Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Оренбургский государственный университет»

Н.А. Соловьев, Е.Н. Чернопрудова

СИСТЕМЫ АВТОМАТИЗАЦИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Рекомендовано Ученым советом федерального государственного бюджетного образовательного учреждения высшего профессионального образования «Оренбургский государственный университет» в качестве учебного пособия для студентов, обучающихся по программам высшего профессионального образования направления 230100.68 Информатика и вычислительная техника, 231000.68 Программная инженерия

Оренбург
2012

Николай Соловьев

Системы автоматизации разработки программного обеспечения

«БИБКОМ»

2012

УДК 681.3(075.8)
ББК 32.973.26-018

Соловьев Н. А.

Системы автоматизации разработки программного обеспечения / Н. А. Соловьев — «БИБКОМ», 2012

В учебном пособии рассмотрены методологические основы построения систем автоматизации разработки программного обеспечения на основе универсального языка моделирования UML. Теоретический материал дополнен примерами автоматизированного проектирования программной системы с аналитическим приложением на основе методов теории статистических решений, вопросами для проверки усвоения материала.

УДК 681.3(075.8)
ББК 32.973.26-018

Содержание

Соловьев Н. А., Чернопрудова Е. Н. Системы автоматизации разработки программного обеспечения

Введение

Создание автоматизированных информационных систем (АИС) – весьма сложная и трудоемкая задача в связи с тем, что современное программное обеспечение (ПО) данного класса составляет сотни тысяч операторов. Будущий специалист в области разработки ПО должен иметь представление о современных методах автоматизации анализа, проектирования, реализации и тестирования АИС, т.е. ориентироваться в современных подходах к технологиям программирования.

Теоретической основой построения систем автоматизации проектирования ПО (САПР ПО) являются методы технологии разработки ПО, автоматизация которых является предметом изучения настоящего учебного пособия. Изложение материала строится в соответствии с основными этапами жизненного цикла ПО.

Основой изложенного материала стал учебник «Технология программирования», разработанного в МГТУ им. Н. Баумана профессором Г.С. Ивановой, и допущенного Министерством образования и науки Российской Федерации (Минобрнауки РФ) для студентов ВУЗов, обучающихся по направлению 2301000 – Информатика и вычислительная техника. Материал доработан в процессе многолетней апробации на кафедре программного обеспечения вычислительной техники и автоматизированных систем «Оренбургский государственный университет».

1 Методология автоматизации разработки программного обеспечения

1.1 Актуальность автоматизации разработки программного обеспечения

Производство программного обеспечения сегодня – крупнейшая отрасль мировой экономики, в которой занято около 3-х млн. специалистов. Еще несколько млн. человек напрямую зависят от качества корпоративных автоматизированных информационных систем (АИС).

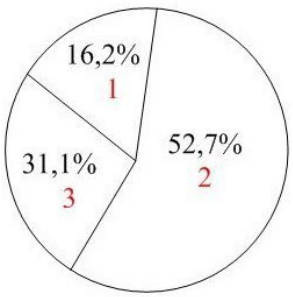
Поэтому состояние отрасли напрямую определяет благополучие специалистов-разработчиков программного обеспечения (ПО).

1.1.1 Кризис программной инженерии, его причины и пути преодоления

Проектирование корпоративных АИС – логически сложная, трудоемкая и длительная работа, требующая высокой квалификации участвующих в ней специалистов. Однако до настоящего времени проектирования АИС нередко осуществляется на интуитивном уровне неформализуемыми методами, включающими в себя элементы искусства, практический опыт и дорогостоящие экспериментальные проверки качества функционирования системы. Кроме того, в процессе создания и функционирования АИС информационные потребности пользователей постоянно изменяются или уточняются, что еще более осложняет разработку и сопровождение таких систем.

В конце XX – го века в программной инженерии сложилось критическая ситуация, неразрешенная до сих пор. Кризис выражается в том, что большие проекты ПО стали выполняться с отставанием графика и со значительным превышением расходов, а разработанный продукт не обладал требуемыми функциональными возможностями или производительностью, что не устраивает потребителей. Так, например, в 1995 г. компания Standish Group проанализировала работу 364 американских корпораций по итогам выполнения более 23 000 проектов, связанных с разработкой ПО.

Результаты анализа, представленные на рисунке 1.1, оказались удручающими.



