**МИНИСТЕРСТВО ОБРАЗОВАНИЯ АЗЕРБАЙДЖАНСКОЙ РЕСПУБЛИКИ**

**АЗЕРБАЙДЖАНСКИЙ ГОСУДАРСТВЕННЫЙ ЭКОНОМИЧЕСКИЙ УНИВЕРСИТЕТ**

**МАГИСТРАТУРА**

***На правах рукописи***

**Махмудзаде Джавид Намиг оглы**

# ТЕМА: «ТЕХНОЛОГИЧЕСКИЕ ПРИНЦИПЫ РЕАЛИЗАЦИИ ПРИКЛАДНОЙ ФУНКЦИОНАЛЬНОСТИ В КОРПОРАТИВНЫХ ИНФОРМАЦИОННЫХ СИСТЕМАХ»

**МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ**

| | | |
|---|---|---|
| **Название и шифр направления:** | **IIM 020000** | **Экономика предприятия и управление** |
| **Название и шифр специальности:** | **IIM 020004** | **Информационные технологии управления** |

**Научный руководитель** **Проф. А.К. Керимов**

**Заведующий кафедрой** **Акад. А.М. Аббасов**

**БАКУ-2015**

## СОДЕРЖАНИЕ

# РЕФЕРАТ

**Актуальность темы**. Актуальность вопросов информатизации всех сфер общественно – экономической жизни сегодня вполне очевидна. Потребность в применении эффективных и адекватных реальной действительности компьютерных программ и технологий сегодня возрастает. Компьютерная технология дает возможность оптимизировать и рационализировать управленческую функцию за счет применения новых средств сбора, передачи и преобразования информации.

Информатизация в области управления экономическими процессами полностью автоматизировала неформализованную сферу экономики, в частности – бухгалтерский учет и аудит. Ведение информационных систем повысили оперативность, качество обработки информации, производительность труда, профессиональную грамотность специалистов, занятых управленческой деятельностью. Использование современных программных средств позволили создать информационный фонд, локальную информационную базу, результатную информацию в стандартной для пользователя форме при решении учетных задач.

В условиях рыночной конкуренции оперативность и точность обработки информации является неотъемлемым фактором при принятии оптимальных решений задач управления в экономике. Применение новых информационных технологий в финансовом и банковском менеджменте позволили банкам активнее реализовать новые виды услуг, осуществлять международные платежи, фондовые и трастовые операции, предоставлять свои услуги по сети INTERNET.

Развитые функциональные и информационные связи в финансовом и банковском менеджменте позволили связать между собой органы казначейства, систему банковских платежей, систему государственной налоговой службы, получателей бюджетных средств и налогоплательщиков для проведения оперативного учета и контроля операций. Обеспечение

функций бизнеса определяется уровнем использования информационных систем с пониманием их организационной концепции и разветвленностью информационных процессов, их построения, а также возможностью применения в деловых процессах.

Информационные системы интегрируют информационные ресурсы, процессы, людей и средства, которые собирают, преобразуют и распространяют информацию в организации. Целью информационных систем является переработка исходных данных в информационные продукты, необходимые для конкретного пользователя. Задачей этих систем является обеспечение принятия решений относительно развития управляемого объекта на основе компьютерных информационных технологий.

Особенность информационной технологии - базового элемента систем состоит в использовании многоуровневой декомпозиции экономической системы и процесса ее моделирования. Количество уровней определяется сложностью моделируемого объекта и степенью детализации построенной модели. Метод позволяет представить систему в виде множества подсистем и элементов, а процесс моделирования разделить на составляющие его процедуры с помощью специальных языковых средств. Для этого используется гипертекстовая технология, которая дает возможность повысить качество проектируемой системы, сократив затраты и сроки ее разработки, повысить производительность труда проектировщиков.

Имеющиеся научные разработки, но этой теме не рассматривают вопросы комплексного моделирования и проектирования в цепи: "деловая стратегия - информационная система - информационная технология".

**Цель исследования.** Цель, с которой написана данная работа, состоит в исследовании возможностей  корпоративных информационных систем, которые могут быть использованы, как удобный механизм описания стратегических альтернатив и принятия одной из области согласия или, что еще интересней- из области компромиссов для организаций, которые не в состоянии (финансовом или ином, например, отсутствие достаточной

информации для разработки «идеальной» стратегии, что является одной из основных проблем вообще для разработки любой стратегии) обладать достаточно формализованным процессом такого выбора. Также их сферой применения может быть промежуточный этап разработки стратегии, а вместе с комплексом решений способность оценить необходимость и достаточность разработки стратегии для организации на начальном этапе, во-первых, в общем представлении общих знаний о совокупности моделей и методов внутрифирменного управления, а, во-вторых, в приведении достаточно ясных и простых примеров построения и исследования моделей и реализации методов, направленных на совершенствование внутрифирменного управления. Особое внимание уделяется применению методов автоматизации и рационализации процедур функционирования и информационных технологий управления бизнес-процессами. Для достижения этой цели проведена классификация информационных систем и информационных технологий, разработаны методологические и методические основы процессов организации информатизации функций бизнеса; разработана система информатизации, обеспечивающая взаимодействие блоков информационных технологий и информационных систем по управлению изменениями развития производственных систем.

**Предмет исследования**. Предметом разработки является механизм взаимодействия информационных технологий, а объектом - функциональные блоки производственной системы бизнеса.

**Теоретической основой диссертации** служат исследования специалистов и аналитиков в области, как экономической теории, интеллектуального анализа данных, так и других областей прикладной экономики, таких как, общей и экономической статистики, экономико-математических методов, технологии создания баз данных, баз знаний и экспертных систем, поскольку область извлечения знаний имеет мультидисциплинарный характер.

**Методологической базой исследования** являются современные работы отечественных и зарубежных специалистов поданной проблеме.

Приведенная методология и методические основы - дополнение известных научных разработок в области теории экономической информатики в части моделирования процессов проектирования информационных технологий управленческих решений по осуществлению функций бизнеса производственных систем.

**Научная новизна**. Научная новизна магистерской диссертации состоит в исследовании теоретико-методических основ формирования кластеров как одной из форм повышения активности организаций различной сферы.

**Теоретическая и практическая ценность** исследования состоит из основных положений, рекомендаций и заключений по поводу организации работы информационных систем.

**Структура и объем работы**. Диссертация состоит из введения, трех глав, заключения и рекомендаций.

**Первая глава** работы посвящена исследованию корпоративных информационных систем - стратегических ИС, представляющих собой совокупность технических и программных средств, реализующих идеи и методы автоматизации всех функций управления предприятием. Далее исследуется специализация КИС — мониторинг событий и трендов, как внутренних, так и внешних, проведен анализ мирового рынка КИС, выделены методологии моделирования информационных систем.

**Во второй главе** дается обзор компонентов, которые КИС должна включать в себя, автоматизирующие как основные, так и вспомогательные бизнес-процессы. Автоматизируя бизнес-процессы по отдельности, не принимая во внимание интересы всей организации, не соблюдая системотехнические принципы, в конце концов, можно прийти к "лоскутной" автоматизации, которая будет в дальнейшем тормозить процесс автоматизации и вести к неоправданно значительным затратам. Только объединив компоненты в единое целое, можно получить систему, обладающую свойствами КИС, и действительно полезную организации.

**Третья глава** посвящена моделям построения программ. Первая называется процедурно-ориентированной и в ней программа представляется как ряд последовательно выполняемых операций (процедур). Программы, построенные с использованием процедурно-ориентированной модели, можно рассматривать как код, воздействующий на данные. Языки программирования, в которых реализован процедурно-ориентированный подход к построению программ, называются процедурными. До определенного времени процедурно-ориентированный подход успешно применялся при разработке программ. Однако по мере увеличения объема и усложнения программ при использовании данного подхода возникают существенные проблемы.

В объектно-ориентированной модели программа рассматривается как совокупность объектов – отдельных фрагментов кода, обеспечивающих выполнение определенных действий и объединяющих данные и методы управления ими. Взаимодействие между объектами производится через определенные интерфейсы. Объектно-ориентированные программы можно характеризовать как данные, управляющие доступом к коду. Объектно-ориентированный подход повышает надежность разрабатываемых программ и обеспечивает возможность многократного использования кода.

В конце работы даны заключение и литература.

# ВВЕДЕНИЕ

Как свидетельствует мировой опыт, экономический рост государства существенно связан с развитием и использованием в экономике современных средств и технологий автоматизации. Успешное ведение бизнеса сегодня практически невозможно без использования современных информационных систем. В наиболее развитых странах именно сектор высоких технологий обеспечивает их процветание и богатство.

Современные корпоративные информационные системы играют в наше время такую же роль, какую сыграло появление машин в XIX веке. Фактически, они стали основной движущей силой научно-технической революции и развития современной мировой экономики. Умело выбранная и внедренная КИС существенно улучшает управляемость предприятия и повышает эффективность его работы [5].

Современная информационная система в масштабе корпорации - это комбинация, тесное переплетение различных информационных технологий, предлагаемых сегодня на рынке. Искусство создания таких систем - в сбалансированной интеграции этих технологий и соответствующих программных и аппаратных средств. Некоторые подходы к этой интеграции описаны в данной статье.

Необходимо отметить, что построение корпоративных систем - дело не одной недели и даже не одного года. Это, как все уже понимают, не просто покупка компьютеров и, в лучшем случае, связывание их между собой. Это, прежде всего, осмысление своего бизнеса, понимание путей его развития и неизбежный вывод о том, что успех в настоящем и будущем может быть обеспечен только в случае, если удастся правильно организовать управление информацией. Это возможно сделать через корпоративную информационную систему, которая должна стать не только основой информационного пространства компании, но и гибким инструментом управления бизнесом в сложных, постоянно меняющихся условиях [9].

# ГЛАВА I. ПОНЯТИЕ КОРПОРАТИВНОЙ ИНФОРМАЦИОННОЙ СИСТЕМЫ – КИС. ОПРЕДЕЛЕНИЯ И НАЗНАЧЕНИЯ КИС

Корпоративная информационная система (КИС, EIS - Executive Information System) – это стратегическая ИС, представляющая собой совокупность технических и программных средств, реализующих идеи и методы автоматизации всех функций управления предприятием. Такая ИС является многопользовательской, функционирует в распределенной вычислительной сети. И хотя понятие корпоративности подразумевает наличие довольно крупной, территориально-распределенной информационной системы, все же вполне правомерно присовокупить сюда системы любых предприятий, вне зависимости от их масштаба и формы собственности.

Корпоративные ИС предназначены для обеспечения большинства бизнес-процессов (желательно всех) всего предприятия (нескольких предприятий), сбора и анализа информации о предприятии и внешней среде с целью решения задач управления предприятием как по вертикали (от первичной информации до поддержки принятия решений высшим руководством), так и по горизонтали (все направления деятельности и технологические операции). Для таких систем характерно высокое быстродействие и чрезвычайная простота в использовании, однако, функциональность подобных систем с точки зрения анализа обычно крайне ограничена.

КИС помогают исполнителям анализировать важную информацию и использовать соответствующие инструментальные средства, чтобы направлять ее для создания стратегических решений в организации. Так, например КИС помогают исполнителям разрабатывать более точное и актуальное целостное изображения операций организации, а также и конкурентов, поставщиков и потребителей (заказчиков) .

## § 1.1. Специализация корпоративной информационной системы

Специализация КИС — мониторинг событий и трендов, как внутренних, так и внешних. Владея своевременной и более широкой информацией и соответствующими инструментальными средствами, менеджеры высшего уровня

лучше готовятся к принятию стратегических изменений для использования возможностей организации и устранения проблем. КИС могут быть конкурентным оружием и инструментальным средством стратегического планирования; улучшать качество решений, которые создаются на высшем уровне; уменьшать объем времени на отслеживание (выявление) проблем и возможностей; улучшить качество планирования на верхних уровнях управления организацией; обеспечивать механизм для улучшения контроля в организации; обеспечивать более скорый и лучший доступ к данным и моделям [11,25,28].

В последнее время, все больше руководителей начинают отчетливо осознавать важность наличия на предприятии КИС, как необходимого инструментария для успешного управления бизнесом в современных условиях. ИС является высоко эффективной, если деятельность предприятия рассматривать как цепь действий, в результате которых происходит постепенное формирование стоимости производимых продуктов или услуг. Тогда с помощью ИС различного функционального назначения, включенных в эту цепь, можно оказывать влияние на стратегию принятия управленческих решений, направленных на увеличение доходов фирмы.

## § 1.2. Мировой рынок КИС

В настоящее время выделяют следующие виды КИС: управления ресурсами предприятий (ERP); управления взаимоотношениями с заказчиками (CRM); управления цепью поставок (SCM) и ряд других, появившихся в последнее время (например, системы электронной коммерции и системы управления имуществом предприятий EAM ( Enterprise Asset Management )).

Интересны и данные, полученные AMR Research после изучения 13 отраслей американской промышленности и 800 компаний. Целью исследования являлось выяснение вопроса, на какие КИС предприятия тратят свои деньги. Результаты исследования выглядят следующим образом: ERP — 43%; CRM — 17%; SCM — 13%; другие КИС — 27%. При этом, высокотехнологичные компании тратят 28% бюджета на КИС, фармацевтические — 20%, а финансовые — 15%.

По различным оценкам в настоящее время на мировом рынке существует более 500 КИС. На рынке ERP-систем бесспорно лидируют компании SAP AG,

Oracle, J.D. Edwards, PeopleSoft, Baan , на их долю пришлось 64% объема данного рынка. При этом на долю европейских фирм приходится 45% мирового рынка. Только на долю SAP AG приходится 33% рынка ERP-систем. Из прочих разработчиков КИС можно также отметить: производителей SCM-систем — компании i2, Manugistics и Numetrix ; производителей CRM-систем — Siebel, Vantive, Clarify и Pivotal; производителей систем электронной коммерции — Ariba, Commerce One, IBM и Broadvision.

