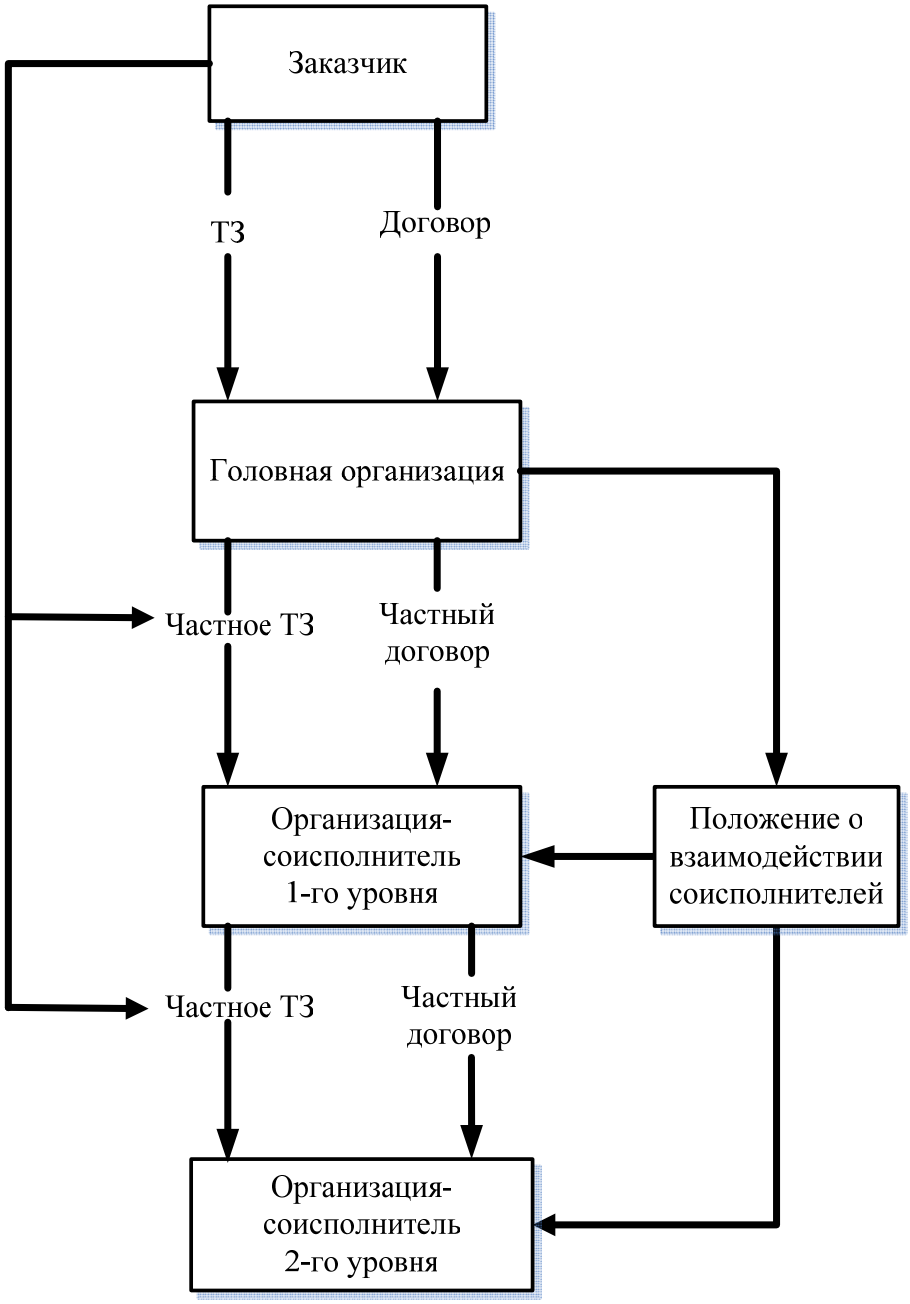


Рис.7.3. Схема организации работ с использованием организаций-соисполнителей

Компания - «системный интегратор» призвана выполнять комплексное решение задач заказчика при построении ИС, поскольку заказчик готов переложить детальную проработку и реализацию проекта на плечи системного интегратора, определив лишь исходные данные и задачи, которые должна решать реализуемая ИС. Такая компания выполняет, как правило, следующий набор функций:

– продажа (дистрибуция, поставка для проектов) аппаратного обеспечения;

– продажа (дистрибуция, поставка для проектов) программного обеспечения;

– консалтинг, проектные работы, сервис, техническая поддержка, обучение.

Объектами, с которыми работают специалисты фирм-интеграторов, являются:

– офисные и корпоративные сети;

– многоуровневые системы хранения информации;

– системы управления технологическими процессами;

– корпоративные автоматизированные информационные системы для крупных банков, нефтегазовых компаний, крупных химических комбинатов и т.д.

Участие системного интегратора на всех этапах процесса проектирования дает возможность создавать более эффективные информационные системы.

Другим вариантом организации системной интеграции является реализация проектов от консалтинга до создания прикладной системы - со сдачей заказчику всей информационной системы «под ключ» с привлечением партнеров для реализации некоторых составляющих проекта.

Под «проектной интеграцией» понимается умение находить составные части для решения комплексной задачи, умение распределять ответственность и составлять план-график работ для того, чтобы задача была действительно решена. Проектная интеграция - это интеграция существующих проектов, привлечение и использование нужных ресурсов [11].

Фирмы - проектные интеграторы рассматривают в качестве партнеров отделы автоматизации предприятий-заказчиков, которые сами для себя выполняют большую работу по автоматизации различных типовых областей, заказывать которую для крупных структур было бы накладно. Поэтому, работая с крупными организациями, проектные интеграторы воспринимают их отделы автоматизации как фирмы, которые производят определенную продукцию в этой области, и стараются договориться с ними об использовании их продукта и рабочего времени в целях осуществления проекта.

Кроме того, функцией фирм-проектных интеграторов является налаживание структурных связей между разными разработчиками с тем, чтобы они могли на коммерческой основе обмениваться проектными решениями. Когда

же появляется много субподрядчиков, то появляется необходимость управления проектом, слежения за исполнением, чтобы все части были правильно состыкованы, сданы в срок и т.д.

Проектный интегратор отличается от системного интегратора тем, что, во-первых, максимально активно использует аутсортинг*) и, во-вторых, делает это максимально эффективно и с минимальными затратами, так, чтобы проект начал работать в реальном времени и как можно быстрее дал экономический эффект. Если системный интегратор создает новые информационные системы, то проектный совершенствует работу ИС путем поиска на рынке уже существующих, внедренных решений и объединения их.

7.2. Организационные формы управления проектированием информационных систем

Формы управления, применяемые в организациях-разработчиках ИС, зависят от выполняемых работ. Как правило, в организациях-разработчиках выполняются, как об этом было сказано выше, работы, связанные с проектированием ИС, с поддержкой и сопровождением ИС.

Формирование организационных форм управления в организациях - разработчиках ИС осуществляется по функциональному, проектному (целевому) и матричному принципам.

Функциональный принцип построения структуры организации используется при выполнении задач проектирования постоянного характера. Для выполнения каждого вида задач, например, разработки постановки задач, информационного обеспечения и т.п., формируются функциональные подразделения из специалистов определенного профиля.

Подобная организационная структура обладает высокой степенью централизации управления, ей присущ авторитарный стиль руководства. В области разработки ИС функциональная структура организации встречается редко.

Для построения организационных структур проектных организаций наиболее часто используется проектный принцип. На основе этого принципа формируется организационное подразделение - проектная группа (проект), которая предназначена для одноразовой разработки ИС.

---

*)Аутсортинг (англ.) – использование внешней организации (поставщика) для обработки банковских и др. финансовых данных при осуществлении коммерческих операций.

Специалисты проектной группы образуют автономную организационную единицу, руководитель (главный конструктор) которой имеет соответствующие полномочия и несет полную ответственность за результаты деятельности проектного коллектива, который после выполнения проекта может быть расформирован.

Матричное построение организационных структур предполагает формирование в организации - разработчике ИС из специалистов функциональных подразделений проектных групп для разработки конкретных проектов. При этом специалисты не теряют принадлежности к соответствующему функциональному подразделению и находятся в двойном подчинении: у руководителя проекта (ответственность по проекту) и у руководителя функционального подразделения (организационная ответственность).

Руководитель проекта наделен всей полнотой ответственности за достижение цели проектирования и значительными правами распорядительства, которые делегируются ему вышестоящим руководством.

При одновременной разработке нескольких проектов в организационную структуру, как правило, вводится планово-производственное подразделение, главной задачей которого являются балансирование ресурсов, потребляемых всеми проектами, и обеспечение координации и текущих изменений в проектах.

Процесс проектирования состоит из многих этапов, на каждом из которых решаются различные задачи, его реализация требует различных знаний, в нем участвуют специалисты разных профилей и квалификации. Поэтому существует объективная потребность разделения труда в коллективе разработчиков системы.

Разделение труда способствует повышению производительности труда, в том числе и проектировщиков ИС; накоплению опыта и знаний; повышению качества проектных решений.

В проектных коллективах разделение труда между специалистами осуществляется, как правило, на основе одного из двух следующих принципов: пооперационного (технологического) или подсистемного.

Разделение труда на пооперационной (технологической) основе базируется на свойстве декомпозируемости процесса проектирования ИС на технологические операции, которые выполняются отдельными специалистами или группами специалистов. В этом случае требуется четкая регламентация интерфейсов между операциями. С этим связаны высокие требования к документи-

рованию проекта. Связь между разделением труда и накоплением специальных знаний и опыта приводит к профессиональной специализации разработчиков ИС (например, системный аналитик, постановщик задач, программист, оператор и т.д.).

Разделение труда в коллективе разработчиков ИС на основе пооперационного принципа, как правило, затруднительно.

Подсистемное разделение труда в коллективе разработчиков ИС базируется на свойстве декомпозируемости проекта на подсистемы, каждая из которых независимо от числа технологических операций проектирования разрабатывается отдельной группой специалистов. В этом случае предполагаются стандартизация и унификация интерфейсов между подсистемами на каждом этапе процесса проектирования ИС. Накопление знаний и опыта приводит к системной специализации разработчиков ИС (например, специалистов по информационному обеспечению, техническому обеспечению, экспертным системам и т.п.) или к специализации по разработке компонентов ИС (информационной базы, пользовательского интерфейса и т.п.).

На практике при разделении труда в проектных коллективах возможно использование обоих вышеназванных принципов.

Выбор целесообразного разделения труда разработчиков ИС зависит от ряда факторов, влияющих с разной степенью на решение проблемы. Наиболее существенными факторами являются:

- потенциал коллектива разработчиков;
- объем и сложность разрабатываемых проектов;
- технология проектирования системы;
- модель жизненного цикла системы.

Степень влияния каждого фактора в конкретных случаях приводит к большому разнообразию разделения труда и связанных с ним организационных форм управления проектированием ИС в проектной группе. При этом используются, как правило, три типовые организационные структуры Проектной группы: открытая, централизованная и децентрализованная.

Открытая организационная структура отличается тем, что закрепленного организационного распределения обязанностей нет. Каждый член коллектива разработчиков является неформальным руководителем на этапе разработки системы, где он более других квалифицирован. Обязанности на отдельных этапах распределяются между разработчиками в соответствии с опытом и способностями.