1 – доля проектов, завершившихся в срок без превышения бюджета;
2 – доля проектов, завершившихся с опозданием, превышением расходов, функционал которых не реализован в полном объеме;
3 – доля аннулированных проектов

Рисунок 1.1 – Результаты анализа проектов в области программной инженерии

Причины кризиса:
– нечеткая и неполная формулировка требований к ПО;

– недостаточное вовлечение пользователей в работу над проектом;

– отсутствие необходимых ресурсов и неудовлетворительное планирование;

– частое изменение требований спецификаций;

– новизна используемой технологии для организации;

– отсутствие грамотного управления проектом.

В конце 20 – го века утвердилось понимание необходимости перехода от кустарных к индустриальным технологиям создания ПО, к созданию совокупности инженерных методов и средств разработки программных продуктов, объединенных общим названием «программная инженерия» (software engineering). Тогда же появилось первое издание, посвященное программной инженерии – IEEE Transaction on Software Engineering.

В основе программной инженерии лежит фундаментальная идея: *разработка ПО является формальным процессом и, следовательно, его можно автоматизировать.*

Таким образом, автоматизация разработки программного обеспечения является **актуальной** инженерной задачей в предметной области специалистов ПОВТАС.

1.1.2 Тенденции развития современных автоматизированных информационных систем

Предметной областью специалистов ПОВТАС являются АИС. Как отмечал Фредерик Брукс, руководитель проекта операционной системы OS/360, самым существенным свойством программных систем (ПС), к классу которых относится АИС, является их сложность. Благодаря уникальности и несхожести своих составных частей АИС принципиально отличается от технических систем, в которых преобладают повторяющиеся элементы.

Тенденциями развития АИС в современных условиях становятся:

– сложность описания (большое количество функций, процессов, элементов данных и сложные взаимосвязи между ними);

– наличие совокупности тесно взаимодействующих компонентов, имеющих локальные задачи и цели функционирования (например, традиционных приложений, связанных с обработкой транзакций, приложений аналитической обработки данных – поддержки принятия решений);

– отсутствие полных аналогов корпоративных АИС, ограничивающие возможность использования типовых проектных решений;

– необходимость интеграции существующих и вновь разрабатываемых приложений;

– функционирование в неоднородной среде на нескольких аппаратных платформах;

– разобщенность и разнородность групп разработчиков по уровню квалификации и сложившимся традициям использования тех или иных инструментальных средств;

– значительная временная протяженность проекта.

Нелинейность роста сложности ПС при увеличении размера системы становится причиной затруднений, возникающих в процессе общения между разработчиками, понимания ими всех возможных состояний программ, ведет к ошибкам в продукте.

В 80-90-х гг. прошлого века при разработке ПО применялись методы, базирующиеся на строго формализованных способах описания ПО и принимаемых технических решений. Однако широкое применение этих методов при разработке конкретных АИС сдерживалось отсутствием адекватных инструментальных средств. Неавтоматизированная разработка АИС сводила преимущества их форматизированного описания к нулю.

Таким образом, к концу XX-го века назрела **необходимость** разработки программно-технологических средств специального класса, реализующих CASEтехнологии создания и сопровождения ПО АИС.

CASE-технология представляет собой совокупность методов проектирования ПО, а также набор инструментальных средств автоматизации, позволяющих в наглядной форме моделировать предметную область, анализировать модель на всех стадиях разработки и сопровождения ПО, а также разрабатывать приложения в соответствии с информационными потребностями пользователей.

1.1.3 Цели, задачи и структура учебного пособия

Цель учебного пособия – изложить современные методы и средства автоматизации разработки ПО АИС на основе CASE – технологии.

Структура учебного пособия представлена на рисунке 1.2.