Кроме того, заметное место на рынке КИС занимают следующие компании; Brain; Sage Group; Symix Systems; Geac Computer; SCT; IBS; Epicor Software; QAD/BMS; Exact Software; IFS и ряд других [19, 34].

**SAP AG (ERP-система SAP R/3)**

Это четвертая по размерам компания-разработчик ПО в мире. Основной продукт компании — ERP-система SAP R/3, в которой реализовано более 1000 бизнес-процессов. В настоящее время разработано 46 версий этой системы на 28 языках. Система внедрена в различных отраслях: аэрокосмической и оборонной; автомобилестроении; банковском деле; химической промышленности; производстве потребительских товаров; проектировании и строительстве; здравоохранении; страховании; СМИ; фармацевтике; розничной торговле и др.

С середины 1990-х гг. компания уделяет большое внимание интеграции своих продуктов с Интернет. Главный акцент SAP — Интернет-портал MySAP.com (поддерживаемый Sun-серверами), с помощью которого по запросам клиентов предоставляется открытая среда персональных решений для совместного ведения бизнеса на базе Интернет, и ПО для онлайновой деятельности. Вторым по важности вопросом для SAP является развитие CRM-технологий. Все модули интегрированы с R/3, но их можно использовать и отдельно.

**Oracle (КИС Oracle Applications)**

Это вторая по величине в мире разработчик ПО. Одними из основных продуктов Oracle являются CRM и ERP приложения, а также ПО для электронной коммерции (E-Business Suite) . Основным ERP-продуктом Oracle является КИС Oracle Applications. Oracle . Комплект продуктов Oracle CRM охватывает все стороны взаимодействия предприятия со своими заказчиками — от маркетинга и

продаж до сервиса — и обеспечивает бесшовную интеграцию с клиентской частью (Front-Office) и с внутренней бизнес-логикой (back-end) ERP-системы.

**Baan (CRM, ERP, SCM и Corporate Knowledge Management)**

Выпускает интегрированные решения и компонентные приложения в области E-Business и электронной коммерции, CRM, ERP, SCM и Corporate Knowledge Management. У Baan — более 15000 инсталляций. Ее стратегические партнеры: IBM, Microsoft, Hewlett Packard, Compaq, Sun Microsystems и др..

Система Baan предназначена для комплексной поддержки бизнеса. Это настраиваемая система, все подсистемы которой конфигурируются под процедуры и задачи бизнеса заказчика. В нее входят подсистемы, решающие перечисленные ниже задачи:

- **поддержка управления проектами** - комплексная поддержка процессов планирования, управления и контроля выполнения разнообразных программ в рамках всей компании; комбинированный учет затрат по проекту, оценка стоимости выполнения программы и т.д.
- **управление потоками хозяйственных операций** - планирование и контроль за ходом хозяйственных операций с целью автоматизации процессов во всех сферах текущей деятельности предприятия.
- **управление финансовыми средствами** - управление наличностью, планирование и управление ценными бумагами; контроль за ликвидностью средств, оценки рисков и т.д.
- **управление инвестициями** - контроль за капиталовложениями и бюджетом, учет расходования средств, анализ прибыльности инвестиционных проектов и т.д.
- **мониторинг текущей деятельности предприятия** - поддержка процессов принятия решений; обеспечение точного контроля за стратегической и текущей финансовой информацией в режиме реального времени и возможности в любой момент времени предоставлять интегрированные данные о состоянии предприятия и т.д.
- **финансовый учет и отчетность** - полный спектр работ по ведению и составлению внешней отчетности (ведение Главной книги, составление бухгалтерской отчетности, составление консолидированной отчетности и т.д.).

- **учет затрат** - весь спектр работ по ведению и составлению внутренней отчетности (учет затрат по продуктам и организационным единицам, анализ прибыльности, расчет косвенных затрат и т.д.).
- **поддержание функций закупки и сбыта продукции** - анализ и обработка поступающих заказов; поддержка процессов, связанных с прогнозированием, составлением и оценкой бюджетов различных уровней и т.д. **управление движением материальных средств** - поддержка всех действий, связанных с управлением складами, учетом материальных средств в местах хранения (инвентаризация), функций транспортировки грузов, учета средств в развитии (при передвижении в процессе выполнения производственных заказов) и т.д.
- **планирование производства** - комплекс работ, связанных с подготовкой производства, обеспечением производственных цепочек необходимыми ресурсами, выполнением производственных заказов, контролем текущего состояния производства, внесением оперативных изменений в зависимости от изменения текущей ситуации (диспетчеризация) и т.д.
- **поддержка обеспечения качества продукции** - выполнение требований международных, государственных и отраслевых стандартов производства; согласование параметров качества продукции, предоставляемых услуг и текущего производственного процесса; контроль процессов испытания выпускаемой продукции и т.д.
- **поддержание сервисных функций жизнедеятельности** - предоставление дополнительных возможностей создания различных классификаторов; поддержание служб контроля за изменениями; обеспечение документооборота, связь с внешними системами и т.д. [22, 37, 39]

Система Baan обладает развитым инструментарием; кроме того, особого упоминания заслуживает пакет "Динамическое моделирование предприятия" (DEM), который может динамически перенастраиваться, позволяя проводить реинжиниринг бизнес-процессов в ходе внедрения и в процессе дальнейшей эксплуатации.

Для создания моделей предприятия в подсистему "Моделирование предприятия" встроен графический инструментарий, позволяющий наглядно

представить бизнес-функции, бизнес-процессы и организационную структуру предприятия. Кроме того, подсистема обеспечивает настройку параметров и конфигурацию системы Baan с учетом формализованных бизнес-процессов и позволяет задать различные виды взаимодействия между ними. Концепция динамического моделирования предусматривает возможность изменять уже созданную бизнес-модель при возникновении новых задач [27].

**J.D.Edwards (OneWorld)**

Сначала компания занималась разработкой ПО для малых и средних компьютеров, фокусируясь в начале 1980-х гг. на платформе IBM System /38. К середине 1980-х гг. J.D. Edwards разработала КИС World Software для платформы IBM AS/400 (прямого потомка System /38). В июне 1996 г . была выпущена кросс-платформенная КИС J.D. Edwards OneWorld, представляющая собой сетецентрическое решение, удовлетворяющее требованиям ERP-стандарта и состоящее из 3 основных интегрированных подсистем: финансы; сбыт/снабжение; производство. Система предназначена для комплексной автоматизации крупных и средних предприятий различных вертикальных рынков. Все бизнес-процессы представлены на графическом уровне, что делает возможным их настройку и перенастройку, не прибегая к дополнительному программированию. Кросс-платформенность и развитые механизмы репликации позволяют работать в режиме распределенной сети. С помощью набора технологических средств ActivEra реализована концепция «от мысли к действию» — возможность адаптации ПО к постоянно изменяющимся условиям бизнеса. OneWorld обладает возможностью объединять различные платформы на базе одного инструментария. Система работает на различных ОС, включая вариации UNIX, а также Windows [30, 31].

**PeopleSoft**

Занимается разработкой ПО управления предприятиями (управлением персоналом; управлением финансами; управлением дистрибьюцией; управлением производством и управлением цепью поставок). Наиболее известным продуктом PeopleSoft является ERP-система с одноименным названием PeopleSoft, в состав которой входит также следующее ПО: PeopleSoft HRMS — приложения управления персоналом (включают Human Resources, Benefits Administration, FSA Administration, Payroll, Payroll Interface, Time and Labor, Pension и Stock

Administration); PeopleSoft Treasury Management — управление финансами; PeopleSoft Project Management — управление проектами; PeopleTools — интегрированный набор разработки и настройки клиент-серверных бизнес-приложений; Procurement — поддерживает закупки, управление запасами, обработку платежей и затрат, управление собственностью [6, 15].

## §1.3. Методологии моделирования информационных систем. Проектирование КИС

Достижение намечаемых целей по внедрению комплексной информационной системы (КИС) предприятий и организаций может быть осуществлено лишь при детальном и всестороннем планировании всего комплекса работ по внедрению. Решение этой задачи осуществляется в рамках проектирования КИС.

При проектировании КИС выполняются следующие работы:

1. обследование Заказчика, выявление профиля деятельности, организационной структуры, бизнес - процессов, порядка функционирования системы планирования и управления, текущего использования средств автоматизации;
2. определение функциональных задач, подлежащих автоматизации в рамках КИС, определение в рамках каких организационных структур в каком порядке и последовательности должны решаться данные задачи;
3. моделирование схем бизнес - процессов в условиях внедрения КИС
4. определение требуемого состава и структуры АРМов системы;
5. проектирование схем информационных потоков и документооборота КИС;
6. определение требуемых технических и программных средств КИС - формирование программно-технических спецификаций;
7. проектирование инфраструктуры КИС – сетевых схем, схем расположения оборудования и пр.

8. определение состава работ по системной интеграции КИС – подготовка ТЗ на системную интеграцию;

9. определение состава работ по настройке, адаптации и доработке КИС с учетом специфики Заказчика – подготовка ТЗ по доработке КИС;

10. определение требуемого информационного насыщения КИС – подготовка ТЗ на информационное насыщение;

11. разработка руководящей и организационной документации по порядку эксплуатации КИС – положений по службам, должностных инструкций и пр. – такая документация включается в документацию системы управления качеством и выполняется в соответствии с международными стандартами серии ISO 9000;

12. определение требуемых работ по подготовке кадров – определение состава и количества учебных групп, состава и объема учебных курсов;

13. разработка календарного плана работ по внедрению КИС;

14. калькуляция затрат на внедрение КИС;

15. разработка плана финансирования работ по внедрению КИС [33, 41].

В основе проектирования информационной системы (ИС) лежит моделирование предметной области (МПО).Чтобы получить адекватный предметной области проект ИС необходимо иметь целостное, системное представление модели, которая должна отражать все аспекты функционирования будущей ИС. При этом подмоделью предметной области понимается система, имитирующая структуру или функционирование исследуемой предметной области и отвечающая основному требованию - быть адекватной этой области.

Предварительное моделирование предметной области позволяет сократить время и сроки проведения проектировочных работ и получить эффективный и качественный проект.

К МПО предъявляются следующие требования:

формализация, обеспечивающая однозначное описание структуры предметной области;•

понятность для заказчиков и разработчиков на основе применения графических средств отображения модели;•

реализуемость, подразумевающая наличие средств физической реализации модели предметной области в ИС;•

обеспечение оценки эффективности реализации модели предметной области на основе определенных методов и вычисляемых показателей.•

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

Структурный аспект предполагает построение:

объектной структуры, отражающей состав взаимодействующих в процессах материальных и информационных объектов предметной области;•

функциональной структуры, отражающей взаимосвязь функций (действий) по преобразованию объектов в процессах;•

структуры управления, отражающей события и бизнес-правила, которые воздействуют на выполнение процессов;•

организационной структуры, отражающей взаимодействие организационных единиц предприятия и персонала в процессах;•

технической структуры, описывающей топологию расположения и способы коммуникации комплекса технических средств.•

Для отображения структурного аспекта МПО и представления информации о компонентах системы в основном используются графические методы, которые должны обеспечить структурную декомпозицию спецификаций с максимальной степенью детализации. Осуществляется выбор языка представления проектных решений. Язык моделирова- ния - это нотация, в основном графическая, которая используется для описания проектов. Нотация представляет собой совокупность графических объектов, используемых в модели. Нотация является синтаксисом языка моделирования. Язык моделирования должен делать решения проектировщиков понятными пользователю и предоставлять проектировщикам средства достаточно формализованного и однозначного определения проектных решений, подлежащих реализации в виде программных комплексов, образующих целостную систему программного обеспечения [4, 42].

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

Оценочные аспекты МПО связаны с разрабатываемыми показателями эффективности автоматизируемых процессов, к которым относятся:

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

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

В основе различных методологий моделирования предметной области ИС лежат принципы последовательной детализации абстрактных категорий. Обычно, модели строятся на трех уровнях: на внешнем уровне (определении требований), на концептуальном уровне (спецификации требований) и внутреннем уровне (реализации требований).

На внешнем уровне модель отвечает на вопрос, ЧТО должна делать система (определяются компоненты системы - объекты).

На концептуальном уровне модель отвечает на вопрос, КАК должна функционировать система (определяется характер взаимодействия компонентов системы).

На внутреннем уровне модель отвечает на вопрос, С ПОМОЩЬЮ каких программно-технических средств реализуются требования к системе. Согласно жизненному циклу ИС, описанные уровни моделей соответственно строятся на этапах анализа требований, логического (технического) и физического (рабочего) проектирования.

Рассмотрим особенности построения МПО на трех уровнях детализации [18, 21, 24, 32].

**Объектная структура**. Объект - это сущность, которая используется при выполнении некоторой функции или операции. Объекты могут иметь динамическую или статическую природу. Динамические объекты используются

в одном цикле воспроизводства, например заказы на продукцию, счета на оплату; статические объекты используются во многих циклах воспроизводства, например, оборудование, персонал.

На внешнем уровне выделяются основные виды материальных объектов (например, сырье и материалы, услуги) и основные виды информационных объектов или документов (например, заказы, накладные).

На концептуальном уровне уточняется состав классов объектов, определяются их атрибуты и взаимосвязи. Строится обобщенное представление структуры предметной области.

На внутреннем уровне концептуальная модель отображается в виде файлов базы данных, входных и выходных документов ИС. Причем динамические объекты представляются единицами переменной информации или документами, а статические объекты - единицами условно постоянной информации в виде справочников, классификаторов.

**Функциональная структура**. Функция (операция) представляет собой преобразователь входных объектов в выходные. Функция может быть представлена одним действием или некоторой совокупностью действий. В последнем случае каждой функции может соответствовать некоторый процесс, в котором могут существовать свои подпроцессы, и т.д.