Административный руководитель в группе осуществляет, как правило, следующие действия:

- взаимодействие с заказчиком;
- планирование и контроль сроков;
- распределение ресурсов, координацию работ;
- отчетность перед руководством организации (если группа работает в составе таковой).

Такая организационная структура формируется из 7-10 человек для творческих решений задач и рекомендуется для работ, выполняемых на ранних этапах проектирования системы - проведении обследования предметной области (объекта управления), анализе и разработке концепции проекта. Такая численность проектировщиков дает возможность полного обмена информацией между ними, а также иметь относительно невысокие затраты на администрирование.

Открытая организационная структура позволяет варьировать количество разработчиков, привлекая для выполнения работ наиболее квалифицированных специалистов, что способствует повышению качества проекта.

Централизованная организационная структура проектной группы предусматривает в качестве руководителя специалиста высокой квалификации, осуществляющего административное и техническое руководство. Он же является основным посредником между группой, заказчиком проекта и внешними организациями.

Данная структура наиболее, приемлема для решения задач, имеющих жесткие ограничения по срокам и затратам на разработку системы. Численность такой группы до семи человек. Особенностью данной организационной структуры проектной группы является четкое распределение функций и полномочий между специалистами. Результаты работы каждого члена группы предоставляются в распоряжение всех участников процесса проектирования. Недостаток заключается в отсутствии проявления инициативы конкретных исполнителей.

Децентрализованная организационная структура проектной группы имеет свойство двух вышеизложенных структур. Данная организационная структура применяется в коллективах с большой численностью разработчиков (свыше 10 человек), осуществляющих проектирование больших ИС, декомпозируемых на подсистемы (контуры, модули) и комплексы задач.

В этом случае руководитель проекта осуществляет управление группой старших специалистов, отвечающих за разработку крупных частей системы, а

те, в свою очередь, осуществляют руководство младшими специалистами, которые поддерживают между собой горизонтальные связи в процессе проектирования. Как правило, младшие специалисты объединены в подгруппы по технологической специализации.

Следует заметить, что для разработки ИС в состав проектной группы конкретной организации могут привлекаться на временное сотрудничество специалисты со стороны пользователей и специалисты-разработчики для решения специфических задач, требующих высокой квалификации и практического опыта в конкретной проблемной области. Участие со стороны пользователя целесообразно на этапе системного анализа и разработки требований к системе, а также при проектировании пользовательского интерфейса.

Независимо от формы организационной структуры коллектива разработчиков для обеспечения процессов проектирования должно быть создано специальное подразделение, организующее использование ЭВМ, автоматизированных рабочих мест проектировщиков, терминальных станций и т.п.; сопровождение базового программного обеспечения; текущий ремонт технических средств.

7.3. Основные компоненты процесса управления проектированием информационных систем

Управление проектированием ИС в функциональном аспекте рассматривается как совокупность взаимосвязанных процессов. Под процессами управления понимаются действия и процедуры, связанные с решением конкретных задач или реализацией функций управления, к которым относятся:

– процессы инициации, связанные с принятием решения о начале выполнения проекта или какого-либо очередного этапа или фазы его;

– процессы планирования - совокупность процедур, связанных с определением целей и критериев успеха проекта и разработкой рабочих схем их достижения;

– процессы исполнения, предназначенные для координации людей и других ресурсов для выполнения плана;

– процессы анализа, дающие возможность определить соответствие плана и исполнения проекта поставленным целям и критериям успеха и принять решения о необходимости применения корректирующих воздействий;

– процессы оперативного управления или регулирования - совокупность процедур, предназначенных для определения необходимых корректирующих воздействий, их согласования, утверждения и применения;

– процессы завершения - процессы формализации выполнения проекта и составления отчетности.

Процессы управления проектами накладываются друг на друга, и происходят с разными интенсивностями на всех стадиях проекта. Кроме того, процессы управления проектами связаны между собой своими результатами. Результат выполнения одного становится исходной информацией для другого. Наконец, имеются взаимосвязи групп процессов различных фаз (этапов) проекта. Например, закрытие одной фазы может являться входом для инициации следующей фазы (пример: завершение фазы проектирования требует одобрения заказчиком проектной документации, которая необходима для начала реализации). В реальном проекте фазы могут не только предшествовать друг другу, но и накладываться. Внутри каждой группы процессы управления проектами связаны друг с другом через свои входы и выходы:

Входы - документы или документированные показатели, согласно которым процесс исполняется.

Выходы - документы или документированные показатели, являющиеся результатом процесса.

Методы и средства - механизмы, по которым вход преобразуется в выход.

Рассмотрим состав и содержание выделенных групп процессов.

Процессы инициации. Инициация включает единственный подпроцесс - авторизацию, т.е. решение начать следующую фазу проекта.

Процессы планирования. Планирование имеет большое значение для проекта и включает сравнительно много процессов. Некоторые из процессов планирования имеют четкие логические и информационные взаимосвязи и выполняются в одном порядке практически во всех проектах. Так, например, сначала следует определить, из каких работ состоит проект, а уж затем рассчитывать сроки выполнения и стоимость проекта. Эти основные процессы выполняются по несколько раз на протяжении каждой фазы проекта.

К основным процессам планирования проектных работ относятся [11]:

1) Планирование целей - разработка постановки задачи (проектное обоснование, основные этапы и цели проекта).

2) Декомпозиция целей - разделение этапов проекта на более мелкие и более управляемые компоненты для обеспечения более действенного контроля.

3) Определение состава операций (работ) проекта - составление перечня операций, из которых состоит выполнение различных этапов проекта.

4) Определение взаимосвязей операций - составление и документирование технологических взаимосвязей между операциями.

5) Оценка длительностей или объемов работ - оценка количества рабочих временных интервалов либо объемов работ, необходимых для завершения отдельных операций.

6) Определение ресурсов (людей, оборудования, материалов) проекта - определение общего количества ресурсов всех видов, которые могут быть использованы на работах проекта (ресурсов организации) и их характеристик.

7) Назначение ресурсов - определение ресурсов, необходимых для выполнения отдельных операций проекта.

8) Оценка стоимости - определение составляющих стоимости операций проекта и оценка этих составляющих для каждой операции, ресурса и назначения.

9) Составление расписания выполнения работ - определение последовательности выполнения работ проекта, длительностей операций и распределения во времени потребностей в ресурсах и затрат с учетом наложенных ограничений и взаимосвязей.

10) Оценка бюджета - приложение оценок стоимости к отдельным компонентам проекта (этапам, фазам, срокам).

11) Разработка плана исполнения проекта - интеграция результатов остальных подпроцессов для составления полного документа.

12) Определение критериев успеха - разработка критериев оценки исполнения проекта.

Кроме перечисленных основных процессов планирования имеется ряд вспомогательных процессов, необходимость в использовании которых сильно зависит от природы конкретного проекта. Такие процессы включают в себя:

1) планирование качества - определение того, какие стандарты качества использовать в проекте, и того, как этих стандартов достичь;

2) планирование организации - определение, документирование и назначение ролей, ответственности и взаимоотношений отчетности в организации;

3) назначение персонала - назначение человеческих ресурсов на выполнение работ проекта;

4) планирование взаимодействия - определение потоков информации и способов взаимодействия, необходимых для участников проекта;

5) идентификация риска - определение и документирование событий

риска, которые могут повлиять на проект;

6) оценка риска - оценка вероятностей наступления событий риска, их характеристик и влияния на проект;

7) разработка методов реагирования - определение необходимых действий для предупреждения рисков и реакции на угрожающие события;

8) планирование поставок - определение того, что, как и когда должно быть поставлено;

9) подготовка условий - выработка требований к поставкам и определение потенциальных поставщиков.

Взаимосвязи между вспомогательными подпроцессами, как и само их наличие, в большой мере зависят от природы проекта.

Процессы исполнения и контроля. Под исполнением подразумеваются процессы реализации составленного плана. Исполнение проекта должно регулярно измеряться и анализироваться для того, чтобы выявить отклонения от намеченного плана и оценить их влияние на проект. Регулярное измерение параметров проекта и идентификация возникающих отклонений далее также относятся к процессам исполнения и именуются контролем исполнения. Контроль исполнения следует проводить по всем параметрам, входящим в план проекта.

Как и в планировании, процессы исполнения можно подразделить на основные и вспомогательные. К основным процессам исполнения можно отнести сам процесс исполнения плана проекта. Среди вспомогательных процессов можно отметить:

– учет исполнения - подготовку и распределение необходимой для участников проекта информации с требуемой периодичностью;

– подтверждение качества - регулярную оценку исполнения проекта с целью подтверждения соответствия принятым стандартам качества;

– подготовку предложений - сбор рекомендаций, отзывов, предложений, заявок и т.д.;

– выбор поставщиков - оценку предложений, выбор поставщиков и подрядчиков и заключение контрактов;

– контроль контрактов - контроль исполнения контрактов поставщиками и подрядчиками;

– развитие команды проекта - повышение квалификации участников команды проекта.

Процессы анализа. Процессы анализа включают анализ плана и анализ исполнения проекта.

Анализ плана означает определение того, удовлетворяет ли составленный план исполнения проекта предъявляемым к проекту требованиям и ожиданиям участников проекта. Он выражается в оценке показателей плана командой и другими участниками проекта.

На стадии планирования результатом анализа плана может быть принятие решения о необходимости изменения начальных условий и составления новой версии плана либо принятие разработанной версии в качестве базового плана проекта, который в дальнейшем служит основой для измерения исполнения. В дальнейшем изложении анализ плана не выделяется в качестве отдельной группы процессов, а включается в группу процессов планирования, делая эту группу по своей природе итеративной. Таким образом, под процессами анализа в дальнейшем понимаются процессы анализа исполнения.

Процессы анализа исполнения предназначены для оценки состояния и прогноза успешности исполнения проекта согласно критериям и ограничениям, определенным на стадии планирования. Для большинства проектов в число основных ограничений и критериев успеха входят цели, сроки, качество и стоимость работ проекта. При отрицательном прогнозе принимается решение о необходимости корректирующих воздействий, выбор которых осуществляется в процессах управления изменениями.

Процессы анализа также можно подразделить на основные и вспомогательные. К основным относятся те процессы анализа, которые непосредственно связаны с целями проекта и показателями, характеризующими успешность исполнения проекта:

– анализ сроков - определение соответствия фактических и прогнозных сроков исполнения операций проекта директивным или запланированным;

– анализ стоимости - определение соответствия фактической и прогнозной стоимости операций и фаз проекта директивным или запланированным;

– анализ качества - мониторинг результатов с целью их проверки на соответствие принятым стандартам качества и определение путей устранения причин нежелательных результатов исполнения качества проекта;

– подтверждение целей - процесс формальной приемки результатов проекта его участниками (инвесторами, потребителями и т.д.).

Вспомогательные процессы анализа связаны с анализом факторов, влияющих на цели и критерии успеха проекта. Эти процессы включают:

– оценку исполнения - анализ результатов работы и распределение

проектной информации с целью снабжения участников проекта данными о том, как используются ресурсы для достижения целей проекта;

– анализ ресурсов - определение соответствия фактической и прогнозной загрузки и производительности ресурсов запланированным, а также анализ соответствия фактического расхода материалов, машинного времени и т.д. плановым значениям.