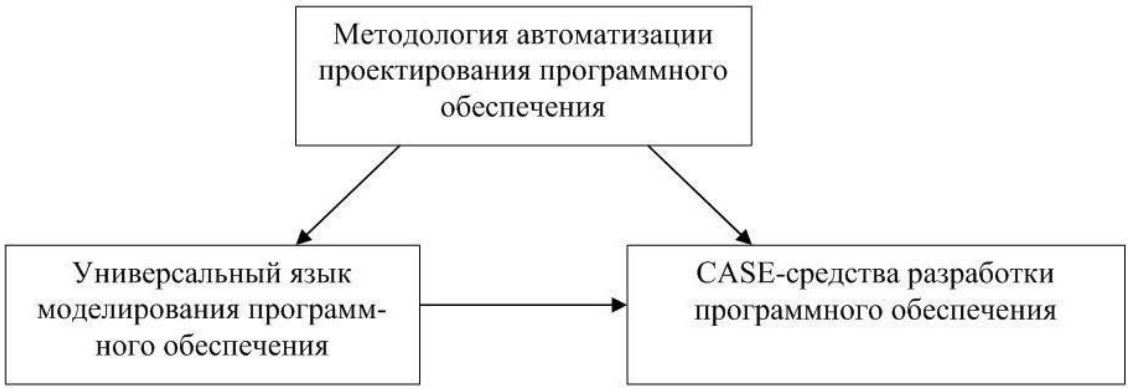


Рисунок 1.2 – Структура учебного пособия

Задачи учебного пособия:

– осветить с системных позиций основные направления исследований, существующие в области программной инженерии;

– рассмотреть современное состояние развития CASE – средств и промышленных технологий разработки ПО;

– изучить унифицированный язык объектно – ориентированного моделирования UML и визуальный редактор на его основе – Rational Rose.

1.1.4 Вопросы и задания для самоконтроля

1 Перечислите причины кризиса программной инженерии.

2 Какая идея лежит в основе программной инженерии?

3 Каковы тенденции развития современных АИС?

4 Дополните определение: «CASE-технология представляет собой совокупность методов проектирования АИС, а также…».

5 Какие методы применялись в 80-90-х годах прошлого века при разработки программного обеспечения (ПО).

1.2 Методологические основы разработки программного обеспечения

Одним из базовых понятий методологии разработки АИС является понятие жизненного цикла (ЖЦ) ее программного обеспечения (ПО). ЖЦ – непрерывный процесс, который начинается с момента принятия решения о необходимости создания ПО и заканчивается в момент его полного изъятия из эксплуатации.

Основным нормативным документом, регламентирующим ЖЦ ПО, является международный стандарт ISO/IEC 12207 (ISO – International Organization of Standardization – Международная организация по стандартизации, IEC – International Electrotechnical Commission – Международная комиссия по электротехнике). Стандарт определяет структуру ЖЦ, содержащую процессы, действия и задачи, которые должны быть выполнены во время создания ПО.

Одним из этапов ЖЦ ПО является проектирование – быстро развивающееся направление исследований в области программной инженерии. Опыт ведения реальных разработок и совершенствование имеющихся программно-аппаратных средств постоянно переосмысливается, в результате чего появляются новые технологии и методы их реализации, которые, в свою очередь, служат основой более современных средств разработки ПО.

1.2.1 Сущность технологии разработки программного обеспечения

Технологии и инструментальные средства разработки составляют основу проекта любой программной системы (ПС). Технологии реализуются через конкретные методы и поддерживающие их стандарты, методики и инструментальные средства, которые обеспечивают процессы реализации различных этапов ЖЦ ПО.

Спиральная модель жизненного цикла ПС, изображенная на рисунке 2.1, наиболее полно отвечает современным подходам к разработке ПО, т.к. предполагают, что технологические процессы выполняются итерационно.