На внешнем уровне моделирования определяется список основных функций или видов процессов.

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

На внутреннем уровне отображается структура информационного процесса в компьютере - определяются иерархические структуры программных модулей, реализующих автоматизируемые функции [10].

**Структура управления**. В совокупности функций процесса возможны альтернативные или циклические последовательности в зависимости от условий протекания процесса. Эти условия связаны с происходящими событиями во внешней среде или в самих процессах и с образованием определенных состояний объектов. События вызывают выполнение функций, которые, изменяют состояния

объектов и формируют новые события, и т.д., пока не будет завершен некоторый процесс.

Каждое событие описывается с двух точек зрения: информационной и процедурной.

Информационно событие отражается в виде некоторого сообщения, фиксирующего факт выполнения некоторой функции изменения состояния или появления нового. Процедурно событие вызывает выполнение новой функции, и поэтому для каждого состояния объекта должны быть заданы описания этих вызовов.

На внешнем уровне определяются список внешних событий и список целевых установок, которым должны соответствовать процессы.

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

На внутреннем уровне выполняется формализация правил в виде триггеров или вызовов программных модулей [20].

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

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

На концептуальном уровне для каждого подразделения задается организационно-штатная структура должностей.

На внутреннем уровне определяются требования к правам доступа персонала к автоматизируемым функциям информационной системы [23].

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

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

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

На внутреннем уровне строится модель «клиент-серверной» архитектуры вычислительной сети [1, 12, 13].

Для правильного отображения взаимодействий компонентов ИС важно осуществлять их совместное моделирование. Методология структурного системного анализа существенно помогает в решении таких задач.

Структурным анализом называют метод исследования системы, которое начинается с ее общего обзора, затем детализируется, приобретая иерархическую структуру с все большим числом уровней. Для таких методов характерно: разбиение на уровни абстракции с ограниченным числом элементов (3-7); ограниченный контекст, включающий существенные детали каждого уровня; использование строгих формальных правил записи; последовательное приближение к результату. Структурный анализ основан на двух базовых принципах: «разделяй и властвуй» и принципе иерархической упорядоченности. Решение трудных проблем путем их разбиения на множество меньших независимых задач (так называемых «черных ящиков») и организация этих задач в древовидные, иерархические структуры значительно повышают понимание сложных систем. Определим ключевые понятия структурного анализа.

Операция - элементарное (неделимое) действие, выполняемое на одном рабочем месте.

Функция - совокупность операций, сгруппированных по определенному признаку.

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

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

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

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

**Пакетный режим работы**. Вся управляющая информация для конкретного выполнения пакета передается в виде законченной программы на входном языке при запуске пакета, и дальнейшая работа пакета проходит без участия пользователя.

Пакетный режим удобен, когда:

а) требуется решать много однотипных задач с использованием одной и той же программы на входном языке;

б) время, затрачиваемое на решение каждой задачи, достаточно велико;

в) программа на входном языке сложна и имеет значительный объем.

**Диалоговый режим работы**. Большинство ППП, применяемых на персональных ЭВМ, ориентировано на диалоговое взаимодействие с пользователем в ходе решения задач.

Простейший диалоговый режим состоит в том, что пользователь инициирует выполнение пакета, вводит задание в форме программы на входном языке и на этом заканчивает управление выполнением пакета. Фактически этот режим отличается от пакетного только возможностью исправления ошибок в ПВЯ, повторного запуска пакета при неудачах.

Более сложный вариант диалогового режима, называемый режимом сопровождения, предусматривает возможность динамического управления выполнением пакета. Управляющая информация вводится по частям

и формируется пользователем в процессе работы с пакетом на основе анализа промежуточных результатов.

Выбор того или иного способа применения ППП зависит от многих факторов, из которых наиболее существенными являются возможности операционной системы и выбранного языка программирования, объемы обрабатываемых данных, продолжительность решения задачи, частота использования ППП, особенности квалификации пользователей пакета и требования к допустимому времени ожидания результатов расчетов [2, 35,36, 40].

# ГЛАВА II. МЕТОДЫ АНАЛИЗА И ПРОЕКТИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ КИС

В большинстве случаев основные бизнес-процессы в первом приближении идентичны для организаций, принадлежащих одной отрасли, или вообще являются уникальными. Для автоматизации таких процессов существуют отраслевые решения или специально разработанные под заказ системы.

Вспомогательные же бизнес-процессы практически идентичны на верхнем уровне абстракции во всех организациях (например, бухгалтерия, логистика, склад, кадры и т.п.), а существующие различия проявляются на нижних уровнях детализации. Такие бизнес-процессы очень хорошо изучены и формализованы. Существует большое количество систем для их автоматизации - от небольших узконаправленных систем (1С-бухгалтерия, БОСС-Кадровик и др.), до комплексных систем класса ERP (SAP/R3, Oracle Applications и др.).

Таким образом, КИС должна включать в себя компоненты, автоматизирующие как основные, так и вспомогательные бизнес-процессы. Автоматизируя бизнес-процессы по отдельности, не принимая во внимание интересы всей организации, не соблюдая системотехнические принципы, в конце концов, можно прийти к "лоскутной" автоматизации, которая будет в дальнейшем тормозить процесс автоматизации и вести к неоправданно значительным затратам. Только объединив компоненты в единое целое, можно получить систему, обладающую свойствами КИС, и действительно полезную организации.

## §2.1. Подходы к декомпозиции систем

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, который выработало человечество за всю свою историю, 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. Этот подход известен под самыми разными названиями, среди них такие, как «разделяй и властвуй» *{divide et impera),* иерархическая декомпозиция и др. 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» — Low Coupling);

• связность отдельных частей внутри каждой подсистемы должна быть максимальной (принцип «сильного сцепления» - High Cohesion).

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 фафических моделей: для описания архитектуры ПО с различныхточек 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.

С другой стороны, разработчики ПО исторически всегда стояли последними в ряду тех, кто использовал компьютерные технологии для повышения качества, надежности и производительности в своей собственной работе (феномен «сапожника без сапог»).

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 [7, 16].

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 описанийсистемных 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.

## §2.2. Методы анализа и проектирования ПО

Структурные методы являются строгой дисциплиной системного анализа и проектирования. Методы структурного анализа и проектирования стремятся преодолеть 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» необходим 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 со 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 [9, 14, 26].

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 (Structured Analysis and Design Technique);

• модель IDEF3;

• 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.

## §2.3. Метод функционального моделирования SADT (IDEFO)

Метод SADT разработан Дугласом Россом (SoftTech, Inc.) в 1969 г. *для* моделирования искусственных систем средней сложности. Данный метод успешно использовался в военных, промышленных и коммерческих организациях США для решения широкого круга задач, таких, как долгосрочное и стратегическое планирование, автоматизированное производство и проектирование, разработка ПО для оборонных систем, управление финансами и материально-техническим снабжением и др. Метод SADT поддерживается Министерством обороны США, которое было инициатором разработки семейства стандартов IDEF (Icam DEFinition), являющегося основной частью программы ICAM (интегрированная компьютеризация производства), проводимой по инициативе ВВС США. Метод SADT реализован в одном из стандартов этого семейства — IDEFO, который был утвержден в качестве федерального стандарта США в 1993 г.

Метод SADT представляет собой совокупность правил и процедур, предназначенных для построения функциональной модели объекта какой-либо предметной области. ***Функциональная*** *модель* SADT отображает функциональную структуру объекта, т.е. производимые им действия и связи между этими действиями.

Основные элементы этого метода основываются на следующих концепциях:

• графическое представление блочного моделирования. Графика блоков и дуг SADT-диаграммы отображает функцию в виде блока, а интерфейсы входа/выхода представляются дугами, соответственно входящими в блок и выходящими из него. Взаимодействие блоков друг с другом описывается посредством интерфейсных дуг, выражающих «ограничения», которые, в свою очередь, определяют, когда и каким образом функции выполняются и управляются;

• строгость и точность. Выполнение правил SADT требует достаточной строгости и точности, не накладывая в то же время чрезмерных ограничений на действия аналитика.

Правила SADT включают: ограничение количества блоков на каждом уровне декомпозиции (правило 3—6 блоков — ограничение мощности краткосрочной памяти человека), связность диаграмм (номера блоков), уникальность меток и наименований (отсутствие повторяющихся имен), синтаксические правила для графики (блоков и дуг), разделение входов и управлений (правило определения роли данных);

• отделение организации от функции, т.е. исключение влияния административной структуры организации на функциональную модель.

Метод SADT может использоваться для моделирования самых разнообразных процессов и систем. В существующих системах метод SADT может быть использован для анализа функций, выполняемых системой, и указания механизмов, посредством которых они осуществляются.

Результатом применения метода SADT является модель, которая состоит из диаграмм, фрагментов текстов и глоссария, имеющих ссылки друг на друга. Диаграммы — главные компоненты модели, все функции организации и интерфейсы на них представлены как блоки и дуги соответственно. Место соединения дуги с блоком определяет тип интерфейса. Управляющая информация

входит в блок сверху, в то время как входная информация, которая подвергается обработке, показана с левой стороны блока, а результаты (выход) показаны с правой стороны. Механизм (человек или автоматизированная система), который осуществляет операцию, представляется дугой, входящей в блок снизу.

Одной из наиболее важных особенностей метода SADT является постепенное введение все больших уровней детализации по мере создания диафамм, отображающих модель.

Построение SADT-модели заключается в выполнении следующих действий:

• сбор информации об объекте, определение его границ;

• определение цели и точки зрения модели;

• построение, обобщение и декомпозиция диаграмм;

• критическая оценка, рецензирование и комментирование.

Построение диаграмм начинается с представления всей системы в виде простейшего компонента — одного блока и дуг, изображающих интерфейсы с функциями вне системы. Поскольку единственный блок отражает систему как единое целое, имя, указанное в блоке, является общим. Это верно и для интерфейсных дуг — они также соответствуют полному набору внешних интерфейсов системы в целом.

Затем блок, который представляет систему в качестве единого модуля, детализируется на другой диаграмме с помощью нескольких блоков, соединенных интерфейсными дугами. Эти блоки определяют основные подфункции исходной функции. Данная декомпозиция выявляет полный набор подфункций, каждая из которых показана как блок, границы которого определены интерфейсными дугами. Каждая из этих подфункций может быть декомпозирована подобным образом в целях большей детализации [3, 17, 29].

Во всех случаях каждая подфункция может содержать только те элементы, которые входят в исходную функцию. Кроме того, модель не может опустить какие-либо элементы, т.е., как уже отмечалось, родительский блок и его интерфейсы обеспечивают контекст. К нему нельзя ничего добавить и из него не может быть ничего удалено.

Модель SADT представляет собой серию диаграмм с сопроводительной документацией, разбивающих сложный объект на составные части, которые

изображены в виде блоков. Детали каждого из основных блоков показаны в виде блоков на других диаграммах. Каждая детальная диаграмма является декомпозицией блока из диаграммы предыдущего уровня. На каждом шаге декомпозиции диаграмма предыдущего уровня называется родительской для более детальной диаграммы.

Синтаксис диафамм определяется следующими правилами:

• диафаммы содержат блоки и дуги;

• блоки представляют функции;

• блоки имеют доминирование (выражающееся в их ступенчатом расположении, причем доминирующий блок располагается в верхнем левом углу диаграммы);

• дуги изображают наборы объектов, передаваемых между блоками;

• дуги изображают взаимосвязи между блоками:

выход-управление;

выход-вход;

обратная связь по управлению;

обратная связь по входу;

выход-механизм.

Одним из важных моментов при моделировании с помощью метода **SADT** является точная согласованность типов связей между функциями. Различают по крайней мере связи семи типов (в порядке возрастания их относительной значимости):

• случайная;

• логическая;

• временная;

• процедурная;

• коммуникационная;

• последовательная;

• функциональная.

***Случайная связь*** показывает, что конкретная связь между функциями незначительна или полностью отсутствует. Это относится к ситуации, когда имена данных на SADT-дугах в одной диаграмме имеют слабую связь друг с другом.

***Логическая связь*** — данные и функции собираются вместе благодаря тому, что они попадают в общий класс или набор элементов, но необходимых функциональных отношений между ними не обнаруживается.

***Временная связь*** — представляет функции, связанные во времени, когда данные используются одновременно или функции включаются параллельно, а не последовательно.

***Процедурная связь*** — функции сгруппированы вместе благодаря тому, что они выполняются в течение одной и той же части цикла или процесса.

***Коммуникационная связь*** — функции группируются благодаря тому, что они используют одни и те же входные данные и/или производят одни и те же выходные данные.

***Последовательная связь*** — выход одной функции служит входными данными для следующей функции. Связь между элементами на диаграмме является более тесной, чем в рассмотренных выше случаях, поскольку моделируются причинно-следственные зависимости.

*Функциональная связь* — все элементы функции влияют на выполнение одной и только одной функции. Диаграмма, являющаяся чисто функциональной, не содержит чужеродных элементов, относящихся к последовательному или более слабому типу связи.

## §2.4. Метод моделирования процессов IDEF3

Метод моделирования IDEF3, являющийся частью семейства стандартов IDEF, был разработан в конце 1980-х годов для закрытого проекта ВВС США. Этот метод предназначен для таких моделей процессов, в которых важно понять последовательность выполнения действий и взаимозависимости между ними.

Хотя IDEF3 и не достиг статуса федерального стандарта США, он приобрел широкое распространение среди системных аналитиков как дополнение к методу функционального моделирования IDEFO (модели IDEF3 могут использоваться для детализации функциональных блоков IDEFO, не имеющих диаграмм декомпозиции).

Основой модели IDEF3 *служит сценарий* процесса, который выделяет последовательность действий и подпроцессов анализируемой системы.