В число процессов анализа не включены анализ взаимодействия с целью оптимизации процедур обработки проектной информации, анализ исполнения контрактов с целью своевременного внесения изменений и предотвращения споров и ряд других процессов, которые не носят регулярного характера, либо составляют часть включенных процессов. В результате анализа принимается решение о продолжении исполнения проекта по намеченному плану, либо принимается решение о необходимости применения корректирующих воздействий.

Процессы оперативного управления. Если исполнение проекта происходит по намеченному плану, то управление сводится к доведению к участникам плановых заданий и контролю над их реализацией.

В случае возникновения отклонений в процессе реализации требуется:

– найти оптимальные корректирующие воздействия;

– скорректировать план оставшихся работ;

– согласовать намеченные изменения со всеми участниками проекта.

Процессы оперативного управления часто называются управлением изменениями и инициируются процессами анализа. К основным процессам оперативного управления относятся:

– общее управление изменениями - определение, согласование, утверждение и принятие к исполнению корректирующих воздействий и координация изменений по всему проекту;

– управление ресурсами - внесение изменений в состав и назначение ресурсов на работы проекта;

– управление целями - корректировка целей проекта по результатам процессов анализа;

– управление качеством - разработка мероприятий по устранению причин неудовлетворительного исполнения.

Среди вспомогательных процессов управления выделяют:

– управление рисками - реагирование на события и изменение рисков в процессе исполнения проекта;

– управление контрактами - координация работ субподрядчиков, корректировка контрактов, разрешение конфликтов.

Процессы завершения. Завершение проекта заканчивается следующими процессами:

– закрытием контрактов - завершением и закрытием контрактов, включая разрешение всех возникших вопросов;

– административным завершением - подготовкой, сбором и распределением информации, необходимой для формального завершения проекта.

При реализации всех вышеперечисленных процессов управления образуется контур управления, в котором используются определенные методы и средства управления.

7.4. Методы планирования и управления проектами и ресурсами

С целью повышения эффективности проектирования ИС, т.е. обеспечения качества проекта в нужный срок с наименьшими стоимостными и трудовыми затратами, необходимо разработать систему управления проектом (СУП), которую можно рассматривать как систему управления операциями и получения аналитических и отчетных сводок [11].

Система управления проектами представляет собой организационно-технологический комплекс методических, технических, программных и информационных средств, направленный на поддержку и повышение эффективности процессов планирования и управления проектом.

Система управления проектами содержит набор функциональных средств, которые помогают менеджеру планировать работы, временные, ресурсные и стоимостные оценки выполнения комплекса работ, а затем в процессе выполнения отслеживать ход работ и корректировать план. Функциональные средства, реализующие взаимосвязанные методы, являются основой для информационных систем, которые моделируют комплекс работ и потребности в ресурсах. Эти методы используют оценки требуемых объемов работ и позволяют менеджеру регулировать выполнение работ по времени, стоимости, составу работ, качеству и организационной структуре исполнения.

Основные преимущества использования информационной системы для управления проектами включают:

– централизованное хранение информации по графику работ, ресурсам и стоимостям;

– возможности быстрого анализа влияния изменений в графике, ресурсном обеспечении и финансировании плана проекта;

– возможность распределенной поддержки и обновления данных в сетевом режиме;

– возможности автоматизированной генерации отчетов и графических диаграмм, разработки документации по проекту.

Процесс управления значительно облегчается, если СУП представить в виде модели, отражающей план разработки, в котором фиксируется весь ход событий для достижения конечной цели при заданных условиях. Составленная модель должна быть адекватна моделируемой системе. Информационная модель проекта, разработанная на начальной стадии планирования, подвергается в дальнейшем переработке в процессе его реализации. Таким образом, базовые методики планирования используются на протяжении всего жизненного цикла проекта.

Существует несколько способов формализованного представления выполняемой совокупности работ, применяемых для целей планирования и управления ими. Широкое распространение при построении моделей систем управления комплексом операций получили графические методы как наиболее универсальные и дающие обозримую информацию о ходе работ, к основным из которых относятся метод построения линейного графика Гантта и метод, основанный на использовании теории графов, - метод сетевого планирования и управления (СПУ).

Диаграмма Гантта, или циклограмма, - горизонтальная линейная диаграмма, на которой работы проекта представляются протяженными по времени отрезками, характеризующимися датами начала и окончания, задержками и, возможно, другими временными параметрами (рис. 7.4).

Получаемый график отличается статичностью и громоздкостью, по результатам отображения работ нельзя оперативно получать информацию о ресурсах, нельзя оперативно управлять, поэтому для целей планирования и управления он может быть применим при небольших объемах работ.

Существенными недостатками традиционных календарных графиков и циклограмм являются:

– неспособность в полной мере отражать взаимосвязи отдельных операций;

– недостаточная гибкость линейной модели;

– трудность ее корректировки при изменившихся условиях;

– ограниченные возможности прогнозирования дальнейшего хода работ, являющиеся факторами, снижающими эффективность процесса управления.

Код работы	Начало	Конец	Длительность в единицах времени	Временные периоды в единицах времени
				1 2 3 4 5 6 7 8 9
1	1.10	2.10	1	
2	2.10	5.10	3	
3	5.10	9.10	4	
4	6.10	9.10	3	

Рис. 7.4. Линейный график Гантта

Линейные модели, кроме того, не отражают той неопределенности, которая бывает присуща управлению проектами. Однако этот метод может быть использован при оптимизации распределения используемых ресурсов.

Сетевые модели свободны от этих недостатков, легко поддаются обработке на ЭВМ и позволяют более эффективно осуществлять планирование, координацию, контроль и управление процессом создания сложных систем.

Методика СПУ - развитая система планирования и управления, предусматривающая выявление и использование резервов времени и материальных ресурсов, дающая возможность прогнозирования и предупреждения возможных срывов в ходе выполнения программы. Она была разработана в конце 50-х годов в США, в 1956 г., М. Уолкером из фирмы «Дюпон» и Д. Келли из группы планирования капитального строительства фирмы «Ремингтон Рэнд». Они попытались использовать ЭВМ для составления планов-графиков крупных комплексов работ по модернизации заводов фирмы «Дюпон». В результате был создан рациональный и простой метод описания проекта с использованием ЭВМ, который первоначально был назван методом Уолкера-Келли, а позже получил название метода критического пути - МКП (или *CPM - Critical Path Method*).

Параллельно и независимо в США был создан метод анализа и оценки программ - *PERT* (*Program Evaluation and Review Technique*). Данный метод был разработан корпорацией «Локхид» и консалтинговой фирмой «Буз, Аллен энд Гамильтон» для реализации проекта разработки ракетной системы «Поларис», объединявшего около 3800 основных подрядчиков и состоявшего из 60 тыс. операций.

Рассмотрим ключевые определения и концепции используемых методов планирования, организации и контроля за проектом.

Работа в плане проекта представляет некоторую деятельность, необходимую для достижения конкретных результатов (конечных продуктов нижнего уровня). Таким образом, работа является основным элементом (дискретной компонентой) деятельности на самом нижнем уровне детализации, на выполнение которого требуются время и ресурсы и который может задержать начало выполнения других работ. Момент окончания работы означает факт получения конечного продукта (результата работы). Работа является базовым понятием и представляет основу для организации данных в системах управления проектами.

В понятие «работа» входит также ожидание, т.е. пассивный процесс, не требующий затрат труда и материальных ресурсов, но отнимающий время.

Под работой подразумевают и простую зависимость, т.е. логическую связь между двумя или большим числом операций, которую иногда называют холостой, или фиктивной, работой, так как она не требует никаких затрат времени, стоимости, труда.

Событие (веха) - результат выполнения работы или дата в ходе осуществления проекта. Событие используется для отображения состояния завершенности тех или иных работ. В контексте проекта менеджеры используют события или вехи для того, чтобы обозначить важные промежуточные результаты, которые должны быть достигнуты в процессе реализации проекта. Важным отличием событий от работ является то, что они не имеют длительности.

Связи предшествования (логические зависимости) отображают природу зависимостей между работами. Большинство связей в проектах относится к типу «конец-начало», когда последующая работа может начаться только по завершении предшествующей работы. Связи предшествования образуют структуру сети. Комплекс взаимосвязей между работами часто

называют логической структурой проекта, поскольку он определяет последовательность выполнения работ.

Сетевая диаграмма (сеть, сетевой график, *PERT*-диаграмма) - графическое отображение работ проекта и их взаимосвязей. В планировании и управлении проектами под термином «сеть» понимается полный комплекс работ и событий проекта с установленными между ними зависимостями. Сетевая диаграмма не является блок-схемой в том смысле, в котором это средство используется для моделирования деловых процессов. Принципиальным отличием ее от блок-схемы является то, что сетевая диаграмма моделирует только логические зависимости между элементарными работами. Она не отображает входы, процессы и выходы и не допускает повторяющихся циклов, или петель. Сетевые диаграммы отображают сетевую модель в графическом виде как множество вершин, соответствующих работам связанных линиями, представляющими взаимосвязи между работами. Этот граф называется сетью типа «вершина - работа», или диаграммой предшествования.

Существует другой тип сетевой диаграммы, называемый «сеть типа вершина - событие». При данном подходе работа представляется в виде линии между двумя событиями (узлами графа), которые, в свою очередь, отображают начало и конец данной работы. *PERT*-диаграммы являются примерами этого типа диаграмм.

Как правило, сетевые диаграммы этого типа используют для графического описания процесса проектирования ИС. Это позволяет применять для анализа сети хорошо отработанный арсенал математических методов проведения расчетов на сетевых графиках. Сетевые методы планирования и управления (СПУ) используются в проектах, которые легко декомпозируются на упорядоченную последовательность операций (работ). Можно выделить следующие особенности использования системы методов СПУ:

1) Системный подход к решению вопросов организации управления процессом создания новых систем.

2) Использование информационно-динамической модели особенного вида (сетевой модели комплекса операций) для логико-математического описания процесса создания системы и алгоритмизации расчетов параметров этого процесса (продолжительности, трудоемкости, стоимости).

3) Применение ЭВМ с целью обработки исходных и оперативных данных для расчета плановых показателей и получения

необходимых аналитических и отчетных сводок.

4) Комплексы работ, для которых применяются методы СПУ, могут иметь одноцелевой или многоцелевой характер.

В основе системы СПУ лежит построение сетевой модели плана - сетевого графика, на котором в определенном порядке наглядно показаны все операции по созданию сначала промежуточных результатов проектирования с определенной степенью готовности и под конец - полное завершение разработки. При построении сетевого графика необходимо для каждой работы (операции) знать работы (операции), непосредственно предшествующие данной и следующие за ней. Сеть выражает, таким образом, соотношения порядка, существующие на множестве работ, характеризующихся временем выполнения, и событий, которые характеризуются временем начала и временем окончания (рис. 7.5).