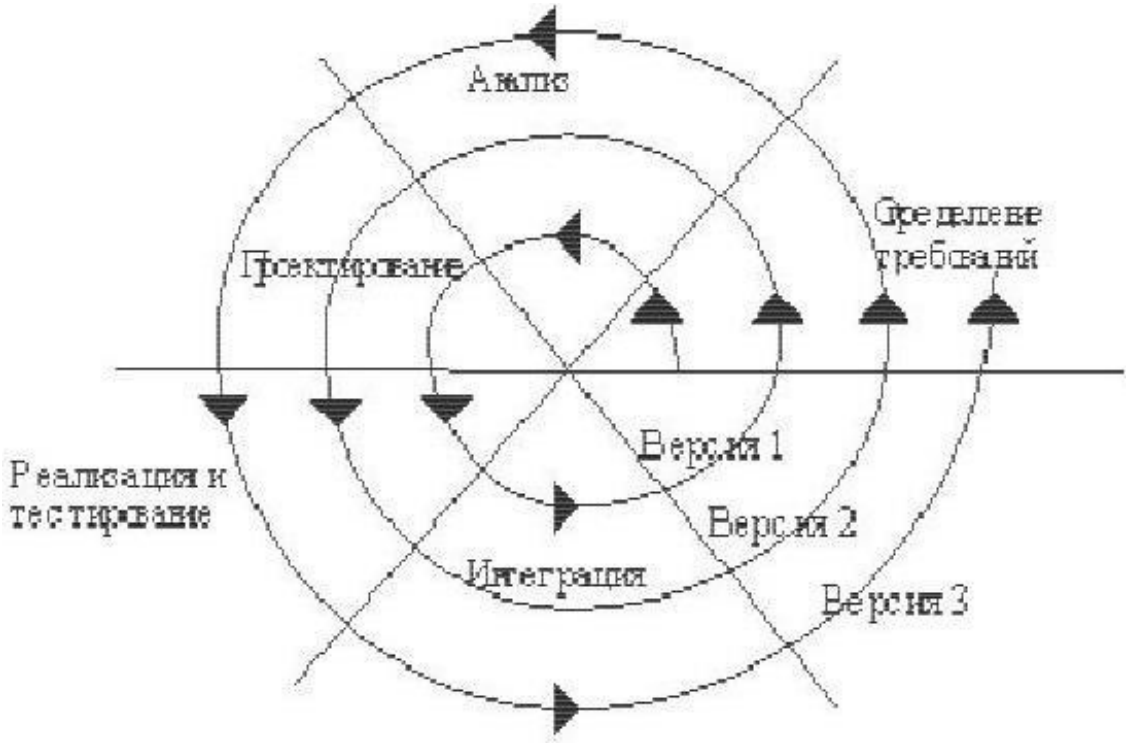


Рисунок 1.3 – Спиральная модель ЖЦ ПС

Основными этапами итерационного процесса являются: постановка задачи, анализ, проектирование, реализация и интегрирование. Наиболее сложным и трудоемким этапом является проектирование ПО.

В настоящее время определение технологии проектирования ПО не имеет устоявшейся формулировки. Учитывая то, что проектирование является одним из этапов ЖЦ ПО, можно остановиться на следующем определении: **технология проектирования ПО это совокупность методов и средств, используемых в процессе создания программных продуктов.**

Как и любая технология, технология программирования представляет собой набор технологических процессов, включающих:

– указание последовательности выполнения технологических операций, обобщенная схема которых представлена на рисунке 1.4;

– перечисление условий, при которых выполняется та или иная операция;

– описание самих операций, каждой из которых ставится в соответствие исходные данные, результаты, а также инструкции, нормативы, стандарты, критерии и методы оценки.

Рисунок 1.4 – Описание технологических операций

Кроме операций и их последовательности, технология определяет метод описания архитектуры проектируемой системы, т.е. модели, используемые на конкретном этапе разработки ПО.

Метод проектирования ПО представляет собой организованную совокупность информационных процессов создания ряда моделей, которые описывают различные аспекты разрабатываемой системы с использованием четко определенных технологических операций.

На формальном уровне метод определяется как совокупность составляющих языка моделирования:

– **концепций** (теоретических основ). В качестве таких основ выступают структурный или объектно – ориентированный подходы (парадигмы) программирования;

– **нотаций,** используемых для построения моделей спецификации статической структуры и динамики поведения проектирования АИС. В качестве таких нотаций обычно используются графические диаграммы (диаграммы потоков данных, диаграммы «сущность – связь», диаграммы вариантов использования (структурный подход), диаграммы классов (ООП));

– **руководства (правила),** определяющих практическое применение метода (последовательность и правила построения моделей, критерии, используемые для анализа результатов).

На рисунке 1.5 представлена структура языка моделирования, отражающего метод описания программного продукта.

Рисунок 1.5 – Составляющие языка моделирования

К сожалению, в настоящее время не существует общепризнанного определения архитектуры ПО. Данное понятия определяется различными способами, например, «Программная архитектура есть абстрактная спецификация системы, состоящая из основных функциональных компонентов, описываемых в терминах их поведения, их интерфейсов и межкомпонентного взаимодействия» (Хэйес – Рос). «Архитектура есть структура компонентов программы – системы, их взаимосвязи, правила и руководящие принципы организации ее проектирования и дальнейшей эволюции» (Галэн, Пэрри).

В основополагающем учебнике под архитектурой понимается совокупность базовых концепций (принципов) её построения, которые определяются сложностью решаемых задач, степенью универсальности ПО и числом пользователей, одновременно с ним работающих.