Как и в методе IDEFO, основной единицей модели IDEF3 является диафамма. Другой важный компонент модели — *действие,* или в терминах IDEF3 ***«единица работы» (Unit of Work - UOW).***

## §2.5. Моделирование бизнес-процессов и спецификация требований

Детализация бизнес-процессов осуществляется посредством бизнес-функций — совокупностей операций, сгруппированных по определенным признакам. Бизнес-функции детализируются с помощью элементарных бизнес-операций [8, 41].

4. Описание элементарной бизнес-операции осуществляется посредством задания алгоритма ее выполнения.

Принципы формирования бизнес-модели на верхних уровнях декомпозиции:

• следует избегать чрезмерной детализации (модель бизнес-процесса верхнего уровня должна содержать не более 6-8 блоков функций);

• следует использовать реально существующие названия функций или работ;

• не следует пытаться детально отразить всю существующую логику процесса (это будет сделано при формировании детальных моделей);

• важно отразить общую последовательность работ, подразделения участвующие в их исполнении, основные ресурсы;

• важно отразить основную логику процесса.

Общее число уровней в модели (включая контекстный) не должно превышать 5—6. Практика показывает, что этого вполне достаточно для построения полной функциональной модели современного предприятия любой отрасли.

При моделировании бизнес-процессов диаграммы потоков данных (DFD) используются для построения моделей «AS-IS» и «AS-TO-BE», отражая, таким образом, существующую и предлагаемую структуру бизнес-процессов организации и взаимодействие между ними. При этом описание используемых в организации данных на концептуальном уровне, независимом от средств реализации базы данных (СУБД), выполняется с помощью ERM.

Ниже перечислены основные виды и последовательность работ при построении бизнес-моделей с использованием методики Йордона.

Начальная контекстная диаграмма потоков данных должна содержать нулевой процесс с именем, отражающим деятельность организации, внешние сущности, соединенные с нулевым процессом посредством потоков данных. Потоки данных соответствуют документам, запросам или сообщениям, которыми внешние сущности обмениваются с организацией.

Определяется состав потоков данных и готовится исходная информация для построения концептуальной модели данных в виде структур данных. Выделяются все структуры и элементы данных типа «итерация», «условное вхождение» и «альтернатива». Простые структуры и элементы данных объединяются в более крупные структуры. В результате для каждого потока данных должна быть сформирована иерархическая (древовидная) структура, конечные элементы (листья) которой являются элементами данных, узлы дерева являются структурами данных, а верхний узел дерева соответствует потоку данных в целом. Результат можно представить в виде текстового описания, подобного описанию структур данных в языках программирования.

Для каждого класса объектов предметной области выделяется сущность. Устанавливаются связи между сущностями и определяются их характеристики (мощность связи и класс принадлежности). Строится диаграмма «сущность-связь» (без атрибутов сущностей).

Для завершения анализа функционального аспекта деятельности организации детализируется (декомпозируется) начальная контекстная диаграмма. При этом можно построить диаграмму для каждого события, поставив ему в соответствие процесс и описав входные и выходные потоки, накопители данных, внешние сущности и ссылки на другие процессы для описания связей между этим процессом и его окружением. После этого все построенные диаграммы сводятся в одну диаграмму нулевого уровня.

Проверяется соответствие между контекстной диаграммой и диаграммой нулевого уровня (каждый поток данных между системой и внешней сущностью на диаграмме нулевого уровня должен быть представлен и на контекстной диаграмме).

Процессы разделяются на группы, которые имеют много общего (работают с одинаковыми данными и/или имеют сходные функции). Они изображаются вместе

на диаграмме более низкого (первого) уровня, а на диаграмме нулевого уровня объединяются в один процесс. Выделяются накопители данных, используемые процессами из одной группы.

Декомпозируются сложные процессы и проверяется соответствие различных уровней модели процессов.

Накопители данных описываются посредством структур данных, а процессы нижнего уровня — посредством спецификаций.

Определяются атрибуты сущностей. Выделяются атрибуты-идентификаторы. Проверяются связи, выделяются (при необходимости) зависимые от идентификатора сущности и связи «супертип-подтип».

Проверяется соответствие между описанием структур данных и концептуальной моделью (все элементы данных должны присутствовать на диаграмме в качестве атрибутов).

В настоящее время наблюдается тенденция интеграции разнообразных методов моделирования и анализа систем, проявляющаяся в форме создания интегрированных средств моделирования.

Одним из таких средств является продукт, носящий название ARIS — Architecture of Integrated Information System, разработанный германской фирмой IDS Scheer.

Система ARIS представляет собой комплекс средств анализа и моделирования деятельности предприятия, а также разработки ИС. Ее методическую основу составляет совокупность различных методов моделирования, отражающих разные взгляды на исследуемую систему. Одна и та же модель может разрабатываться с использованием нескольких методов, что позволяет использовать ARIS специалистам с различными теоретическими знаниями и настраивать его на работу с системами, имеющими свою специфику.

Методика моделирования ARIS основывается на разработанной профессором Августом Шером теории построения интегрированных ИС, определяющей принципы визуального отображения всех аспектов функционирования анализируемых компаний.

ARIS поддерживает четыре типа моделей, отражающих различные аспекты исследуемой системы:

• *организационные модели,* представляющие структуру системы — иерархию организационных подразделений, должностей и конкретных лиц, связи между ними, а также территориальную привязку структурных подразделений;

• *функциональные модели,* содержащие иерархию целей, стоящих перед аппаратом управления, с совокупностью деревьев функций, необходимых для достижения поставленных целей;

• *информационные модели,* отражающие структуру информации, необходимой для реализации всей совокупности функций системы;

• *модели управления,* представляющие комплексный взгляд на реализацию бизнес-процессов в рамках системы.

Для построения перечисленных типов моделей используются собственные методы моделирования ARIS, а также известные методы и языки моделирования — ERM, UML, OMT и др.

В процессе моделирования каждый аспект деятельности предприятия сначала рассматривается отдельно, а после детальной проработки всех аспектов строится интегрированная модель, отражающая все связи между различными аспектами.

ARIS не накладывает ограничений на последовательность построения указанных выше типов моделей. Процесс моделирования можно начинать с любого из них в зависимости от конкретных условий и целей, преследуемых разработчиками.

Модели в ARIS представляют собой диаграммы, элементами которых являются разнообразные объекты — «функция», «событие», «структурное подразделение», «документ» и т.п. Между объектами устанавливаются разнообразные связи. Так, между объектами «функция» и «структурное подразделение» могут быть установлены связи следующих видов:

• выполняет;

• принимает решение;

• участвует в выполнении;

• должен быть проинформирован о результатах;

• консультирует исполнителей;

• принимает результаты.

Каждому объекту соответствует определенный набор атрибутов, которые позволяют ввести дополнительную информацию о конкретном объекте. Значения атрибутов могут использоваться при имитационном моделировании или для проведения стоимостного анализа.

Таким образом, по результатам выполнения этого этапа возникает набор взаимосвязанных моделей, представляющих собой исходный материал для дальнейшего анализа.

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 (тем не менее, в новую версию стандарта предполагается включить процесс «Измерение» (measurement)).

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).

При отсутствии адекватной и достоверной оценки невозможно обеспечить четкое планирование и управление проектом. В целом ситуация в данной области, сложившаяся в индустрии информационных технологий, выглядит далеко не блестящей.

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 за 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.

## §2.6. Методы оценки и их классификация

**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. Оценка может быть достаточно детальной в зависимости от глубины аналогий. Слабость модели заключается в том, что степень подобия нового проекта и предыдущих, как правило, не слишком велика.

Самый лучший вариант - это использование накопленных в организации исторических данных, позволяющих сопоставить трудоемкость вашего проекта с

трудоемкостью предыдущих проектов аналогичного размера. Однако это возможно только при следующих условиях:

• в организации аккуратно документируются реальные результаты предыдущих проектов;

• по крайней мере, один из предыдущих проектов (а лучше несколько) имеет аналогичный характер и размер;

• жизненный цикл, используемые методы и средства разработки, квалификация и опыт проектной команды нового проекта также подобны тем, которые имели место в предыдущих проектах.

**Закон Паркинсона.** Согласно этому закону усилия, затраченные на работу, распределяются равномерно по выделенному на проект времени. Здесь критерием для оценки затрат по проекту являются человеческие ресурсы, а не целевая оценка самого программного продукта.

**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.

Описанные методы оценки применимы, если документированы требования будущей системы. В таком случае существует возможность определить функциональные характеристики разрабатываемой системы. Однако во многих проектах оценка затрат проводится только на основе предварительных требований к системе. В этом случае лица, участвующие в оценке стоимости проекта, будут иметь минимум информации для работы. Процедуры анализа требований и создания спецификаций весьма дорогостоящи.

Поэтому менеджерам следует составить смету на их выполнение еще до утверждения бюджета для всего проекта. Практика в этой области такова, что независимые оценки (т.е. выполненные людьми, которые никак не зависят от команды разработчиков) обычно неточны. Единственный способ, позволяющий получить заслуживающую доверия оценку, — это когда компетентная команда — менеджер проекта вместе с ведущими специалистами по созданию архитектуры, разработке и тестированию — выполняют несколько итераций по оценке трудоемкости и анализу чувствительности модели. Для того чтобы проект мог быть успешно выполнен, эта команда должна затем признать свое авторство произведенной оценки трудоемкости.

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:

• размера конечного продукта (для компонентов, написанных вручную), который обычно измеряется числом строк исходного кода или количеством функциональных точек, необходимых и для реализации данной функциональности;

• особенностей процесса, используемого для получения конечного продукта, в частности его способность избегать непроизводительных видов деятельности (переделок, бюрократических проволочек, затрат на взаимодействие);

• возможностей персонала, участвующего в разработке ПО, в особенности его профессионального опыта и знания предметной области проекта;

• среды, которая состоит из инструментов и методов, используемых для эффективного выполнения разработки ПО и автоматизации процесса;

• 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).

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.

• По аналогии. Если в прошлом приходилось иметь дело с подобным проектом, и его оценки известны, то можно, отталкиваясь от них, приблизительно оценить свой проект.

• Путем подсчета размера по определенным алгоритмам на основе исходных данных — требований к системе.

Проблемы оценки 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.

Основными единицами измерения размера ПО являются:

• количество строк кода (LOC — Lines of Code);

• функциональные точки (FP — Function Points).

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.

Преимущества использования LOC в качестве единиц измерения:

• широкое распространение и легкая адаптируемость;

• возможность сопоставления методов измерения размеров и производительности в различных фуппах 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).

Недостатки применения LOC:

• 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;

• профаммисты 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.

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). Рассматриваемый в данном разделе сокращенный вариант методики оценки трудоемкости разработки ПО основан на материалах IFPUG и компании SPR (Software Productivity Research), которая является одним из лидеров в области методов и средств оценки характеристик ПО.

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.

Порядок расчета трудоемкости разработки ПО:

• 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;

• оценка трудоемкости разработки (с использованием различных статистических данных).

В состав функциональных типов (function type) включаются следующие элементы приложений разрабатываемой системы.

***Внутренний логический файл (internal logicalfiky ILF)*** — идентифицируемая совокупность логически взаимосвязанных записей данных, поддерживаемая внутри приложения посредством элементарного процесса.

***Внешний интерфейсный файл (external interface file, Elf)*** **—** идентифицируемая совокупность логически взаимосвязанных записей данных, передаваемых другому приложению или получаемых от него и поддерживаемых вне данного приложения.

**3.** ***Входной элемент приложения (external input, EI)*** — элементарный

процесс, связанный с обработкой входной информации приложения — входного документа или экранной формы. Обрабатываемые данные могут соответствовать одному или более ILF.

**4.** ***Выходной элемент приложения (external output, EO)*** **—** процесс, связанный с обработкой выходной информации приложения — выходного отчета, документа, экранной формы.

**5.** ***Внешний запрос (external query, EQ)*** **~ элементарный процесс,** состоящий из комбинации «запрос/ответ», не связанный с вычислением производных данных или обновлением ILF (базы данных).

Количество функциональных типов по данным (внутренних логических файлов и внешних интерфейсных файлов) определяется на основе диафамм «сущность-связь» (для структурного подхода) и диаграмм классов (для объектно-ориентированного подхода). В последнем случае в расчете участвуют только устойчивые (persistent) классы, или классы-сущности.

Устойчивый класс соответствует ILF (если его объекты обязательно создаются внутри самого приложения) или EIF (если его объекты не создаются внутри самого приложения, а получаются в результате запросов к базе данных).

Для каждого выявленного функционального типа (ILF и EIF) определяется его сложность (низкая, средняя или высокая). Она зависит от количества связанных с этим функциональным типом элементарных данных (data element types, DET) и элементарных записей (record element types, RET), которые в свою очередь определяются следующим образом:

• DET — уникальный идентифицируемый нерекурсивный элемент данных (включая внешние ключи), входящий в ILF или EIF;

• RET — идентифицируемая подфуппа элементов данных, входящая в ILF или EIE На диаграммах «сущность-связь» такая подгруппа обычно представляется в виде сущности-подтипа в связи «супертип-подтип».

Один DET соответствует отдельному атрибуту или связи класса. Количество DET не зависит от количества объектов класса или количества связанных объектов. Если данный класс связан с некоторым другим классом, который обладает явно заданным идентификатором, состоящим более чем из одного атрибута, то для каждого такого атрибута определяется один отдельный DET (а не один DET на всю

связь). Производные атрибуты могут игнорироваться. Повторяющиеся атрибуты одинакового формата рассматриваются как один DET.

Одна RET на диаграмме устойчивых классов соответствует либо абстрактному классу в связи обобщения (generalization), либо классу - «части целого» в композиции, либо классу с рекурсивной связью «родитель-потомок» (афегацией).

Количество транзакционных функциональных типов (входных элементов приложения, выходных элементов приложения и внешних запросов) определяется на основе выявления входных и выходных документов, экранных форм, отчетов, а также по диаграммам классов (в расчете участвуют граничные классы).