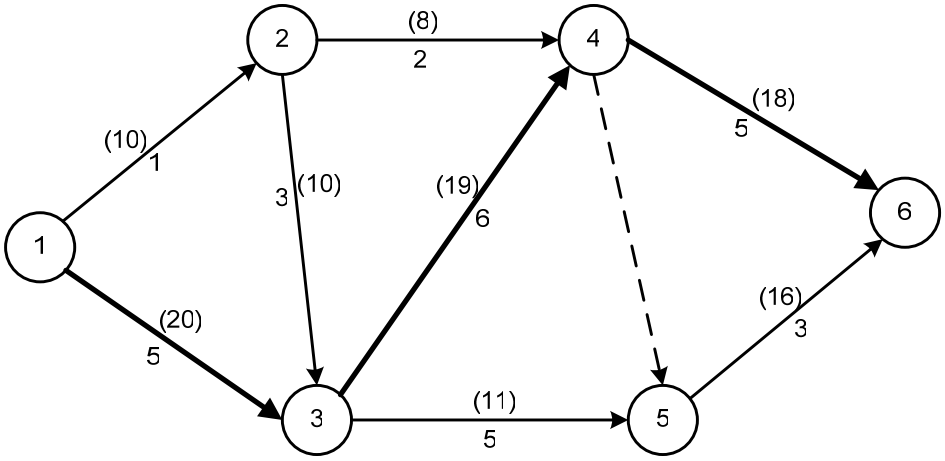


Рис.7.5. Схема сетевого графика

На сетевых графиках события изображаются кружками с порядковыми номерами, действительные работы, ожидания - сплошными стрелками, фиктивные работы или зависимости - пунктирными линиями - стрелками. Стрелки указывают последовательность выполнения операций. Взаимосвязь кружков и стрелок является графическими символами сетевой модели, которые должны строиться по определенным правилам:

1) Любая работа обозначается стрелкой, которая соединяет только два события и отражает процесс перехода от одного события к другому.

2) Событие, из которого стрелка выходит, называется начальным или предшествующим по отношению к дальнейшей работе. Событие, в которое стрелки входит, является конечным или последующим.

3) Начало стрелки показывает, с какого события данная работа начинается, а конец стрелки, - в каком событии она заканчивается.

4) Работы имеют временные оценки, которые проставляются на стрелках. Событие считается свершившимся тогда, когда будет закончена самая длительная из всех входящих в него работ.

5) Требуемые для выполнения работы размеры ресурсов указываются на стрелках в скобках.

Максимальный по продолжительности полный путь в сети называется критическим; работы, лежащие на этом пути, также называются критическими (на графике они отражаются двойными стрелками). Выявление критического пути позволяет установить работы (операции), определяющие ход выполнения проекта. Критические работы в ходе проектирования должны выполняться строго по графику. Именно длительность критического пути определяет наименьшую общую продолжительность работ по проекту в целом.

Метод критического пути позволяет рассчитать возможные календарные графики выполнения комплекса работ на основе описанной логической структуры сети и оценок продолжительности выполнения каждой работы, определить критический путь проекта. Длительность выполнения всего проекта в целом может быть сокращена за счет сокращения длительности работ, лежащих на критическом пути. Соответственно любая задержка выполнения работ критического пути повлечет увеличение длительности проекта. Концепция критического пути обеспечивает концентрацию внимания менеджера на критических работах.

Все критические работы являются потенциально «узкими» местами плана. Критических путей может быть несколько. Пути, продолжительность которых приближается к критическому пути, называются субкритическими. Остальные пути - некритические. Наличие критического пути позволяет использовать его в качестве основы для оптимизации плана. Работы, лежащие на некритическом пути, обладают некоторыми резервами времени, которые являются важными показателями работы сети. Однако основным достоинством метода критического пути является возможность манипулирования сроками выполнения задач, не лежащих на критическом пути.

Временной резерв, или запас времени, - это разность между самым ранним возможным сроком завершения работы и самым поздним допустимым временем ее выполнения. Управленческий смысл временного ре-

зерва заключается в том, что при необходимости урегулировать технологические, ресурсные или финансовые ограничения проекта он позволяет менеджеру задержать работу на это время без влияния на общую продолжительность проекта и продолжительность непосредственно связанных с ней задач.

Структура разбиения работ (СРР) - иерархическая структура последовательной декомпозиции задач проекта на подзадачи. Структура разбиения работ является изначальным инструментом для организации работ, обеспечивающим разделение общего объема работ по проекту в соответствии с порядком их выполнения в организации. На нижнем уровне детализации выделяются работы, соответствующие детализированным элементам деятельности, отображаемым в сетевой модели. СРР представляет иерархическую композицию, которая помогает разработчику для достижения следующих целей:

– структуризации работ на основные компоненты и подкомпоненты;

– обеспечения направленности деятельности на достижение всего комплекса целей;

– разработки системы ответственности за выполнение работ проекта;

– разработки системы отчетности и обобщения информации по проекту.

Структурная схема организации (ССО) имеет формат, подобный формату *СРР*. Каждому элементу нижнего уровня в *СРР* должны соответствовать один элемент или несколько элементов из ССО. Таким образом, ССО является средством определения ответственных за выполнение работ в сложных организациях и обеспечивает основу для разработки структуры системы отчетности.

Ресурсы - обеспечивающие компоненты деятельности, включающие исполнителей, энергию, материалы, машинное время, оборудование и т.д. Соответственно с каждой работой можно связать функцию потребности в ресурсах.

Методики назначения и выравнивания ресурсов позволяют менеджеру проанализировать сетевой план, построенный с помощью метода критического пути, с тем, чтобы обеспечить доступность и использование определенных ресурсов на протяжении всего времени выполнения проекта.

Назначение ресурсов состоит в определении потребности каждой работы в различных типах ресурсов. Методики выравнивания ресурсов представляют собой, как правило, программно-реализованные эвристические алгоритмы планирования при ограниченных ресурсах. Эти средства помогают менеджеру создать реальное расписание проекта с учетом потребности проекта в ресурсах и фактически доступных в данный момент времени ресурсов.

Ресурсная гистограмма - гистограмма, отображающая потребности проекта в том или ином виде ресурсов в каждый момент времени.

Ресурсное календарное планирование - планирование сроков начала работ при ограниченных наличных ресурсах. Проверка ресурсной реализуемости календарного плана требует сопоставления функций наличия и потребности в ресурсах для проекта в целом. Сдвигая некритические работы вплоть до их поздних сроков начала (окончания), можно видоизменить ресурсный профиль, обеспечивая оптимальное использование ресурсов. Информация, полученная в результате ресурсного анализа проекта, помогает заострить внимание менеджера и членов команды на тех моментах работы, где эффективное управление ресурсами будет являться ключевым фактором успеха.

Анализ реализуемости проекта - понятие реализуемости, имеет ряд своих разновидностей: логическую реализуемость (учет логических ограничений на возможный порядок выполнения работ во времени); временной анализ (расчет и анализ временных характеристик работ: ранняя/поздняя дата начала/окончания работы, полный, свободный временной резерв и др.); физическую (ресурсную) реализуемость (учет ограниченности наличных или доступных ресурсов в каждый момент времени выполнения проекта); финансовую реализуемость (обеспечение положительного баланса денежных средств как особого вида ресурса).

Параметрами, по которым должно вестись управление проектом, являются следующие: время, стоимость, ресурсы, технико-экономические показатели (ТЭП). Время управления проектом учитывается всегда, остальные параметры - в необходимых случаях. В зависимости от различного сочетания параметров возникают и соответствующие разновидности системы управления комплексом работ по конечным ее параметрам и их сочетаниям, к которым относятся:

- время;
- время - стоимость;

– время - ресурсы;
– время - стоимость - ресурсы;
– время - ТЭП;
– время - стоимость - ТЭП;
– время - ресурсы - ТЭП;
– время - стоимость - ресурсы - ТЭП.

Чаще всего используются системы с параметром «время». Возможность учета в системе всех видов ресурсов, к которым в первую очередь следует отнести рабочую силу, оборудование и денежные ресурсы, значительно расширяет сферу планирования и управления системой.

Можно отметить, что применение методологии СПУ дает ряд преимуществ в организации управления проектами, поскольку позволяет:

– четко отобразить объем и структуру решаемой задачи, выявить с достаточной степенью детализации работы, определить события, свершение которых необходимо для достижения заданной цели;

– выявить и всесторонне проанализировать взаимосвязь между работами, так как в самом принципе построения сетевой модели заложено точное отражение всех зависимостей между работами;

– разработать обоснованный план выполнения комплекса работ по созданию новой системы;

– более эффективно использовать ресурсы, так как анализ сетевой модели и выявление критических работ и резервов времени на «некритических» работах помогают руководителям определить возможности перераспределения ресурсов с целью ускорения выполнения критических работ и, следовательно, сократить сроки завершения разработки в целом;

– заранее анализировать результаты осуществления различных вариантов плана на ЭВМ;

– быстро обработать с помощью ЭВМ большие объемы данных и обеспечить руководство своевременной и исчерпывающей информацией о фактическом состоянии работ, облегчающих принятие обоснованных решений;

– осуществить обоснованное прогнозирование проектных работ и сконцентрировать внимание руководителей на их выполнении, что помогает руководителям заранее выявить «узкие» места и своевременно принять меры по их устранению;

– систематически корректировать оперативные планы работ в соответствии с фактическим состоянием разработки;

– накапливать в удобной форме систематизированную статистику по продолжительности, трудоемкости и стоимости выполнения типовых работ с целью разработки в последующем справочно-нормативных материалов для планирования и контроля.

7.5. Технология применения метода СПУ для разработки проекта информационной системы

Перед началом разработки проекта составляется организационный план проведения работ. Он состоит из трех разделов:

1) Исходный план - план-график выполнения работ проекта, содержащий исходные сведения об основных временных и стоимостных параметрах работ, который принят к исполнению. В исходном плане обычно фиксируются объемы работ, плановые даты начала и окончания задач проекта, длительности задач, расчетные стоимости задач. Он составляется в виде сетевого графика.

2) План материально-технической базы проектирования, в котором отражены вопросы, связанные с обеспечением проектировщиков необходимым инструментарием: бумагой, бланками документов, рабочим местом на объекте, возможностью использования комплекса существующих способов обработки для обеспечения предполагаемого способа обработки, обеспечением машинными носителями.

3) Квалификационный план разработчиков, где указывается форма проведения проектировочных работ. При бригадной форме устанавливаются перечень исполнителей по проектированию, должность, оклад, формы и методы контроля над работой проектировщиков, возможности взаимной увязки материалов различных бригад.

Процессы планирования и управления проектами с применением СПУ охватывают три этапа:

1) разработка первоначального исходного сетевого плана;

2) оптимизация плана и приведение его в соответствие с ограничениями;

3) оперативное управление и систематический контроль за ходом разработок.

Этап 1 начинается с определения состава работ по стадиям и этапам процесса проектирования. Центральной задачей является выбор единицы работ. Известно несколько способов решения этой задачи. В качестве еди-

ницы проектных работ используется понятие технологической операции проектирования.

После завершения подготовительных работ, располагая исходными данными, строят первый вариант сетевого графика (приближенный) на основе логического расчленения процесса проектирования. Затем проводится дифференцирование процесса проектирования двумя способами:

– по отдельным наименованиям работ - предметный способ;

– по временным этапам, последовательности выполняемых работ.

Работа завершается составлением начального план-графика. Каждая работа должна завершаться результатом. Из анализа план-графика выясняется возможность начала очередной работы после частичного завершения предыдущих работ.