Далее для каждого выявленного функционального типа (EI, EO или EQ) определяется его сложность (низкая, средняя или высокая). Она зависит от количества связанных с этим функциональным типом DET, RET и файлов типа ссылок (file type referenced, FTR) ~ ILF или EIF, читаемых или модифицируемых функциональным типом.

**Правила расчета DET для EI:**

• каждое нерекурсивное поле, принадлежащее (поддерживаемое) ILF и обрабатываемое во вводе;

• каждое поле, которое пользователь хотя и не вызывает, но оно через процесс ввода поддерживается в ILF;

• логическое поле, которое физически представляет собой множество полей, но воспринимается пользователем как единый блок информации;

• группа полей, которые появляются в ILF более одного раза, но в связи с особенностями алгоритма их использования воспринимаются как один DET;

• группа полей, которые фиксируют ошибки в процессе обработки или подтверждают, что обработка закончилась успешно;

• действие, которое может быть выполнено во вводе.

**Правила расчета DET для EO:**

• каждое распознаваемое пользователем нерекурсивное поле, участвующее в процессе вывода;

• поле, которое физически отображается в виде нескольких полей его составляющих, но используемое как единый информационный элемент;

• каждый тип метки и каждое значение числового эквивалента при графическом выводе;

• текстовая информация, которая может содержать одно слово, предложение или фразу;

• литералы не могут считаться элементами данных;

• переменные, определяющие номера страниц или генерируемые системой логотипы не являются элементами данных.

**Правила расчета DET для EQ.**

***Правила определения* DET *для вводной части:***

• каждое распознаваемое пользователем нерекурсивное поле, появляющееся во вводной части запроса;

• каждое поле, которое определяет критерий выбора данных;

• группа полей, в которых выдаются сообщения о возникающих ошибках в процессе ввода информации в DET или подтверждающих успешное завершение процесса ввода;

• фуппа полей, которые позволяют выполнять запросы.

***Правила определения OETдля выводной части:***

• каждое распознаваемое пользователем нерекурсивное поле, которое появляется в выводной части запроса;

• логическое поле, которое физически отображается как группа полей, однако воспринимается пользователем как единое поле;

• группа полей, которые в соответствии с методикой обработки могут повторяться в ILF;

• литералы не могут считаться DET.

• колонтитулы или генерируемые системой иконки не могут считаться DET.

Сложность EQ определяется как максимальная из сложностей EI и EO, связанных с данным запросом.

## §2.7. Алгоритмическое моделирование трудоемкости разработки программного обеспечения. Теоретические (математические) модели

Математическое моделирование трудоемкости разработки ПО основано на сопоставлении экспериментальных данных с bормой существующей

математической функции. В начале 1960-х годов Питер Норден из фирмы IBM пришел к выводу, что в проектах по исследованию и разработке может применяться хорошо прогнозируемое распределение трудовых ресурсов, основанное на распределении вероятности, называемом кривой Рэлея (Rayleigh distribution). Позднее, в 1970-х годах Лоуренс Патнэм из компании Quantitative Systems Management применил результаты Нордена к разработке ПО. Используя статистический анализ проектов, Патнэм обнаружил, что взаимосвязь между тремя основными параметрами проекта (размером, временем и трудоемкостью) напоминает функцию Нордена-Рэлея, отражающую распределение трудовых ресурсов проекта в зависимости от времени.

Функция Рэлея моделируется дифференциальным уравнением:

$$dy/dt = 2 * K * a * t * \exp\left(-at^2\right),$$

где *dy/dt* — скорость роста персонала проекта; $t$ — время, прошедшее от начала проекта до изъятия продукта из эксплуатации; $K$ — область под кривой - представляет полную трудоемкость в течение всего жизненного цикла (включая сопровождение), выраженную в человеко-годах; $a$ — константа, которая определяет форму кривой (фактор ускорения) и вычисляется по формуле

$$a = 1/2t_d^2,$$

где $t_d$ - время разработки.

Приняв ряд допущений, Патнэм получил следующее уравнение:

$$E = 0{,}4 * [S/C]^3 * 1/(t_d)^4,$$

где $E$ — трудоемкость разработки ПО, S — размер ПО в LOC, $t_d$ —планируемый срок разработки, C — технологический фактор, учитывающий различные аппаратные ограничения, опыт персонала и характеристики среды программирования. Он определяется на основе хронологических данных по прошлым проектам.

**Статистические (регрессионные) модели.** Статистические модели используют накопленные хронологические данные, чтобы получить значения для коэффициентов модели. Для определения соотношений между параметрами модели и трудоемкостью разработки ПО используется регрессионный анализ. Существуют две формы статистических моделей: линейные и нелинейные.

Линейные статистические модели имеют следующий вид:

$$\text{Трудоемкость} = b_0 + \sum_{i=1}^{n} b_i * x_i.$$

где $x_i$ - факторы, влияющие на трудоемкость, $b_i$ — коэффициенты модели. Линейные модели работают не слишком хорошо, поскольку практика показывает, что соотношения между трудоемкостью и размером ПО нелинейны. По мере роста размера ПО возникает экспоненциальный отрицательный эффект масштаба.

Нелинейные статистические модели имеют следующий вид:

$$\text{Трудоемкость} = A * (\textit{Размер}\ \text{ПО})^b,$$

где $A$ — комбинация факторов, влияющих на трудоемкость; $b$ — экспоненциальный коэффициент масштаба.

Статистические модели просты для понимания, но имеют следующий недостаток: результаты справедливы в основном только для конкретной ситуации. Другой недостаток — при увеличении количества входных параметров количество данных, необходимых для калибровки модели, также возрастает.

# ГЛАВА III. МЕТОДЫ РАЗРАБОТКИ ПРИЛОЖЕНИЙ

Существует две модели построения программ. Первая называется *процедурно-ориентированной* и в ней программа представляется как ряд последовательно выполняемых операций (процедур). Программы, построенные с использованием процедурно-ориентированной модели, можно рассматривать как код, воздействующий на данные. Языки программирования, в которых реализован процедурно-ориентированный подход к построению программ, называются процедурными. До определенного времени процедурно-ориентированный подход успешно применялся при разработке программ. Однако по мере увеличения объема и усложнения программ при использовании данного подхода возникают существенные проблемы.

В *объектно-ориентированной* модели программа рассматривается как совокупность объектов – отдельных фрагментов кода, обеспечивающих выполнение определенных действий и объединяющих данные и методы управления ими. Взаимодействие между объектами производится через определенные интерфейсы. Объектно-ориентированные программы можно характеризовать как данные, управляющие доступом к коду. Объектно-ориентированный подход повышает надежность разрабатываемых программ и обеспечивает возможность многократного использования кода.

Следует отметить, что объектно-ориентированное программирование не является новой концепцией – первые объектно-ориентированные языки (Simula, Smalltalk) появились около 30 лет назад.

Практически все современные алгоритмические языки поддерживают принципы объектно-ориентированного программирования. Наибольшее распространение в последнее время получили три объектно-ориентированных языка: C++, Object Pascal и Visual Basic, ставших дальнейшим развитием давно известных процедурных языков C, Pascal и QuickBasic.

## § 3.1. Основы языка Object Pascal

В данной главе рассматривается язык Object Pascal, используемый в системе визуального программирования Delphi фирмы Borland.

Object Pascal обеспечивает значительно более высокую скорость разработки программ за счет обнаружения на этапе компиляции программы ряда ошибок, которые компилятор языка C++ пропускает без предупреждения. В то же время Object Pascal в полной мере реализует концепции объектно-ориентированного программирования, в чем практически не уступает C++.

Язык Object Pascal является строгим языком, что во многом обусловлено учебным характером его предшественника языка Pascal.

**Структура программы в Object Pascal**

Программа, написанная на языке Object Pascal, состоит из ряда разделов (или блоков). Начало каждого раздела указывается с помощью специальных зарезервированных слов. В общем виде программа Object Pascal имеет следующий вид:

```
// Заголовок программы
Program имя_программы;
// Раздел объявления используемых модулей
Uses
Модуль_1. Модуль_2. Модуль_3:
// Раздел объявления используемых меток
Label
Метка_1, Метка_2;
// Раздел описания констант
Const
идентификатор_константы_1 = значение_1;
идентификатор_константы_2 = значение_2;
идентификатор_константы_3 = выражение_1;
// Раздел описания пользовательских типов.
Type
идентификатор_типа_1 = определение_типа_1;
идентификатор_типа_2 = определение_типа_2;
// Раздел объявления переменных
Var
идентификатор_переменной_1 : определение_ переменной_1;
идентификатор_переменной_2,
идентификатор_переменной_3 : идентификатор_типа_2;
// Раздел объявления процедур и функций программы
Procedure процедура:
// текст процедуры
Function функция_1 : определение_типа_1:
// текст функции
begin
// текст программы
end.
```

**Заголовок программы**

В заголовке после служебного слова Program указывается имя программы. Хотя заголовок программы не является обязательным разделом, при написании программы в среде Delphi имя программы надо указывать. При этом имя основного файла проекта должно совпадать с именем программы, указанным в заголовке.

**Раздел объявления модулей**

Начало раздела объявления модулей указывается с помощью директивы Uses. Имена используемых модулей просто перечисляются через запятую (модули Object Pascal будут обсуждаться далее). Программа может содержать только один блок Uses, причем он должен следовать сразу за заголовком программы.

Разделов объявления меток, типов, констант и переменных может быть несколько, и они могут следовать в любом порядке.

**Раздел объявления меток**

Начало раздела объявления меток указывается с помощью директивы Label. В данном разделе через запятую перечисляются имена используемых в программе меток. Идентификатор метки может состоять из символов латинского алфавита и/или цифр, а также знаков подчеркивания.

**Раздел описания типов**

В Object Pascal существует довольно большое количество стандартных типов и множество типов, описанных в стандартных модулях. Однако при разработке программ, особенно объектно-ориентированных, программисту необходима возможность создавать свои пользовательские типы данных, которые носят название «типы данных, определяемые пользователем». Для описания пользовательских типов используется раздел объявления типов, начинающийся с директивы Type. При создании типа указывается его идентификатор и после знака равенства приводится описание типа. Самым простым способом объявления собственного типа является просто объявление типа, аналогичного уже существующему, например:

```
Type
идентификатор_типа_1 = integer;
```

Наиболее часто программисту приходится определять так называемые структурные типы, назначение которых будет обсуждаться несколько позже.

Идентификатор типа может содержать буквы латинского алфавита, цифры и знак подчеркивания. Первым символом идентификатора обязательно должна быть либо латинская буква, либо символ подчеркивания.

**Раздел переменных**

Начало раздела переменных объявляется с помощью служебного слова Var. В данном разделе должны быть описаны все переменные программы. Компилятор Object Pascal не допускает использования переменных, не объявленных в разделе Var. Объявление переменной, не задействованной в программе, не приводит к ошибке компиляции, однако компилятор выдаст предупреждение о том, что переменная объявлена, но никогда не используется. Например, если объявить неиспользуемую переменную A, то компилятор выдаст следующее сообщение:

[Hint] Projectl.dpr(8): Variable 'A' is declared but never used in 'Projectl'

Число в скобках после имени файла идентифицирует номер строки, в которой описана неиспользуемая переменная.

При объявлении переменной указывается идентификатор и через двоеточие – тип неременной, например:

```
var
идентификатор_переменной_1 : integer:
Идентификаторы переменных одного типа можно перечислять через запятую:
var
идентификатор_переменной_1. идентификатор_переменной_2.
идентификатор_переменной_3 : integer;
```

Для нестандартных типов имя типа должно быть описано в разделе Type, находящемся выше раздела Var, в котором оно используется.

Идентификатор переменной может состоять из символов латинского алфавита, цифр и символов подчеркивания. Первым символом идентификатора обязательно должна быть латинская буква или символ подчеркивания.

**Раздел констант**

Раздел констант содержит объявления констант и начинается с директивы Const. Константа фактически является переменной, значение которой устанавливается не в процессе выполнения программы, а на этапе компиляции. Значение константы не может изменяться программно, при попытке присвоить константе какое-либо значение компилятор выдает сообщение об ошибке. В объявлении константы можно использовать не только конкретные значения, но и выражения (которые могут быть вычислены на стадии компиляции, то есть не

должны содержать переменных и вызовов функций пользователя). При объявлении константы указывается идентификатор и через знак равенства – значение или выражение. Тип константы определяется присваиваемым ей значением или типом результата, получаемого при вычислении выражения.

Идентификатор константы может состоять из символов латинского алфавита, цифр и символов подчеркивания. Первым символом идентификатора обязательно должна быть латинская буква или символ подчеркивания.

Помимо обычных констант, в Object Pascal можно использовать так называемые типизированные константы, или константы-переменные, которые также объявляются в разделе Const. Как и обычным константам, на этапе компиляции константам-переменным также присваивается какое-либо значение. Однако значения констант-переменных можно изменять в процессе выполнения программы, то есть работать с ними как с обычными переменными. Константы-переменные фактически являются обычными переменными, которым при запуске программы присваиваются некоторые значения. Объявление константы-переменной отличается от объявления обычной константы тем, что после идентификатора константы через двоеточие указывается ее тип, а затем после знака равенства – значение. Пример:

```
const
// константа целого типа:
идентификатор_константы_1 = 100;
// константа вещественного типа:
идентификатор_константы 2 = 100.0;
// константа строкового типа:
идентификатор_константы 3 = '100';
// константа, заданная выражением:
идентификатор_константы 4 = (2.5+1)/(2.5-1)
// константа-переменная целого типа:
идентификатор_константы 1 : integer = 20:
// константа-переменная действительного типа:
идентификатор_константы 2 : real = 3.14:
```

**Типы данных в Object Pascal**

Язык Object Pascal отличается строгой типизацией данных. При присваивании переменной какого-либо значения компилятор всегда проверяет соответствие типов. Поэтому все переменные, используемые в программе, обязательно должны быть описаны в разделе объявления переменных.

Типы данных, используемые в Object Pascal, можно разделить на две группы: *простые* и *структурные.*

Несколько особняком стоят *указательные* типы, которые нельзя отнести ни к простым, ни к структурным типам. Указательные типы предназначены для работы с большими структурами данных и памятью.

В последних версиях языка Object Pascal добавлена возможность объявления так называемых *вариантных* типов. Вариантные переменные могут динамически изменять свой тип в процессе выполнения программы.

**Простые типы**

*Простыми* являются типы данных, которыми напрямую может манипулировать процессор (или математический сопроцессор). Простые типы делятся на две группы:*порядковые* и *действительные.* Различие между этими группами заключается в следующем: *порядковые* типы представляют собой счетные множества чисел, лежащих в определенном диапазоне; *действительные* типы не могут быть представлены в виде счетного множества чисел (если не принимать во внимание конечную точность представления действительных чисел при использовании цифровой вычислительной техники).

**Порядковые типы**

*Порядковые* типы подразделяются на целые, символьные, логические, перечисляемые и диапазонные.

*Целые типы.* В переменных целого типа отсутствует дробная часть. В Object Pascal определено довольно большое количество стандартных целых типов, различающихся наличием или отсутствием знака, а также занимаемым объемом памяти. Диапазон значений каждого типа однозначно определяется этими двумя факторами: для *n*-разрядного числа без знака диапазон значений от 0 до 2", для числа со знаком – от $-2^{n-1}$ до $2^{n-1}$ -1.

*Символьные типы.* Классическим методом представления символьной информации является использование 7-разрядной кодировки ASCII (American Standard Code for Information Interchange – Американский стандартный код для обмена информацией). Однако информация обычно хранится в 8-разрядном участке памяти. С помощью 8 бит можно закодировать 256 символов. Кодировка первых 128 символов является стандартной и используется для представления букв латинского алфавита, цифр, символов арифметических действий и ряда

специальных символов, которые не могут быть введены с клавиатуры и применяются в качестве управляющих, например, при выводе информации на принтер. Отнесение управляющих кодов к символьному типу несколько условно, так как многие из них вообще не отображаются в виде каких-либо символов. Следующие 128 символов (с кодами от 128 до 255) называются расширенным набором ASCII. Существует несколько вариантов расширенного набора символов, которые используются для отображения символов русского алфавита, символов псевдографики и т.п. В Delphi применяется расширенный набор символов ANSI.

В последнее время довольно широкое распространение получила 16-разрядная кодировка символьной информации, называемая UNICODE. Данная кодировка позволяет применять гораздо более обширный набор символов и, вероятно, будет получать все большее распространение. В пользу этого говорит и тот факт, что она используется в последних версиях Microsoft Office. Пока это вызывает некоторые проблемы у русскоязычных пользователей, так как старые кириллические шрифты True Type не могут применяться в этих версиях MS Office. Следует отметить, что кодировка первых 256 символов UNICODE совпадает с кодировкой ANSI.

В Object Pascal поддерживается как кодировка ANSI (8-разрядная), так и кодировка UNICODE (16-разрядная). Соответственно определены два символьных типа:

– AnsiChar, или Char, – символьный тип с 8-разрядной кодировкой ANSI;

– WideChar – символьный тип с 16-разрядной кодировкой UNICODE.

**Логические типы**

Переменные логического типа могут принимать только два значения – true (истина) или false (ложь). В классическом языке Pascal был определен только один логический тип – Boolean. Переменные данного типа занимали в памяти 1 байт. В последних версиях языка Object Pascal для совместимости с другими языками определены три логических типа, различающихся занимаемым объемом памяти:

– Boolean, или ByteBool, – 1 байт;

– Word Bool –2 байта;

– Long Bool – 4 байта.

**Перечисляемые типы**

Этот тип определяется перечислением соответствующих идентификаторов, разделяемых запятыми и заключаемых в круглые скобки. Переменные данного типа содержат дискретные значения, представляемые не числами, а *именами:*

```
type
перечисляемый_тип = (first, second, third):
```

В данном примере перечисляемый_тип представляет идентификатор перечисляемого типа, а идентификаторы first, second и third – возможные значения переменной типа перечисляемый_тип. Если в разделе var объявить переменную типа перечисляемый_тип, то этой переменной можно будет присваивать только значения first, second и third. Значения перечисляемых типов не являются числами и им нельзя присваивать числовые значения.

*Диапазонные типы.* Переменные диапазонного типа содержат значения, соответствующие некоторому заданному диапазону любого порядкового типа. Определение диапазонного типа имеет следующий вид:

```
type
диапазонный_тип = минимум..максимум;
```

Диапазонные типы сохраняют все особенности исходного типа и совместимы с ним.

**Действительные типы**

Переменные *действительного* типа используются для представления чисел, имеющих дробную часть. В современной вычислительной технике действительные числа представляются в форме с плавающей точкой. Для работы с ними применяется математический сопроцессор, который сейчас имеется практически на каждом компьютере. В математическом сопроцессоре используются форматы представления чисел с плавающей точкой, стандартизованные Американским институтом инженеров электротехники и электроники (Institute of Electrical and Electronic Engineers, IEEE). Данные форматы различаются объемом занимаемой памяти и количеством значащих цифр мантиссы (с увеличением количества значащих цифр объем занимаемой памяти возрастает).

В Object Pascal используются три формата IEEE: Single, Double и Extended, предназначенные для хранения чисел с разрядностью 32,64 и 80 бит соответственно. В более ранних версиях Object Pascal был определен тип Real, в

котором для представления чисел с плавающей точкой использовались 48 бит. Этот формат был несовместим с форматами математического сопроцессора и требовал дополнительного времени на преобразование в стандартный вид. В последних версиях Object Pascal тип Real аналогичен типу Double, а для совместимости со старыми версиями введен дополнительный тип Real48, использующий 48 бит.

Помимо форматов с плавающей точкой, в Object Pascal определены два вещественных формата с фиксированной точкой: Comp и Currency. Тип Comp представляется с помощью 64 бит и содержит только целые числа в диапазоне от $-2^{63}+1$ до $2^{63}-1$. Однако этот тип относится к вещественным и не совместим с целыми типами. Тип Currency также использует 64 бита, но содержит дробную часть, под которую отводится 4 десятичных знака. Данные форматы применяются для программирования операций с денежными единицами. Использование типов с фиксированной точкой позволяет уменьшить ошибку округления.

**Структурные типы**

Структурные типы данных позволяют использовать переменные, содержащие несколько значений. Элементами структурных типов можно манипулировать и по отдельности, и как единым целым. Элементы структурного типа могут быть как простыми, так и структурными.

Данные в переменной структурного типа по умолчанию выравниваются по границе слова или двойного слова для обеспечения наиболее быстрого доступа к данным. Для отключения выравнивания при объявлении переменной структурного типа применяется ключевое слово packed:

```
type
идентификатор_типа_1 = packed array[0..100] of byte:
var
идентификатор_переменной_1 : идентификатор_типа_1;
идентификатор_переменной_2 : packed arrayt[0..200] of char;
```

В Object Pascal определены следующие структурные типы:

– строки;

– массивы;

– множества;

– записи;

– файлы;

– классы.

**Строковые типы**

В Object Pascal определены три типа для представления текстовых строк.

ShortString. Данный тип аналогичен типу String ранних версий языка Pascal. Его переменные могут содержать строку длиной до 255 символов с фиксированным размером 256 байт. Фактически, тип ShortString представляет собой массив символов, индексированный от 0 до 255. Под хранение символов строки выделяются байты с 1-го по 255-й. Байт с нулевым номером используется для хранения длины строки.

AnsiString. Переменные этого типа могут хранить строку практически неограниченной длины. Максимальное количество символов в такой строке ограничено только адресным пространством компьютера (например, на компьютерах IBM PC число символов в строке может достигать величины $2^{32}$). Переменные данного типа занимают в памяти 4 байта и представляют собой адрес первого символа строки.

WideString. Этот тип аналогичен типу AnsiString, но, в отличие от последнего, символы строки WideChar представляются в кодировке UNICODE, то есть занимают два байта.

**Массивы**

В языке Object Pascal, используемом в системе Delphi, определены два типа массивов – *статические* и *динамические.*

*Статические массивы* идентичны обычным массивам, которые использовались еще в классическом языке Pascal. Данные массивы объявляются с помощью ключевого слова array, после которого в квадратных скобках указывается диапазон изменения индексов массива, а затем через слово of – тип элементов массива. Например, следующее объявление задает переменную типа «массив десяти целых чисел с индексами от 1 до 10»:

```
var
A : array[1..10] of integer;
```

Индексы массива, указываемые в квадратных скобках, фактически являются диапазонным типом. Индексация массивов может быть произвольной, однако общепринято начинать ее с нуля.

Массивы могут быть *многомерными.* В этом случае в объявлении массива в квадратных скобках через запятую указываются несколько диапазонов изменения

индексов. Например, для определения матрицы вещественных чисел размерностью 10 x 5 можно использовать следующее объявление:

```
var
A : array[0..9, 0..4] of double;
```

Для обращения к элементу массива указываются его имя и индекс элемента в квадратных скобках. Если массив многомерный, то в квадратных скобках указывается соответствующее количество индексов через запятую. При этом обращение к элементу массива выполняется как обращение к обычной переменной соответствующего типа.

*Динамический массив* представляет собой указатель на первый элемент массива. При объявлении динамического массива не указывается его размер, то есть диапазон изменения индекса:

```
var
A : array of char;
```

Хотя переменная динамического массива фактически является указателем, работа с динамическим массивом почти идентична работе со статическим массивом. Отличие наблюдается только при выполнении операции присваивания переменных. Например, если объявлены два динамических массива A1 и A2, то после выполнения операции присваивания A1 := A2 обе переменных будут ссылаться на один и тот же фрагмент памяти, то есть фактически будут являться одним массивом. Изменение элементов массива A1 будет приводить к такому же изменению тех же элементов массива A2.

Нумерация элементов динамических массивов всегда начинается с нуля.

Размер динамических массивов определяется во время выполнения программы, для чего используется процедура:

```
SetLength (var S; NewLength : Integer)
```

Здесь S – строка или динамический массив; NewLength – размер строки или массива.

Для строк типа ShortString процедура SetLength просто устанавливает индикатор длины строки (символ с нулевым номером) в значение, заданное параметром NewLength. В этом случае величина NewLength не должна превышать 255.

**Множества**

*Множество* представляет собой набор значений какого-либо порядкового типа. Для объявления переменной типа множества используется ключевое слово set:

```
 type
NumSet = set of AnsiChar;
var
A1 : NumSet;
A2 : set of 0..100;
```

Минимальный и максимальный порядковые номера типа, на основе которого создается множество, должны лежать в пределах от 0 до 255.

Множеству можно присваивать произвольное подмножество:

```
A1:= ['A', 'B', ‘C’, ’D’, 'E'];
```

**Записи**

*Записи* представляют собой структурный тип, объединяющий элементы различных типов. Объявление записи выглядит следующим образом:

```
 type
MyRecType = record
field1 : integer;
field2.field3 : real:
field4 : array[0..4] of char;
end;
var
RecVarl : MyRecType : RecVar2 : record
fieldl : byte:
field2.field3 : extended:
end;
```

Элементы записи называются *полями.* Для обращения к отдельному полю используется идентификатор переменной записи и через точку указывается идентификатор поля: RecVarl.field1. Кроме того, существует специальный оператор with...do, предназначенный для работы с записями. Использование данного оператора выглядит следующим образом:

```
with RecVarl do field1:= 10:
```

Запись может содержать вариантную часть, задаваемую с помощью оператора case. Например, запись следующего вида содержит вариантные поля num_int и byte1, byte2, byte3, byte4:

```
var
RecVar = record
field1 : real;
case byte of
1: num_jnt : integer:
2: bytel.byte2, byte3, byte4 : byte
end;
```

Все варианты занимают в памяти одно и то же место. Например, поле byte1 в рассмотренном примере будет содержать первый байт переменной типа integer, хранящейся в поле num_int, поле byte2 – второй байт и т.д.

**Файлы**

*Файловый тип* данных используется для организации операций файлового ввода-вывода данных. Файловые переменные подразделяются на *типизированные* и *не-типизированные.*

Объявление переменной файлового типа подобно объявлению массива, только без указания числа элементов. При этом вместо слова array используется ключевое слово file. Для *типизированных* файлов после слова file через of указывается тип элементов файла. Этот тип может быть любым, кроме file и class. Объявление*нетипизированной* файловой переменной отличается только тем, что тип элементов файла не указывается. Для работы с текстовыми файлами используется специальный тип Text или TextFile. Пример:

```
var
F1 : file of real; // переменная файла вещественных чисел
F2 : file; // нетипизированная файловая переменная
F2 : TextFile; // переменная текстового файла
```

**Классы**

*Классы* являются структурным типом, похожим на тип record. Однако они позволяют объединять в одной структуре не только данные, но и методы их обработки – процедуры и функции. Более подробно классы обсуждаются в разделе «Основы объектно-ориентированного программирования».

**Указательные типы**

Переменная указательного типа представляет собой адрес, по которому расположено значение переменной. Переменные указательного типа всегда занимают в памяти фиксированный объем – 4 байта (при этом они могут ссылаться на объемы данных любого размера, вплоть до $2^{32}$).

Указатель может ссылаться на данные конкретного типа (типизированный указатель) или просто на область памяти (нетипизированный указатель). Для объявления нетипизированного указателя используется ключевое слово pointer. Для объявления типизированного указателя используется имя соответствующего типа, перед которым ставится знак разыменования (^):

```
var
```

```
P1 : pointer: // нетипизированный указатель
P2 : ^real: // указатель на переменную типа real
P3 : ^MyType: // указатель на переменную типа MyType.
// определенного пользователем
```

Работа с переменными указательного типа имеет ряд особенностей. Необходимо помнить, что переменная-указатель является адресом. Чтобы обратиться к значению, на которое данный указатель ссылается, необходимо использовать перед именем неременной знак разыменования (^). Например, чтобы работать с переменной P1 как с переменной вещественного типа, используется обращение ^P1.