Каждый исполнитель (отдел) формирует первичную сеть, в результате возникает необходимость объединения первичных сетей - сшивание детальных графиков.

При разработке первого варианта графика одновременно предусматривается закрепление за работами отдельных исполнителей.

В завершение перового этапа разрабатывается на ЭВМ сетевой график, и определяются основные показатели:

– продолжительность каждой работы $t_{i\text{-}j}$;

– время совершения события раннее $t_{p(i)}$, позднее $t_{n(i)}$;

– время начала работы раннее $t_{рн(i\text{-}j)}$, позднее $t_{пн(i\text{-}j)}$;

– время окончания работы раннее $t_{ро(i\text{-}j)}$, позднее $t_{по(i\text{-}j)}$;

– полный резерв времени $R_{(i\text{-}j)}$;

– частичный резерв времени работы $r_{(i\text{-}j)}$;

– частичный резерв времени события $r_{(i)}$.

Как правило, сетевой график состоит из набора сетевых графиков, соответствующих отдельным этапам проектирования.

В качестве показателя времени, затрачиваемого на выполнение работ, выбирается неделя, декада, месяц.

Для работ, имеющих нормативную продолжительность, расчеты проводятся по формуле:

$$t_{(i-j)} = \frac{Q_{ij}}{A_{(i-j)} f},$$

где Q_{ij} - трудоемкость работы (чел/дней); $A_{(i\text{-}j)}$ - количество исполнителей;

f = 0,85 - коэффициент перевода рабочих дней в календарные.

Сети с однозначными временными оценками называются детерминированными.

Для сетей, у которых обоснование нормы продолжительности отсутствует, временные оценки устанавливаются в условиях полной неопределенности. Для оценки продолжительности пользуются вероятностными методами, а сети называются стохастическими. Принимается закон распределения времени выполнения работ гауссовским и с вероятностью p=0,99 определяют доверительные границы - минимальные и максимальные продолжительности выполняемых работ.

Любая последовательность работ в сетевом графике, в котором конечное событие предшествующей работы совпадает с начальным событием последующей, называется путем. Длина пути рассчитывается так:

$$T(L) = \sum t(i - j).$$

В сетевом графике получается несколько путей от начального события к конечному. Путь наибольшей длины называется критическим путем на выполнение всех проектных работ:

$$T_{\text{кр}} = max\ \{T(L)\}.$$

После составления и расчета сетевого графика планируется использование разнообразных ресурсов (трудовых, материальных, необходимое оборудование, стоимость) по каждой работе.

Требуется, исходя из количества выделенных ресурсов на проектирование, определить такие календарные сроки выполнения операций проектирования заданной технологии, которые минимизируют общее время разработки проекта.

Этап 2. Происходит корректировка исходного сетевого графика, его оптимизация по следующим критериям:

- время $T_{\text{кр}} \leq T_{\text{дир}}$ (директивное время);
- затраты материальных ресурсов;
- затраты денежных ресурсов;
- ТЭП.

Первоначально сеть корректируется по критерию «время» без учета ограничений.

При возникновении дополнительного резерва времени $R_{доп} = t_{дир} - t_{кр}$ его используют для увеличения продолжительности отдельных критических работ при последующей оптимизации.

При $T_{кр} > T_{дир}$ сеть пересматривается с целью уплотнения (изменения топологии сети, расчленение работ и совмещение во времени и т.п.). Если испробованные меры не позволяют сократить $T_{кр}$, то ставиться перед руководством вопрос об увеличении $T_{дир}$. После расчета временных параметров проводится анализ установления соответствия параметров сетевого графика заданным ограничениям использования ресурсов. Оптимизация плана состоит в снятии ресурсов и переброске их на критический путь с целью сокращения времени на критическом пути. Процесс коррекции носит итеративный характер.

Помимо основных работ в сетевой график должны включаться процедуры контроля над проектной деятельностью, которые важны для СУП.

Этап 3. Сетевой график удобно применять в процессе оперативного управления проектированием. Руководители проектов отвечают за сроки, расходы и качество результата.

Контроль над соблюдением сроков и затрат обеспечивается функцией учета. Контроль качества проекта включает:

– контроль соответствия функций разработанной системы требований к системе;

– контроль качества проекта по принятым качественным критериям (НТУ проекта).

Процедура контроля проекта завершается сопоставлением полученных значений параметров системы с требуемыми значениями. В результате анализа принимается решение либо о корректировании проекта системы, либо пересматриваются целевые установки системы.

7.6. Выбор системы для управления проектами

Важной составной частью системы управления проектами являются инструментальные средства, с помощью которых реализуются методы СПУ и МКП, представляющие собой совокупность программных средств, направленных на поддержку и повышение эффективности процессов планирования и управления проектом. Выбор типов программного обеспечения по управлению проектами в организации осуществляют в следующей последовательности:

– анализ требований пользователей;

– анализ рынка;
– выбор программного обеспечения.

В системе управления проектами можно выделить три уровня управления проектами, соответствующих определенным категориям пользователей ПО, выполняющих специфические функции:

1) Уровень высшего руководства, на котором происходит определение целей и задач предприятия, принимается решение о финансировании, оценивается приоритетность проектов.

2) Стратегический уровень, состоящий из профессионалов по управлению проектами, занимающихся планированием и контролем корпоративных проектов. Как правило, этот уровень представляется небольшим количеством людей, основная обязанность которых именно управление проектами и которые в своей работе опираются на программное обеспечение по управлению проектами. Роль подобных профессионалов является ключевой в организации, и они работают как группа поддержки по управлению проектами.

3) Уровень операций, для которого работа с программным обеспечением по управлению проектами вторична. Это ответственные за проекты на местах, менеджеры проектов, руководители групп. На уровне операций требуется инструмент по управлению и контролю над проектом, но на небольшие отрезки времени.

В таблице 7.1 представлены требования к программным средствам планирования и управления проектными работами со стороны специалистов трех вышеперечисленных уровней.

К числу основных факторов, предопределяющих выбор инструментального средства для управления проектами, можно отнести следующие:

– тип задач, для которых потребуется система управления проектами;
– характер деятельности организации с точки зрения возможности и целесообразности применения проектной формы планирования и управления;
– вид деятельности, которая может планироваться в виде проектов;
– уровень детальности, до которого необходимо планировать и контролировать проекты.

Для поддержки различных управленческих задач используются различные программные средства.

1. Для укрупненного описания и анализа проекта на прединвестиционной стадии в большей степени подходит специализированное ПО анализа проектов, которое позволяет выполнить оценки основных показателей рентабельности проекта в целом и обосновать эффективность капиталовложений. Примером системы для анализа проектов является хорошо известная на российском рынке программа *Project Expert* фирмы *PRO-INVEST- Consulting*.

Необходимо отметить, что для описания плана инвестиций в *Project Expert* используются традиционные подходы сетевого планирования, предполагающие разбиение проекта на комплекс взаимозависимых задач и описание требуемых для их выполнения ресурсов. В *Project Expert* реализованы *Gantt-* и *PERT*-диаграммы.

Таблица 7.1 Требования к программным средствам планирования и управления проектными работами

Уровень высшего руководства	Стратегический уровень	Уровень операций
1. Легкость в применении	1. Средства временного, ресурсного, стоимостного планирования, анализа рисков	1 .Простота использования
2. Возможность получать демонстрационные отчеты	2. Возможность интеграции с другими приложениями	2. Легкость изучения
3. Мощные возможности обобщения сведений	3. Средства для свертывания данных по проекту (предоставление отчетов руководству) и углублению для планирования на более детальном уровне	3."Прозрачность" процедур ввода данных
4. Средства для интеграции с данными из других программных приложений	4. Средства для контроля за реализацией проекта	4. Наглядность
5. Процедуры для планирования сверху-вниз	5. Гибкость при настройке выходных форм отчетности	

2. Однако если управление проектами в организации не завершается обоснованием инвестиций и существует потребность в контроле над ходом реализации проекта, то необходимо переходить к использованию ПО управления проектами. Следует отметить, что *Project Expert* имеет возможность обмена данными с пакетами управления проектами *MS Project* и *Time Line.*

Если принципиальное решение об использовании системы для управления проектами (УП) принято, то для выбора пакета необходимо ответить на вопросы, связанные с выяснением состава функций планирования и управления, которые требуется реализовать:

– только планирование или планирование и контроль над ходом проекта;

– планирование и контроль лишь сроков выполнения работ;

– планирование и контроль финансовых вложений без детального планирования использования ресурсов;

– детальное планирование использования ресурсов;

– многопроектное управление.

Далее следует определить также требования к следующим компонентам проекта:

– к размерности проектов и детальности планирования;

– организационной структуре управления и отчетности;

– сколько проектов будет вестись одновременно и будут ли они взаимозависимыми;

– каково примерное количество задач в одном проекте;

– сколько видов ресурсов будет задействовано в одном проекте;

– как будут разделяться ресурсы между проектами.

Выбираемое средство управления проектами должно включать следующие базовые функциональные возможности.

1. Средства описания комплекса работ проекта, связей между работами и их временных характеристик.

1.1. Средства описания и типы планирования:

– выполнить работу «как можно раньше»;

– выполнить работу «как можно позже»;

– работы с фиксированной датой начала-окончания;

– возможность привязки длительностей задач к объему назначенных ресурсов;

– вычисляемые резервы времени (полный, свободный) и т.д.

1.2. Средства установки логических связей между задачами.

1.3. Многоуровневое представление проекта.

1.4. Поддержка календаря проекта, поддержка календарей ресурсов.

2. Средства поддержки информации о ресурсах и затратах по проекту и назначения ресурсов и затрат по отдельным работам над проектом.

2.1. Ведение списка наличных ресурсов, возможность задания нормального и максимального объемов ресурса.

2.2. Поддержка ресурсов с фиксированной стоимостью и ресурсов, стоимость которых зависит от длительности их использования.

2.3. Расчет требуемых объемов ресурсов.

2.4. Ресурсное планирование (выделение перегруженных ресурсов и использующих их задач, автоматическое/командное выравнивание профилей загрузки ресурсов с учетом ограничений по времени или с учетом ограничения на ресурс, с учетом приоритетов задач).

3. Средства контроля над ходом выполнения проекта.

3.1. Средства отслеживания состояния задач проекта (фиксация плана расписания проекта, средства ввода фактических показателей состояния задач (процент завершения)).

3.2. Средства контроля над фактическим использованием ресурсов (бюджетное количество и стоимость ресурса, фактическое количество и стоимость ресурса, количество и стоимость ресурсов, требуемых для завершения работы).

4. Графические средства представления структуры проекта, средства создания различных отчетов по проекту.

4.1. Диаграмма Гантта (часто совмещенная с электронной таблицей и позволяющая отображать различную дополнительную информацию).

4.2. *PERT*-диаграмма (сетевая диаграмма).

4.3. Средства создания необходимых для планирования отчетов (отчет по состоянию выполнения расписания, отчеты по ресурсам и по назначению ресурсов, профиль ресурса, отчет по стоимости).