При работе с динамическими переменными им можно присваивать значение nil. Слово nil является зарезервированным и присвоение данного значения указателю означает, что он ни на что не ссылается.

Объявление переменной указательного типа не означает, что при запуске программы для нее будет выделен необходимый объем памяти. Поэтому перед обращением к динамической переменной необходимо выделить для нее память, а но окончании работы с указателем эту память освободить. Для этого используются специальные процедуры языка Object Pascal, перечисленные ниже.

procedure New(var P : Pointer);

Выделяет необходимый объем памяти для хранения значения переменной, на которую ссылается указатель P. Объем выделяемой памяти определятся типом указателя. Если недостаточно свободной памяти, генерируется исключительная ситуация EOutOfMemory.

procedure Dispose(var P : Pointer);

Освобождает область памяти, на которую ссылается указатель P.

procedure GetMem(var P: Pointer: Size: Integer);

Выделяет Size байт для размещения данных, на которые ссылается указатель P. Если памяти недостаточно, генерируется исключительная ситуация EOutOfMemory.

procedure FreeMem(var P: Pointer [: Size: Integer]):

Освобождает область памяти, на которую ссылался указатель P. Необязательный параметр Size определяет, сколько байтов памяти будет освобождено. Если параметр Size указывается, то его значение должно точно соответствовать объему памяти, выделенному переменной P процедурой GetMem.

procedure FreeAndNil(var P):

Освобождает память, на которую ссылается указатель P, и присваивает указателю P значение nil.

Для типизированных указателей выделение и освобождение памяти следует производить с помощью процедур New и Dispose.

**Вариантные типы**

*Вариантные типы* используются в тех случаях, когда необходимо передавать значение, тип которого заранее неизвестен.

Для объявления переменной вариантного типа используется зарезервированное слово variant. Под переменную данного типа отводится 16 байт. В них содержится код типа, а также значение переменной или указатель на значение.

Тип variant позволяет хранить все простые типы данных (кроме Int64), а также динамические массивы.

В Object Pascal определены два особых значения переменных типа variant:

– Unassigned – означает, что переменной пока не присвоено значение какого бы то ни было типа, то есть к вариантной переменной ни разу не обращались (данное значение присваивается вариантной переменной при ее инициализации);

– Null – означает, что вариантная переменная содержит данные неизвестного типа.

Для получения информации о типе данных, хранимых в вариантной переменной, используется специальная функция VarType. Коды, возвращаемые данной функцией, приведены в табл. 7.3.

Для работы с вариантными массивами в Object Pascal существует ряд специальных функций:

```
function VarArrayCreate (const Bounds:array of Integer:
VarType:Integer): Variant:
```

Эта функция предназначена для создания вариантного массива с границами, заданными параметром Bounds, и типом элементов, заданных параметром VarType. Значение VarType должно быть одним из кодов, приведенных в табл. 7.3. Например, следующая строка создает массив вещественных чисел, состоящий из 10 элементов (переменная A должна быть описана как variant):

```
A : = VarArrayCreate([0,9],varDouble):
```

Элементы массива могут иметь вариантный тип и, следовательно, содержать данные различных типов. Например, пусть имеется вызов:

```
A : = VarArrayCreate ([0.4], varVariant):
```

После выполнения этого вызова будут допустимы следующие присваивания:

```
A[0]: = 1;
A[1]: = 1234.5678;
A[2]: = 'Hello world';
A[3]: = True;
 function VarArrayOf (const Values:array of variant):Variant;
```

Данная функция возвращает одномерный вариантный массив с элементами, представленными в переменной Values. Нижний индекс массива всегда равен О, а верхний определяется количеством элементов массива, переданного в Values. Например, после выполнения следующего оператора переменная А будет являться вариантным массивом, состоящим из 3 элементов вариантного типа:

```
A: = VarArrayOf([10. 3.14, 'Text']);
procedure VarArrayRedim (var A: Variant; HighBound: Integer);
```

Применяется для изменения верхнего предела HighBound вариантного массива А. Значения элементов массива, определенные перед изменением предела, сохраняются.

```
function VarArrayLock (var A: Variant): Pointer;
```

Используется для фиксирования вариантного массива А и возвращения указателя на первый элемент массива. Пока массив зафиксирован, его размерность не может быть изменена, и вызов функции VarArrayRedim будет безрезультатным. Для отмены фиксации используется процедура: VarArrayUnlock(var A; Variant)

## § 3.2. Операторы языка Object Pascal

Операторы предназначены для контроля за порядком вычисления выражений и количеством вычислений. Операторы, используемые в языке Object Pascal, условно можно разделить на две группы: простые операторы и структурные операторы. К простым операторам следует отнести операторы присваивания и безусловного перехода. Группу структурных операторов составляют условные операторы, операторы циклов и составной оператор.

**Оператор присваивания**

*Оператор присваивания* предназначен для изменения значения переменных. Результат выражения, расположенного справа от оператора присваивания (:=),

заносится в переменную, расположенную слева. При использовании оператора присваивания всегда необходимо строго соблюдать правила соответствия типов.

**Оператор безусловного перехода**

*Оператор безусловного перехода* goto передает управление оператору, помеченному указанной меткой:

goto label5;

Метка должна быть описана в разделе label того блока, в котором выполняется оператор goto. В тексте программы идентификатор метки отделяется двоеточием от оператора, на который метка указывает:

label5: A: = expression;

**Условный оператор**

*Условный оператор* обеспечивает выполнение или невыполнение определенного оператора в зависимости от выполнения или невыполнения заданного условия. В Object Pascal определены два условных оператора: if и case...of.

**Оператор if**

Оператор if обеспечивает выбор из двух вариантов:

```
if логическое_выражение
then оператор_1 // выполняется, если
// логическое_выражение = true
else оператор_2; // выполняется, если
// логическое_выражение = false
```

В операторе if часть else необязательна:

```
if логическое_выражение
then оператор: // выполняется, если значение
// логическое_выражение = true
```

В Object Pascal определены следующие операции сравнения двух значений:

A > B – больше;
A < B – меньше;
A >= B – больше или равно;
A <= B – меньше или равно;
A = B – равно;
A <> B – не равно.

В логическом выражении, приводимом после слова if, допускается использование следующих логических операций:

not – инверсия;
and – логическое умножение;
or – логическое сложение;
xor – исключающее или.

Например, логическое выражение может быть сформулировано следующим образом:

```
(A = 0) and (B < 0) or (C >=0)
```

**Оператор case**

Оператор case...of применяется для выполнения одного оператора из нескольких в зависимости от значения переменной (или результата вычисления выражения), указываемой между словами case и of. Данная переменная называется *селектором.* Селектор обязательно должен иметь порядковый тип. После описания селектора следует список операторов, каждому из которых предшествует одна или несколько меток, отделяемых от оператора двоеточием. Заканчивается оператор ключевым словом end. Метки представляют значения, которые может принимать селектор. При обращении к оператору case выполняется оператор, метка которого соответствует значению селектора:

```
case селектор of
1 : оператор_1: // выполняется, если селектор = 1
2.3 : оператор_2 // выполняется, если селектор = 2
// или если селектор = 3 end:
```

Метки в операторе case могут задаваться в виде диапазонов. Оператор case может иметь блок else. Оператор, расположенный после else, выполняется в том случае, если значение селектора не соответствует ни одной из указанных меток.

```
case селектор of
'A' : оператор_1; // выполняется, если селектор = 'A'
'D'..'H' : оператор_3: // выполняется, если селектор лежит
// в диапазоне от 'D' до 'H'
else оператор_4; // выполняется во всех остальных
// случаях
end;
```

**Операторы цикла**

*Операторы цикла* обеспечивают возможность многократного повторения одного или нескольких операторов. В Object Pascal определены три оператора цикла: for... do, while...do и repeat...until.

**Цикл for**

Цикл for...do предназначен для выполнения строго заданного количества повторений. Между словами for и do для некоторой переменной целого типа (называемой переменной цикла) указывается диапазон изменения:

```
for i:= начальное_значение to конечное_значение do оператор;
for i:= начальное_значение downto конечное_значение do оператор;
```

При первом обращении к оператору for переменная цикла всегда принимает значение начальное_значение.

Если при указании диапазона изменения переменной цикла используется зарезервированное слово to, то для того чтобы оператор, следующий после слова do, был выполнен, необходимо, чтобы текущее значение переменной цикла было не больше конечное_значение. При каждом выполнении оператора, следующего за do, значение переменной цикла увеличивается на 1. Если используется ключевое слово downto, то оператор, указанный после do, выполняется в том случае, если началь-ное_значение не меньше, чем конечное_значение; при каждом проходе значение переменной цикла уменьшается на 1.

Цикл for используется в тех случаях, когда заранее известно, сколько раз надо выполнить оператор.

**Цикл while...do**

Оператор while...do обеспечивает выход из цикла по условию. Условие указывается между словами while и do и представляет собой логическое выражение:

while логическое_выражение do оператор;

Оператор, следующий за do, выполняется до тех пор, пока результат логического выражения не станет равным false. Значение логического выражения вычисляется в первую очередь, и если оно изначально равно false, то оператор, указанный после do, не будет выполнен ни разу.

**Цикл repeat...until**

Конструкция repeat... until также обеспечивает выход из цикла по условию:

```
repeat
оператор_1;
оператор_2;
оператор_n;
until логическое_выражение:
```

Операторы, расположенные между ключевыми словами repeat и until, будут выполняться до тех пор, пока логическое выражение, указанное после until, не примет значение true.

В отличие от цикла while, логическое выражение, обусловливающее выход из цикла, вычисляется после выполнения операторов тела цикла. Поэтому, даже если оно изначально равно true, операторы все равно будут выполнены один раз.

### Составной оператор

*Составной оператор* позволяет интерпретировать группу операторов как один оператор. Это необходимо для работы практически со всеми рассмотренными операторами. Например, условные операторы и операторы цикла for и while обеспечивают переход по условию или выполнение цикла только для одного оператора. Если же заключить группу операторов между словами begin и end, то они будут восприниматься как один оператор:

```
 …
if a<b
then begin
c: = выражение_1;
d: = выражение_2;
end;
else begin
c: = выражение_3:
d: = выражение 4;
end;
…
case i of
1: begin оператор_1;
оператор_2;
end;
2.3: оператор_3;
end;
…
for i: = 0 to 10 do begin
a: = выражение_1
Ь: = выражение_2;
end;
…
while a > b do
begin
a: = выражение_1;
b: = выражение_2;
end;
```

### Процедуры и функции

Процедуры и функции представляют собой блоки программного кода, имеющие точно такую же структуру, как и программа (единственное отличие заключается в том, что процедуры и функции не могут содержать раздел uses).

### Процедуры

Ниже приведен пример программной реализации процедуры:

```
procedure proc_id(<список параметров>): // заголовок процедуры
const // раздел описания локальных констант
const1 = valuel;
type // раздел описания локальных типов
type_jd1 = type_def1;
var // раздел описания локальных переменных
var_id1 : type_id1:
var_id2, var_id3 : type_def2;
```

```
begin
…// текст процедуры
end:
```

Заголовок процедуры состоит из зарезервированного слова procedure, идентификатора процедуры и списка параметров, заключенного в круглые скобки (список параметров не обязателен, можно создавать процедуры без параметров). Параметры, указываемые в заголовке процедуры, называются *формальными* и предназначены для обмена данными между процедурой и основной программой. В списке указывается идентификатор параметра и через двоеточие – его тип. Друг от друга параметры отделяются точкой с запятой:

```
procedure proc_jd(param1 : integer : param2:real);
```

Отметим основные свойства процедуры.

– Количество передаваемых параметров не ограничено.

– Внутри процедуры формальные параметры представляют собой обычные переменные или константы.

– Вызов процедуры осуществляется с помощью оператора вызова, состоящего из идентификатора процедуры и списка параметров, перечисляемых через запятую:

```
…
proc_id(A, B);
…
```

– Параметры, указываемые при вызове процедуры, называются *фактическими.* Они представляют собой переменные или константы, описанные в программе.

– Передача параметров производится через стек и может выполняться либо по значению, либо но ссылке.

**Функции**

*Функции* отличаются от процедур только тем, что их идентификатор возвращает некоторое значение. Поэтому при описании функции необходимо через двоеточие указать тип возвращаемого значения:

```
function MyFunc(A : integer):single;
begin
…
end;
```

Идентификатор функции может быть использован в выражении;

```
var_id1: = 10*MyFunc(var_id2)/var_id3;
```

В теле функции для возвращения значения используется либо идентификатор функции, либо неявно определенный идентификатор result. Например, функция, вычисляющая величину, обратную заданному параметру, может быть описана двумя эквивалентными способами.

Первый способ:

```
function reverse(a : double):double;
begin
reverse: = 1/double
end;
```

Второй способ:

```
function reverse(a : double):double;
begin
result: = 1/double
end;
```

С переменной result внутри функции можно работать как с обычной переменной.

## § 3.3. Модули Object Pascal

При разработке программ в среде Delphi широко используются так называемые *модули.* Они позволяют объединить логически связанные типы данных, переменные, процедуры и функции в один программный блок. Причем все идентификаторы, описанные в модуле, могут быть использованы в других программных блоках. Фактически, модуль представляет собой нечто вроде библиотеки подпрограмм, типов данных, переменных и констант. Для использования идентификаторов, описанных в модуле в программе (или другом модуле), достаточно объявить имя модуля в разделе uses.