К числу самых известных систем управления проектами относятся системы *Microsoft Project, Timeline, Primavera, Artemis Views, Spider Project, Open Plan.*

Сравнительная характеристика некоторых из них представлена в таблице 7.2

Таблица 7.2. Сравнительная характеристика ПО СУП

Фирма-изготовитель	*Microsoft*	*TimeLine Solutions*	*Primavera Systems, Inc.*	*Artemis International*	*Open Plan*
Версии программного продукта	*Project 4.1, Project- 98*	*TimeLine 1.0, TimeLine 6.5* для *Windows*	*SureTrak, Primavera Project Planner (P3), Monte Carlo for Primavera*	*Artemis Views*	*Open Plan Professional*
Мощность проекта	Малые, средние и крупные проекты	Средние проекты (10.000 задач и 1000 видов ресурсов)	Средние и крупные проекты	Крупные инженерные проекты	Крупные проекты масштаба корпорации
Использование современных стандартов	*ODBC* и *OLE* 2.0,	*ODBC, OLE 2.0, DDE, Symantec Basic*	*ODBC*	*ODBC*	*ODBC*
Групповая работа с проектами	*Microsoft Mail u Microsoft Exchange,* корпоративный *Web*-сайт	ЭП	ЭП	ЭП	ЭП, корпоративный *Web*-сайт
Средства разграничения доступа к файлам проектов	Папки *Exchange*	+	+	+	+

Фирма-изготовитель	*Microsoft*	*TimeLine Solutions*	*Primavera Systems, Inc.*	*Artemis International*	*Open Plan*
Помощь пользователю	*Create Your, First Project* и *Cue Cards*	Функция Инструктор *Guide Line* и *Guide Line Maker*	+	+	+
Средства создания отчетов	Диаграммы Гантта, *PERT*-диаграммы *Report Gallery*	Диаграммы Гантта, *PERT*-диаграммы, календарный график, *Cristal Reports* 4	Диаграммы Гантта, *PER*T-диаграммы, *Quest*	Диаграммы Гантта, *PER*T-диаграммы	Диаграммы Гантта, *PERT*-диаграммы, таблицы, ресурсные и стоимостные гистограммы
Концепции многопроектного планирования	*Project*-98	*TimeLine* 6.5	Возможность определения иерархии и права доступа к мастер-проекту и подпроектам	*Project View*	+

Фирма-изготовитель	*Microsoft*	*TimeLine Solutions*	*Primavera Systems, Inc.*	*Artemis International*	*Open Plan*
Импорт/экспорт данных в форматах	*ASC*II, *CSV, Excel*, *MS Access,Lotus* 1-2-3, *dBASE* и *Fox-Pro,* средства записи макрокоманд, *Visual Basic*5.0 *for Applications*	*ASC*II, *CSV, Lotus* 1-2-3, *dBASE*	*P3* и *MS Project*	СУБД *Oracle, SQLBase, SQL Server, Sybase*	*Open Plan* в форматах *Oracle*, *SQL Server, Sybase, xBase, Microsoft Access, Visual Basic*
Русификация	+	+	-	-	+
Автоматизируемые функции	1. Планирование 2. Контроль исполнения 3. Анализ 4. Управление изменениями 5. Завершение	1. Планирование 2. Контроль исполнения 3. Анализ 4. Управление изменениями 5. Завершение	1. Планирование 2. Контроль исполнения 3. Анализ 4. Управление изменениями 5. Завершение	1. *Resource View* 2. *Track View* 3. *Cost View*	1. Планирование при ограниченном времени 2. Управление всеми видами ресурсов 3. Анализ затрат 4. Анализ рисков 5 Мультипроектный анализ и структуризация

ОТЛАДКА, ИСПЫТАНИЕ И ОБСЛУЖИВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ

8.1. Проверка работоспособности и правильности функционирования ИС

Проверка информационной системы выполняется как в режиме отладки, так и в процессе эксплуатации. Проверка характеризуется достоверностью выводов о техническом состоянии аппаратуры и программ. Она характеризуется затратами на проведение проверок, временем проверок и расходами ресурсов измерительной аппаратуры [10].

В процессе проверки проводится декомпозиция ИС, а также используются имитаторы. При разработке способов проверки и критериев работоспособности используются два подхода: функциональный и аппаратный.

Функциональный способ выполняет проверку каждой функции ИС, преимущественно базовых и основных функций. По правильности выполнения отдельных функций судят о работоспособности системы в целом.

При использовании аппаратного подхода оговаривают класс неисправностей, подлежащих проверке. Недостатком этого метода является ограниченный класс рассматриваемых неисправностей, что проводит к уменьшению полноты проверки.

Функциональный подход ориентирован на исследование ИС как устройства переработки информации. Аппаратный подход ориентирован на исследование аппаратных средств.

Проверка работоспособности ИС осуществляется с помощью тестовых программ. Тестовые программы можно классифицировать на следующие виды:

1) наладочные;
2) контролирующие;
3) диагностические.

Контролирующие тестовые программы служат для проверки ИС на завершающей стадии комплексной наладки, а также в процессе эксплуатации систем. Эти программы строятся с соблюдением принципа расширяемости областей, в соответствии с которым предполагается исправность части оборудования и программного обеспечения, используемых в тестах.

Диагностические тестовые программы служат для поиска неисправностей в ИС и обеспечивают автоматизацию этого процесса.

Отладка комплексов программ проводиться с целью установления факта работоспособности программного обеспечения, соответствия требованиям технического задания, а также для выявления и устранения ошибок. Эти работы занимают до 35% всех затрат на создание программного обеспечения ИС.

Отладку проводят с использованием следующих четырех методов:

1) Структурный контроль и выявление ошибок методом проверки удовлетворения программ системе формализованных правил построения программ. Метод в основном применяется на нижних уровнях иерархии программ. Позволяет выявить ошибки записи/чтения переменных, некоторые виды зацикливания, наличие тупиковых и лишних участков и другие нарушения правил построения структуры программ.

2) Отладка по частным детерминированным реализациям тестов. Этот метод предусматривает получение промежуточных и результирующих величин при функционировании программы с некоторым детерминированным набором исходных данных и сравнения значений с эталонами. Статическая отладка включает в себя проверку функционирования и проверку точности программ

3) Отладка во всем диапазоне изменения исходных переменных с проверкой функционирования системы в динамике.

4) Отладка по статистическим характеристикам реализации тестов. Результаты проверяются по их законам распределения и параметрам этих законов. В процессе отладки проверяются динамические характеристики программ по времени их реализации, последовательности выполнения, использованию памяти и другим показателям. Эта отладка обеспечивает достаточно полное обнаружение ошибок: принципиальных, алгоритмических и системных.

Основным принципом отладки является последовательное иерархическое усложнение задач, увеличение объема одновременно функционирующих программ, объема преобразуемой информации.

По уровню сложности, степени связи с реальным масштабом времени и функционированием реальной аппаратуры процесс отладки комплекса программ для ИС делится на четыре этапа:

1) автономная статическая и автономная динамическая отладка функционирующих задач;

2) статическая и динамическая отладка всего комплекса программ без подключения к объекту управления;

3) отладка комплекса программ с реальными источниками и приемниками информации;

4) статистические испытания программ.

Автономная отладка программ предусматривает проверку работоспособности и коррекцию программ. Здесь проверяется схема принятия логических решений, правильность преобразования квазинепрерывных величин, уточняется состав используемой и выдаваемой информации. По каждому маршруту прохождения программ рассматривается время реализации и вероятность данного маршрута. При статической отладке проверяется реакция программ на отдельные значения входных переменных. Результаты работы сравниваются с известным результатом, оценивается точность решения задачи. При динамической отладки весь диапазон изменения входных переменных задается имитаторами, а выходные данные передаются имитаторам приемников информации.

Комплексная отладка систем реального времени ставит целью проверку совместной работы задач под управлением программы диспетчера. При этом источники и приемники информации имитируются. Статическая комплексная отладка характеризуется двумя особенностями:

– игнорированием реального времени включения программ диспетчером;

– слабым учетом реального временного взаимодействия с другими программами и подсистемами, включаемыми диспетчером в другие моменты времени.

Динамическая комплексная отладка проверяет взаимодействие групп программ с учетом изменения реального времени. Внешние абоненты имитируются специальными программами или аппаратными имитаторами. Основными задачами динамической отладки являются:

1) Проверка начального режима включения комплекса программ.

2) Проверка взаимодействия программ при наиболее опасных и трудных для функционирования исходных данных.

3) Проверка обеспечения устойчивости функционирования комплекса программ при искажении входной информации, при сбоях и отказах в устройствах ИС.

4) Проверка эффективности комплекса программ функционального контроля ИС.

При динамической комплексной отладке проверяется динамика заполнения буферных накопителей, динамика подключения программ, выработка сигнала подключения периодических программ, функционирование служебных и управляющих программ. Последовательное подключение и наращивание объема функциональных групп программ проводится по одной из двух стратегий:

1) Сначала подключаются к диспетчеру функциональные группы программ, почти независящие от информации, вырабатываемой другими алгоритмами. Далее последовательно наращиваются алгоритмы, использующие информацию от других функционирующих алгоритмов.

2) Функциональные программы подключаются в естественном порядке вызова в зависимости от поступающих сообщений или формируемых заявок. При этом с помощью имитаторов вводиться вся необходимая внешняя информация.

Сложным является контроль процесса обмена информацией между функциональными задачами. Сложность вызывается вследствие случайного потока поступающей информации от внешних абонентов, возможности искажения и потери информации, влияния предыстории функционирования алгоритмов.

Комплексная отладка с реальным объектом служит для контроля завершения отладки ИС с учетом особенностей характеристик реального объекта управления. На этом этапе окончательно устанавливаются характеристики процессов управления, отлаживаются системы функционального контроля ИС, алгоритмы контроля систем передачи данных, проверяется помехоустойчивость и надежность функционирования комплекса программ.

Статистические испытания программ представляют наиболее продолжительный этап отладки. Испытания проводятся с помощью статистических тестов. Проверке подлежат статистические характеристики результатов вычислений. Случайные изменения входных переменных расширяют диапазон возможных состояний входных переменных, в соответствии с чем, могут изменяться логические маршруты обработки информации. Это позволяет расширить количество проверяемых логических маршрутов по сравнению с теми, которые удалось проверить по детерминированным тестам.

8.2. Организация эксплуатации информационных систем

Эффективность использования информационной системы во многом зависит от правильной организации эксплуатации и обслуживания системы. Сложность обслуживания вызывается сложностью обслуживания самой ИС и большим объемом технической документации.

На стадии проектирования должны решаться такие вопросы как:

1) Контроль технического состояния технических средств информационной системы.
2) Восстановление работоспособности ИС.
3) Обеспечение заданных условий эксплуатации.
4) Повышение квалификации обслуживающего персонала.

1. Планирование проверок технического состояния систем

В информационных системах с аппаратным контролем могут появиться ситуации, когда возникающие неисправности не могут обнаруживаться техническими средствами. В связи с эти возникает необходимость использования программного контроля. Контролирующие программы могут включаться в случайные моменты времени, либо в заранее определенные моменты времени, что используется чаще в системах (рис. 8.1).

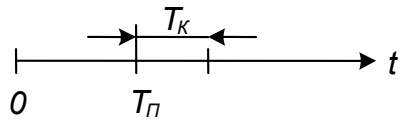


Рис.8.1. Диаграмма периодичности контроля ИС.

Обозначения: $T_П$ - периодичность запуска контролирующей программы; $T_К$ - длительность процесса контроля.

Предположим, что поток отказов в аппаратуре имеет случайный характер и описывается экспоненциальным законом плотности распределения вероятности:

$$P(t) = \exp(-\lambda \Delta t),$$

где λ - интенсивность отказов аппаратуры; Δt – продолжительность времени работы.

В момент времени Δt на интервале времени $0 \le \Delta t \le (T_П + T_К)$ могут возникнуть отказы в системе с вероятностью:

$$Q(\Delta t) = 1 - P(\Delta t) = 1\text{-}\exp(-\lambda \Delta t) \qquad (8.1)$$

Из соотношения (8.1) следует, что максимальную достоверность имеют задачи, решаемые непосредственно после окончания этапа проверки, а минимальную - задачи, решаемые непосредственно перед этапом проверки.

Так как проверка циклическая, то $Q(\Delta t)$ представляет собой периодическую функцию с периодом $T_П$. Значение $Q(\Delta t)$ меняется в пределах:

$$0 < Q(\Delta\, t) < Q(T_П).$$

Уменьшение $T_П$ увеличивает достоверность решаемых задач, но при этом возрастает относительное время на организацию контроля, что эквивалентно уменьшению производительности ИС. Поэтому возникает задача выбора рациональной периодичности проведения проверок.

ИС может находиться в следующих состояниях, которые будем характеризовать событиями:

1) A - ИС работоспособна;
2) $\overline{A}$ - неработоспособное состояние системы;
3) B - ИС используется в задачах принятия решений;
4) $\overline{B}$ - система используется не по назначению в некоторый момент времени.

Каждое событие характеризуется потерями:

1) $F(A,\ B) = 0$, система работоспособна и используется по назначению, следовательно, потери в этой системе равны нулю;
2) $F(A,\ \overline{B}) =$ A1, система работоспособна и проводится ее проверка. A1 - потери на проверку работоспособности системы;
3) $F(\overline{A},\ \overline{B}) =$ A2, ремонт неисправной системы. Потери A2 - затраты на восстановление системы;
4) 4) $F(\overline{A},\ B) =$ A3, использование неработоспособной системы для принятия решений. A3 - потери от ошибочных решений.

Принимаем, что в системе выполняется соотношение:

$$0 < \text{A1} \le \text{A2} \le \text{A3}.$$

Пусть в момент $t = 0$ система исправна и начинает работать, в момент времени $T_П$ работа системы прерывается на контроль, который продолжается в течение времени $T_К$.

Данный цикл ($T_П + T_К$) возникает с вероятностью исправной работы системы:

$$P = \exp(-\lambda T_П).$$

В этом случае потери за один цикл будут равны:

$$(T_П \cdot 0 + T_К A1) = T_К A1 . \qquad (8.2)$$

В случае выявления контролем неработоспособности системы, как это показано на рисунке 8.2, начинаются ремонтные работы, которые продолжаются в течение времени $T_Р$.

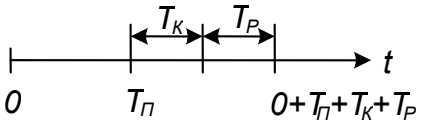


Рис.8.2. Обнаружена неисправность системы

При этом начало времени очередного цикла работы перемещается в точку

$$0 + (T_П + T_К + T_Р).$$

Потери в этом случае будут равны:

$$((T_П + T_К)A3 + T_РA2)$$

с вероятностью $Q = (1-P) = 1-\exp(-\lambda T)$. (8.3)

Выражения (8.2) и (8.3) носят случайный характер.

Общие потери $C(L)$ за большое число циклов работы системы L будут определяться выражением:

$$C(L) = L(T_КA1\exp(-\lambda T_П) + ((T_П + T_К) A3 + T_РA2)(1-\exp(-\lambda T_П)).$$

Продолжительность времени работы ИС за L циклов равна:

$$t(L) = L\,((T_П + T_К)\exp(-\lambda T_П) + (T_П + T_К + T_Р)\,(1-\exp(-\lambda T_П)).$$

Средние потери в единицу времени $\overline{C}$ будут равны:

$$\overline{C} = \frac{C(L)}{t(L)}. \qquad (8.4)$$

Упростим формулу (8.4) заменой $\exp(-\lambda T_П)$ первыми двумя членами разложения экспоненты в ряд Тейлора:

$$\exp(-\lambda T_П) = 1-\lambda\, T_П \ .$$

Тогда после замены получаем:

$$\overline{C} = \frac{\lambda(A3T_П^2 + T_П(A3T_К + A2T_Р - A1T_К)) + A1T_К}{T_П(1 + T_Р\lambda) + T_К}. \qquad (8.5)$$

Учитывая, что $T_Р\lambda << 1$ и, что $T_К \le T_П$ формула (8.5) упрощается и принимает вид:

$$\overline{C} \cong A1T_К(\frac{1}{T_П} + \lambda) + A2T_Р\lambda + A3T_П\lambda \ . \qquad (8.6)$$

В этой формуле первое слагаемое учитывает затраты системного времени на контроль работоспособности. Второе слагаемое учитывает затраты на проведение ремонтных работ. Третье слагаемое отражает затраты связанные с использованием неработоспособной системы.

Воспользуемся аналитическими методами анализа этой функции. Находим экстремум функции (8.6):

$$\frac{d\overline{C}}{dT_П} = 0\,, \qquad T_П = \sqrt{\frac{A1 \cdot T_К}{A3 \cdot \lambda}}\,. \qquad (8.7)$$

Для оптимальной периодичности проведения контроля средние потери будут минимальными и равными:

$$\overline{C}_{МИН} \cong 2\sqrt{A1A3T_К\lambda} - A1T_К\lambda + A2T_Р\lambda \ . \qquad (8.8)$$

На практике сопоставить функции потерь А1 и А3 не всегда удается. Поэтому периодичность проверок выбирается из других соображений, а именно:

1) по обеспечиваемой достоверности работоспособности системы;

2) по затратам системного времени на проверки.

Минимальная по времени достоверность работоспособности системы I при организации периодических проверок $T_П = T_К + T_П$ равна:

$$I = P(t) = \exp(-\lambda(T_К+T_П)) \cong 1-\lambda\,(T_К+T_П).$$

Тогда максимальная недостоверность работоспособности системы будет:

$$H = 1 - I \cong \lambda\ (T_К + T_П). \qquad (8.9)$$

Введем понятие относительных затрат системного времени на проведение поверок:

$$\gamma = \frac{T_К}{T_П + T_К}. \qquad (8.10)$$

С учетом выражений (8.9 и 8.10) получим формулу средних потерь в системе:

$$C = A1\frac{\gamma}{1-\gamma} + A3H\ . \qquad (8.11)$$

Задаваясь ограничениями $H \le H^*$ и $\gamma \le \gamma^*$ можно записать условие, с учетом (8.9) и (8.10), для выбора периодичности поверок:

$$\frac{T_К}{\gamma^*} \le T_П + T_К \le \frac{H^*}{\lambda} \quad \text{или} \qquad (8.12)$$

$$\gamma^* \cdot H^* \ge T_К \cdot \lambda\ . \qquad (8.13)$$

Последнее неравенство (8.13) представляет условие непротиворечивости для ИС с заданными средствами тестового контроля.

При задании одного из ограничений γ^* или H^* для обеспечения другому показателю минимального значения, период проверок $T_П$ выбирается из условия равенства в соотношении (8.12) соответствующей границе.

8.3. Обеспечение информационной системы запасными устройствами

В процессе эксплуатации ИС отказавшие устройства при проведении ремонта заменяются работоспособными из запасных инструментов и приборов (ЗИП). ЗИП пополняется по заявкам служб эксплуатации. Комплект ЗИП рассчитывается по статистическим данным об эксплуатации ИС. Объем ЗИП ограничен затратами ресурсов, габаритами и т.д. С другой стороны недопустимы простои ИС из-за отсутствия запасных элементов. Поэтому для эффективной эксплуатации ИС необходимо рассчитывать рациональный объем ЗИП [10].

Допустим, что ИС состоит из m числа устройств разного типа. Пусть схема соединения устройств последовательная. Обозначим: m_i – количество запасных устройств i-го типа; P – вероятность того, что за время эксплуатации от 0 до t_0 не будет недостатка в любом из m типов устройств и модулей; $Q(m_i)$ - вероятность того, что за время от 0 до t_0 произойдет m_i или менее отказов элементов i-го типа. Условимся, что отказ любого устройства приводит к отказу всей ИС. Тогда вероятность отказа ИС можно записать формулой:

$$Q = 1 - \prod_{i=1}^{m} Q(m_i) . \qquad (8.14)$$

С другой стороны, с учетом стоимости i-го устройства c_i, стоимость всего ЗИП будет равной:

$$S = \sum_{i=1}^{m} m_i c_i .$$

Для обеспечения непрерывной работы ИС выделяются средства в объеме С. Ограничения средств на приобретение ЗИП запишутся неравенством:

$$\sum_{i=1}^{m} m_i c_i \le C . \qquad (8.15)$$

Требуется так распределить элементы в составе ЗИП по типам, чтобы при выполнении условия (8.15) минимизировать вероятность отказа ИС на заданном временном интервале $[0, t_0]$:

$$\min Q(t_0) = \min\left(1 - \prod_{i=1}^{m} Q(m_i)\right) \qquad (8.16)$$

Сформулированная задача относится к типовой задаче оптимального резервирования, и она решается известными методами математического программирования.

Рассмотрим метод определения $Q(t_0)$. Допустим, что ЗИП при расходовании элементов не пополняется. Пусть ξ_0 , ξ_1 ,.. ξ_m,- время работы элементов ЗИП, представляет случайные величины.

Время от начала работы ИС, с учетом отказа последнего элемента из ЗИП m_i , будет равным:

$$S_i = \sum \xi_i .$$

Для экспоненциального закона распределения ξ_i с интенсивностью отказов элементов λ_i и их взаимной независимости случайная величина S_i будет иметь закон распределения Эрланга (m+1)-го порядка с плотностью вероятностей [10]:

$$f(t) = \frac{\lambda_i (\lambda_i t)^{k-1} e^{-\lambda_i t}}{(k+1)!},$$

где $k = m_i + 1$, $k \geq 0$.

Вероятность безотказной работы ИС в течение времени t_0 при наличии в ЗИП m_i = (k-1)запасных элементов i-го типа равно:

$$P(S_i \geq t_0) = 1 - \int_0^{t_0} f(t)dt = 1 - \sum_{r=0}^{k-1} (\lambda_i t_0)^r e^{-\lambda_i t_0} / r! \qquad (8.17)$$

Каждое r-е слагаемое в (8.17) это вероятность того, что за время t_0 откажет r элементов. Правая часть (8.17) выражает вероятность того, что за время t_0 произойдет не более чем m_i отказов. Поэтому:

$$P(S_i \geq t_0) = Q(m_i).$$

Задача поиска минимума вероятности отказов (8.16) может быть сформулирована следующим образом:

$$\min(1-\prod_{i=1}^{M}\sum_{r=0}^{m_i}(\lambda_i\ t_0)\,e^{-\lambda_i t_0}\ /\ r!) \qquad (8.18)$$

при условии $\sum m_i c_i \le c$.

При отсутствии ограничений на ресурсы задача комплектования ЗИП решается на основе оценки ожидаемого количества отказов в заданном интервале времени работы ИС.