Структура модуля Object Pascal имеет следующий вид:

```
unit имя_модуля; // заголовок модуля
interface // блок интерфейса
uses
модуль_1. модуль_2;
const
константа_1 = значение_1;
константа_2 = выражение_1;
type
тип_1 = определение_типа_1;
var
идентификатор_переменной_1 : определение_типа_1;
идентификатор_переменной_2 : определение_типа_2;
procedure идентификатор_процедуры_1;
function идентификатор_функции_1 : определение_типа_3;
implementation // блок реализации
uses
модуль_3, модуль_4;
```

```
const
константа_3 = значение_2;
type
тип_2 = определение_типа_4;
var
идентификатор_переменной_3,
идентификатор_переменной_4 : определение_типа_5;
procedure процедура_1;
begin
…
end;
function функция_1 : определение_типа_6;
begin
…
end;
procedure идентификатор_процедуры_1;
begin
…
end;
function идентификатор_функции_1 : определение_типа_7;
begin
…
end;
initialization // блок инициализации
оператор_1;
оператор_2:
finalization // блок завершения
оператор_3;
оператор_4;
end.
```

В приведенном примере модуля можно выделить структурные единицы.

*Заголовок модуля* состоит из ключевого слова unit и имени модуля, которое обязательно должно совпадать с именем файла. В отличие от заголовка программы, заголовок модуля является обязательным.

*Блок интерфейса* содержит описание констант, типов, переменных, процедур и функций, которые будут доступны в других программах и модулях. Интерфейсная часть начинается с зарезервированного слова interface и всегда должна следовать сразу за заголовком. Блок интерфейса может содержать раздел uses, который должен быть описан в первую очередь, до всех других объявлений. В блоке интерфейса описываются только заголовки процедур и функций. Их текст приводится в разделе реализации.

*Блок реализации* начинается с зарезервированного слова implementation и может содержать объявления констант, типов, переменных, процедур и функций. Все эти объявления доступны только в данном модуле. Кроме того, в разделе implementation размещается реализация всех процедур и функций, заголовки

которых объявлены в блоке интерфейса. Все идентификаторы, описанные в разделе интерфейса, доступны в разделе реализации.

## § 3.4. Основы объектно-ориентированного программирования

Концепция объектно-ориентированного программирования позволяет упростить разработку сложных программ и повысить их надежность. Однако объектно-ориентированная модель построения программ принципиально отличается от процедурно-ориентированной. Ее основу составляет не алгоритм, а иерархия объектов, из которых состоит программа (хотя разработка отдельных объектов все равно требует алгоритмического подхода). Поэтому для эффективного использования ООП требуется иной взгляд на проблему, иначе даже применение объектно-ориентированных языков не поможет добиться объектно-ориентированного стиля программирования.

В данном разделе описываются основные принципы ООП, которые иллюстрируются примерами на языке Object Pascal.

### Основные понятия и отличительные черты ООП

В основе объектно-ориентированного программирования лежит идея объединения данных и действий, которые производятся над этими данными, в одной структуре.

Каждая используемая в программе переменная имеет смысл только тогда, когда может принимать какие-либо значения. Множество значений, которые может принимать переменная, является определяющей характеристикой переменной и называется ее *типом.* Тип переменной, в свою очередь, определяет набор операций, которые можно к ней применять.

В объектно-ориентированном программировании базовыми единицами программ и данных являются *классы.*

### Классы

*Класс* – это структура данных, которая может содержать в своем составе переменные, функции и процедуры. Переменные, в зависимости от назначения, называются*полями* (field) или *свойствами.* Процедуры и функции, входящие в состав класса, называются *методами.*

В Object Pascal определен структурный тип class. Объявление типа class похоже на объявление типа record, однако в нем могут содержаться не только поля-переменные, но и методы. Кроме того, в объявлении класса используется ряд специальных зарезервированных слов, определяющих область видимости нолей и методов. В отличие от всех остальных типов, тип class обязательно должен быть описан как пользовательский тип в разделе type, например:

```
type
TMyClass = class
field1 : type_definition1:
field2 : type_definition2;
procedure method1;
function method2 : type_definition3;
end;
```

Затем в разделе var может быть объявлена переменная объектного типа:

```
var
Object1 : TMyClass;
```

При объявлении класса вначале описываются поля, а затем – методы. Поля класса являются переменными, входящими в состав его структуры. Они предназначены для использования внутри класса. В описании объектного типа присутствуют только заголовки методов. Сами методы описываются в разделе реализации того модуля, в котором объявляется новый объектный тип.

**Объекты**

*Объектом,* или *экземпляром класса,* называется переменная объектного типа.

Чтобы объект мог обмениваться данными с другими объектами, используются свойства. *Свойства объекта* определяют его состояние. Технология ООП запрещает работать с объектом иначе, чем через методы, то есть изменение состояния объекта производится только через вызов методов этого объекта. Этим существенно ограничивается возможность приведения объекта в недопустимое состояние и/или несанкционированного разрушения объекта.

Взаимодействие между объектами осуществляется с помощью сообщений. Объект может посылать сообщения другим объектам и принимать сообщения от них.*Сообщение* является совокупностью данных определенного типа, передаваемых объектом-отправителем объекту-получателю, имя которого указывается в сообщении. Получатель реагирует на сообщение выполнением

некоторого метода, имя которого также может быть указано в сообщении, или никак не реагирует на него:

Объект можно интерпретировать как модель некоторого реального объекта или процесса, которая обладает следующими свойствами:

– поддается хранению и обработке;

– способна взаимодействовать с другими объектами и вычислительной средой, посылая сообщения и реагируя на принимаемые сообщения.

В системе ООП совокупность объектов образует среду, в которой вычисления выполняются путем обмена сообщениями между объектами.

# ЗАКЛЮЧЕНИЕ

Предложенный в данной работе набор понятий является достаточно абстрактным для того, чтобы сформулировать КИС вне привязки к конкретным программно-аппаратным решениям и в то же время достаточно конкретным для определения полезной функциональности (сервисы и приложения как средство решения задач пользователя КИС) и эксплуатационных характеристик (свойства и службы) проектируемой системы.

Изложенные выше понятия и принципы вполне конкретны. Будучи принятыми в качестве основополагающих при построении информационной системы, они выливаются в конкретные организационные шаги и технические действия, которые в совокупности можно охарактеризовать как рациональные технологии. Будучи последовательно проведенными в жизнь, они с высокой гарантией приведут к желаемому результату.

Особое значение в контексте предложенного в работе подхода приобретают:

Серверные продукты и технологии, качество которых в основном предопределяет качество проектируемой КИС.

Готовые прикладные решения (специализированные приложения), определяющие прикладную функциональность КИС.

Компании, поставляющие большой набор серверных продуктов и технологий, в совокупности с интегрированными с ними готовыми прикладными решениями (специализированными приложениями).

## Литература

1. *Бек К.* Экстремальное программирование: Пер. с англ. — СПб.: Питер, 2002.

2. *Боггс У., Боггс М.* UML и Rational Rose: Пер. с англ. - М.: ЛОРИ, 2000.

3. *Боэм Б.У.* Инженерное проектирование программного обеспечения: Пер. с англ. — М.: Радио и связь, 1985.

4. *Брауде Э. Дж.* Технология разработки профаммного обеспечения: Пер. с англ. - СПб: Питер, 2004.

5. *Брукс Ф.* Мифический человеко-месяц или как создаются программные системы: Пер. с англ. - СПб.: Символ-Плюс, 1999.

6. *Буч Г.* Объектно-ориентированный анализ и проектирование с примерами приложений на C++. — 2-е изд.: Пер. с англ. — М.: Издательство Бином, СПб.: Невский диалект, 1999.

7. *Буч Г. и др.* Язык UML. Руководство пользователя / Г. Буч, Дж. Рамбо, А. Джекобсон: Пер. с англ. — М.: ДМК, 2000.

8. *Вендров А.М.* CASE-технологии. Современные методы и средства проектирования информационных систем. - М.: Финансы и статистика, 1998.

9. *Вендров А.М.* Практикум по проектированию программного обеспечения экономических информационных систем: Учеб. пособие. — М.: Финансы и статистика, 2002.

10. *Вигерс К.* Разработка требований к программному обеспечению: Пер. с англ. — М.: Русская редакция, 2004.

11. *Гамма Э. и др.* Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес: Пер. с англ. — СПб: Питер, 2001.

12. *Гома Х.* UML. Проектирование систем реального времени, распределенных и параллельных приложений: Пер. с англ. — М.: ДМК, 2002.

13. *Йордон Эд.* Путь камикадзе: Пер. с англ. - М.: ЛОРИ, 2001.

14. *Каляное Г.Н.* Консалтинг при автоматизации предприятий. — М.: СИНТЕГ, 1997. (Серия «Информатизация России на пороге XXI века»)

15. *Коберн А.* Быстрая разработка программного обеспечения: Пер. с англ. - М.: ЛОРИ, 2002.

16. *Кватрани Т.* Визуальное моделирование с помощью Rational Rose 2002 и UML: Пер. с англ. - М.: Вильяме, 2003.

17. *Коберн Л,* Современные методы описания функциональных требований к системам: Пер. с англ. — М.: ЛОРИ, 2002.

18. *Конноли* Г., *Беге К.* Базы данных: проектирование, реализация и сопровождение. Теория и практика. — 3-е изд.: Пер. с англ. - М.: Вильяме, 2003.

19. *Крантен Ф.* Введение в Rational Unified Process: Пер. с англ. - М.: Вильяме, 2002.

20. *Ларман К.* Применение UML и шаблонов проектирования. — 2-е изд.: Пер. с англ. — М.: Вильяме, 2002.

21. *ЛеффингуэллД,у УидригД,* Принципы работы с требованиями к программному обеспечению. Унифицированный подход: Пер. с англ. - М.: Вильяме, 2002.

22. *Липаев В,В,* Документирование и управление конфигурацией программных средств. Методы и стандарты. — М.: СИНТЕГ, 1998.

23. *Липаев В.В.* Системное проектирование сложных программных средств для информационных систем. — 2-е изд. — М,: СИНТЕГ, 2002.

24. *Маклаков СВ.* BPwin и ERwin. CASE-средства разработки информационных систем. - М.: Диалог-МИФИ, 1999.

25. *Маклаков СВ.* Моделирование бизнес процессов с BPwin 4.0. - М.: Диалог-МИФИ, 2002.

26. *Марка Д.А., МакГоуэн К.* Методология структурного анализа и проектирования. — М.: МетаТехнология, 1993.

27. *Мацяшек Л.* Анализ требований и проектирование систем. Разработка информационных систем с использованием UML: Пер. с англ. - М.: Вильяме, 2002.

28. *Мюллер Р.* Базы данных и UML. Проектирование: Пер. с англ. -М.: ЛОРИ, 2002.

29. *Нейбург Э. Дж., Максимчук RA.* Проектирование баз данных с помощью UML: Пер. с англ. — М.: Вильяме, 2002.

30. *Одинцов И.* Профессиональное программирование. Системный подход. — СПб.: БХВ-Петербург, 2002.

31. *Орлов СА.* Технологии разработки программного обеспечения. - СПб.: Питер, 2002.

32. Оценка и аттестация зрелости процессов создания и сопровождения программных средств и информационных систем (ISO/IEC TR 15504-CMM): Пер. с англ. А.С. Агапова и др. - М.: Книга и бизнес, 2001.

33. *Палмер СР., Фелсинг Дж.М.* Практическое руководство по функционально-ориентированной разработке ПО: Пер. с англ. — М.: Вильяме, 2002.

34. Принципы проектирования и разработки программного обеспечения. Учебный курс MCSD. ~ 2-е изд.: Пер. с англ. — М.: Русская редакция, 2002.

35. *Рамбо Дж. и др.* UML. Специальный справочник/Дж. Рамбо, Г. Буч, А. Якобсон: Пер. с англ. - СПб: Питер, 2002.

36. *РозенбергД., Скотт К.* Применение объектно-ориентированного моделирования с использованием UML и анализ прецедентов: Пер. с англ. - М.: ДМК, 2002.

37. *Ройс У.* Управление проектами по созданию программного обеспечения: Пер. с англ. — М.: ЛОРИ, 2002.

38. *Соммервилл И.* Инженерия программного обеспечения. — 6-е изд.: Пер. с англ. - М.: Вильяме, 2002.

39. *Фатрелл Р. и др.* Управление программными проектами: достижение оптимального качества при минимуме затрат / Р. Фатрелл, Д. Шафер, Л. Шафер: Пер. с англ. — М.: Вильяме, 2003.

40. *Фаулер М., Скотт К.* UML в кратком изложении. Применение стандартного языка объектного моделирования: Пер. с англ. — М.: Мир, 1999.

41. *Черемных СВ. и др.* Структурный анализ систем: IDEF-технологии / С.В.Черемных, И.О. Семенов, В.С. Ручкин. — М.: Финансы и статистика, 2001.

42. *Черемных СВ. и др.* Моделирование и анализ систем. IDEF-технологии: практикум/С.В.Черемных, И.О. Семенов, В.С. Ручкин.- М.: Финансы и статистика, 2002.

43. *Элиенс А.* Принципы объектно-ориентированной разработки программ. — 2-е изд.: Пер. с англ. — М.: Вильяме, 2002.

44. *Якобсон А. и др.* Унифицированный процесс разработки программного обеспечения / А. Якобсон, Г. Буч, Дж. Рамбо: Пер. с англ. - СПб.: Питер, 2002.

# Xülasə

Bu işin yazılmasının məqsədi korporativ informasiya sistemlərinin tədqiqatıdır ki, onlardan strateji alternativlərin təsvirinin və razılaşma və ya təşkilatlar üçün kompromisslər sahəsinin seçilməsinin əlverişli mexanizmi kimi istifadə edilə bilər.

## Summary

The purpose, for which this paper was written, is to study the possibilities of corporative information systems, which can be used as a convenient mechanism for describing the strategic alternatives and taking one of the areas of agreement or even the very interesting from compromises for organizations.