Пусть задана вероятность наличия элементов в ЗИП p_i . Тогда вероятность появления (*k-1)* отказов будет:

$$1-\sum_{r=0}^{k-2}\ (\lambda_i\ \ t_0)^{\ r}\,e^{-\lambda_i t_0}/r!,$$

а появления *k* отказов:

$$1-\sum_{r=0}^{k-1}\ (\lambda_i\ \ t_0)^{\ r}\,e^{-\lambda_i t_0}/r!\ .$$

Тогда необходимо выбрать такое количество запасных элементов *k*-го типа в ЗИП, чтобы выполнялось неравенство:

$$(1-\sum_{r=0}^{k-2}(\lambda_i\ \ t_0)^{\ r}\,e^{-\lambda_i t_0}/\ r\ !)>(1-p_i)>$$

$$>(1-\sum_{r=0}^{k-1}\ (\lambda_i\ \ t_0)^{\ r}\,e^{-\lambda_i t_0}/r\ !)\ .$$

При этом количество запасных элементов по типам устройств будет различным, т.к. оно зависит от надежности устройств.

8.4. Организация обучения и тренировка пользователей информационной системы

Эффективность использования ИС во многом зависит от квалификации персонала, знаний, умений и навыков.

Получили распространение два вида обучающих программ: линейная и разветвленная методика обучения [10]. Обучающие программы, как правило, рассчитываются на пользователей средней квалификации.

В линейных обучающих программах материал излагается последовательно, стимулирующим фактором в обучении является подтверждение правильности выдаваемых ответов.

В разветвленных программах учитываются ответы обучаемого. От их верности выбирается дальнейший порядок обучения. При совершении непредусмотренных ошибок повторяется задание обучаемому, выдав ему некоторые общие указания.

Применяются также обучающие программы с элементами линейной и разветвленной методик обучения.

Перед организацией процесса обучения составляется программа обучения. Разработка обучающей программы предусматривает определение объема знаний и навыков, которые должны получать обучаемые. Подбираются темы и вопросы для обучения, составляются структурная схема программы, на основе которой будет вестись обучение.

Модель обучения ориентирована на конкретный тип ИС. Рассмотрим некоторые модели обучения операторов на примере информационно-управляющих систем (ИУС).

Модель 1

Обучаемому выдается предупредительный сигнал из множества сигналов $c_i \in \{C\}$. Обучаемый отвечает на этот сигнал, управляющей реакцией из множеств возможных $r_i \in \{R\}$. На основании анализа правильности r_i обучаемый наказывается штрафом S_k и процесс обучения повторяется. В результате накопления опыта обучаемый будет выбирать управляющее воздействие, минимизирующее штраф. Данная модель обучения используется в тех случаях, когда имеется полный перечень исходных ситуаций и возможных действий оператора, взаимосвязь между действиями и реакцией обучающего.

Модель 2

На каждом этапе обучения обучаемому предоставляется одна из N возможных ситуаций, при этом фиксируется реакция оператора b_i , i = 1, 2,... N, на данную ситуацию и затрачиваемое при этом время τ_i. По каждой ситуации вычисляется значение функции «успеха»:

$$a_i = b_i \exp(-\alpha(\tau_i - \tau_{доп})),$$

где $b_i = \begin{cases} 1 - \textit{правильное действие}, \\ 0 - \textit{неправильное действие} \end{cases}$

Качество работы обучаемого оценивается по всем ситуациям:

$$A = \sum_{i=1}^{N} a_i .$$

Задача оптимального управления процессом обучения сводится к достижению максимального значения А.

При обучении операторов ИУС могут применяться и другие методики обучения. Весьма перспективными являются адаптивные системы обучения. С ними можно ознакомиться в специальной литературе по обучающим системам.

Модель 3

Процесс обучения сводится к задаче стабилизации показателя качества работы обучаемого около заранее заданных значений:

$$| x_j - x_j^* | \le \Delta x_j , j=1, 2, ..m ,$$

где x_j – показатель качества работы; Δx_j – допустимы пределы отклонения.

Показателем может быть среднее количество правильных ответов «*a*» данных из M этапов обучения:

$$a = \frac{1}{N} \sum_{i=1}^{M} \sum_{j=1}^{n_j} x_j ,$$

где $x=\begin{cases}1-правильный\ ответ,\\ 0-неверный\ ответ.\end{cases}$

n_j – число ответов на i-м этапе; N – общее число данных.

Критерии оценки качества обучения весьма разнообразны. Задача поиска лучшего способа управления процессом обучения встречается в следующих двух вариантах:

1) Выбрать такую последовательность предъявления ситуаций, чтобы при заданном общем количестве предъявлений получить максимальное значение обобщенной характеристики уровня подготовки.

2) При заданной последовательности предъявления ситуаций так организовать обучение, чтобы достичь заданного показателя эффективности H за кратчайший интервал времени.

В качестве показателя эффективности для оценки степени подготовленности оператора ИУС выбирается время запаздывания:

$$t_з = \sum_{j=1}^{m} \frac{t_j}{m} \quad ,$$

где t_j – время от момента поступления информации до завершения операций по ее обработке; m – общее число операций.

В системе обучения операторов могут использоваться также адаптивные системы обучения.

8.5. Обеспечение заданных условий эксплуатации ИС

Работоспособность и эффективность информационной системы зависит от организации эксплуатационного обслуживания. В первую очередь это касается окружающей среды. Нормальными условиями работы системы являются [10]:

- температура окружающей среды $+20\pm5^0C$;
- относительная влажность воздуха 65±15%;
- атмосферное давление 84-106,7 кПа;
- уровень шумов не более 75 дб;

– запыленность при размере частиц до 3 мкм не более 1 мг/м^3;
– освещенность на высоте 0,8 м от пола 350-400 лкс.

Требуемый температурный режим поддерживается отводом тепловой энергии от нагретых поверхностей: естественным воздушным охлаждением, принудительным воздушным охлаждением, жидкостным охлаждением.

При эксплуатации комплекса технических средств (КТС) необходимо обеспечить меры безопасности: электробезопасность для установок напряжением до 1000 В; пожарную безопасность; эргономические требования для обслуживающего персонала.

В процессе эксплуатации ИС проводятся планово-профилактическое техническое обслуживание КТС. В процессе технического обслуживания определяются и заменяются элементы, параметры которых вследствие износа находятся на пределе допустимого, проверяются элементы и узлы, не охваченные системой автоматического контроля исправности ТС.

Планово-профилактическое техническое обслуживание выполняется периодически. Различают следующие виды обслуживания:

1) суточное обслуживание продолжительностью 1 ч.;
2) недельное обслуживание продолжительностью 6 ч.;
3) месячное обслуживание продолжительностью 16 ч.;
4) полугодовое обслуживание продолжительностью 72 ч.

Продолжительность обслуживания определяется техническими условиями на устройства.

Суточное обслуживание включает: внешний осмотр ТС; чистоту устройств; проверку температуры и влажности в помещении; работы суточного регламента технических средств.

Недельное техническое обслуживание дополнительно включает осмотр креплений разъемов и ТЭЗ-ов. Проводятся недельные регламентные работы технических средств. Проверяется работа ЭВМ с помощью комплекса программ технического обслуживания.

Месячное техническое обслуживание дополнительно включает проверку работы органов управления и сигнализации; функционирование ЭВМ при изменении на ±5% напряжений вторичных источников питания. Обслуживание кончается проверкой ЭВМ с помощью комплекса программ технического обслуживания при номинальном напряжении питания.

При полугодовом техническом обслуживании дополнительно проводится осмотр и чистка ТЭЗ-ов, блоков питания, кабелей и жгутов. Прове-

ряется заземление путем измерения переходного сопротивления. Выполняются работы полугодового регламента технических средств. Проверяется защита системы электропитания и сигнализации о перегреве. Измеряется и регулируется частота задающего генератора. Обслуживание завершается проверкой функционирования ЭВМ при изменении на ±5% напряжения вторичных источников питания и прогонкой при номинальном напряжении комплекса программ технического обслуживания.

Процесс технического обслуживания считается законченным, если не было сбоев при выполнении комплекса программ технического обслуживания.

ЛИТЕРАТУРА

1. Баронов В.В., Калянов Г.Н., Попов Ю.И., Титовский И.Н. Информационные технологии и управление предприятием. М.: Компания АйТи, 2004. -328с.

2. Садердинов А.А., Трайнев В.А. Построение комплексных программно-технических проектов интегрированных систем организационного управления (обобщение теории и практики проектирования).- М.: изд. Книготорговый центр «Маркетинг», 2001.-287с.

3. Нетесова О.А. Методы оценки эффективности автоматизированных информационных систем. Вологодская государственная молочнохозяйственная академия. - Вологда. 2004. -116с.

4. Костров А.В., Матвеев Д.А. Информационный менеджмент. Оценка эффективности информационных систем: Учеб. пособие/Владимирский государственный университет. - Владимир. 2004. -116с.

5. Каплан Р.С., Нортон Д.П. Сбалансированная система показателей. От стратегий к действию. - 2-е изд. -М.: ЗАО «Олимп-Бизнес», 2003. -320с.

6. Информационные системы / Петров В.Н. –СПб.: Питер, 2003. -688с.

7. Системный анализ и принятие решений: словарь-справочник: Учеб. пособие для вузов/Под ред. В.Н. Волковой, В.Н. Козлова. - М.: Высш. Шк. 2004 -616с.

8. Мамиконов А.Г. Основы построения АСУ: Учебник для вузов. - М.: Высш. Школа, 1981.-248с.

9. Автоматизированные информационные технологии в экономике: Учебник/ М.И. Семенов, И.Т., Трубилин, В.И. Лойко, Т.П. Барановская; Под общ. ред. И.Т Трубилина. - М.: Финансы и статистика, 2001.-416с.

10. Хетагуров Я.А., Древс Ю.Г. Проектирование информационно-вычислительных комплексов: Учебник для вузов. М.: Высшая школа, 1987. -280 с.

11. Проектирование экономических информационных систем: Учебник/Г.Н. Смирнова, А.А. Сорокин А.А., Ю.Ф. Тельнов; Под ред. Ю.Ф. Тельнова. - М.: Финансы и статистика, 2002. -512с.

12. Вендров А.М. Проектирование программного обеспечения экономических информационных систем: Учебник. -2 изд. - М.: Финансы и статистика, 2005. -544с.

13. Методы и модели информационного менеджмента: учеб. пособие/Д.В. Александров, А.В. Костров, Р.И. Макаров, Е.Р. Хорошева; под. ред. А.В. Кострова.- М.: Финансы и статистика, 2007.-336с.

14. Мертенс П. Интегрированная обработка информации. Операционные системы в промышленности: учебник/пер. с нем. М.А. Костровой. – М.: Финансы и статистика, 2007.-424с.

Учебное издание

МАКАРОВ Руслан Ильич
ХОРОШЕВА Елена Руслановна

МЕТОДОЛОГИЯ ПРОЕКТИРОВАНИЯ
ИНФОРМАЦИОННЫХ СИСТЕМ

Учебное пособие

Подписано в печать 14.08.08.
Формат 60x84/16. Усл. печ. л. 19,53. Тираж 500 экз.
Заказ
Издательство Владимирского государственного университета
600000, Владимир, ул. Горького, 87