Несмотря на отличия, имеющиеся в определениях программной архитектуры, в каждом из них делается акцент на структурные аспекты организации ПО. Отсюда наиболее адекватным необходимо признать следующее определение – **программная архитектура** представляет собой совокупность моделей структурных элементов системы с видимыми извне свойствами и механизмами их взаимодействия.

Различают одно и многопользовательскую архитектуры.

Однопользовательские архитектуры реализуют в виде:

– программа или программное средство (адресованный компьютеру *набор инструкций*, точно описывающий последовательность действий, которые необходимо выполнить для решения конкретной задачи);

– пакета программ (*совокупность программ*, решающих задачи некоторой предметной области, например, библиотека программ);

– программной системы (организованная совокупность программ, позволяющих решать широкий класс задач из одной предметной области);

– программного комплекса (*совокупность программных систем*, обеспечивающих решение класса сложных задач предметной области).

Многопользовательские программные системы организуют сетевое взаимодействие отдельных компонентов ПО, построенные по принципам «файл – сервер», «клиент – сервер» и т.д.

Реальное применение любой технологии проектирования, разработки и сопровождения АИС в конкретной организации и конкретном проекте невозможно без ряда стандартов (правил, соглашений), которые должны соблюдаться всеми участниками проекта. К таким стандартам относятся:

– стандарт проектирования;

– стандарт оформления проектной документации;

– стандарт пользовательского интерфейса.

Содержание стандартов рассматривается в курсе ТРПО.

Для успешной реализации проекта объект проектирования (АИС) должен быть прежде всего адекватно описан, т.е. должны быть построены полные и непротиворечивые модели архитектуры ПО.

Модели представляют собой средства для визуализации описания, проектирования и документирования архитектуры программной системы. По мнению одного из авторитетных специалистов в области программной инженерии Гради Буча, моделирование является центральным звеном всей деятельности по созданию ПО.

Модели строятся для того, чтобы понять и осмыслить архитектуру и поведение будущей ПС, облегчить управление процессом ее создания и документировать принимаемые проектные решения.

Таким образом, методы проектирования составляют центральную часть формализованной дисциплины выполнения проекта любого ПО и является основой совершенствования технологий.

1.2.2 Эволюция технологий проектирования программного обеспечения

Известная формула Вирта «алгоритмы + структура данных = программа» свидетельствует, что в недрах ПО существуют два начала, две противоположности, находящиеся, как водится, в диалектическом единстве и борьбе. Одно начало императивное, алгоритмическое, а другое – декларативное, непроцедурное, основанное на моделях.

Для уточнения содержания технологий и определения тендеции их развития, целесообразно рассмотрение технологий проектирования ПО в историческом аспекте. На рисунке 1.6 показана эволюция технологий проектирования ПО – объективный процесс единства и борьбы названных противоположностей, движущей силой которого является увеличение сложности разрабатываемых программных продуктов.

Рисунок 1.6 – Эволюция технологий проектирования

С теоретической точки зрения технологии различаются содержанием и последовательностью технологических операций, методами их описания и архитектурой разработанного ПО.

Технологии процедурного (стихийного) программирования. Заря информационных технологий была ознаменована полным, безраздельным торжеством алгоритмического начала. Это начало, чуждое стилю человеческого мышления, которое практически не использует алгоритмическую форму для изложения своих результатов, привело к возникновению новой профессии – программистов. В этот период (40..60 гг. XX-го века) программирование фактически являлось искусством. Программисты надолго оттеснили от компьютеров специалистов предметных областей знаний.

Последовательность технологических операций, характерная для технологий процедурного программирования, представлена на рисунке 1.7.

Первые программы имели простейшую архитектуру и состояли собственно из программ (процедур на машинном языке) и обрабатываемых данных.

Рисунок 1.7 – Последовательность операций технологии процедурного программирования и их исполнители

На верхнем уровне рисунка 1.8 изображена архитектура таких программ.

Сложность программ в машинных кодах ограничивалась способностью программиста одновременно мысленно отслеживать последовательность выполняемых операций и место нахождения данных в физической памяти.

Появление АССЕМБЛЕРА и высокоуровневых языков FORTRAN, ALGOL позволило повысить сложность разрабатываемых ПО за счет использования подпрограмм. Однако слабым местом такой архитектуры было то, что при увеличении количества подпрограмм возрастала вероятность искажения части данных, которые не делились на глобальные и данные подпрограмм.

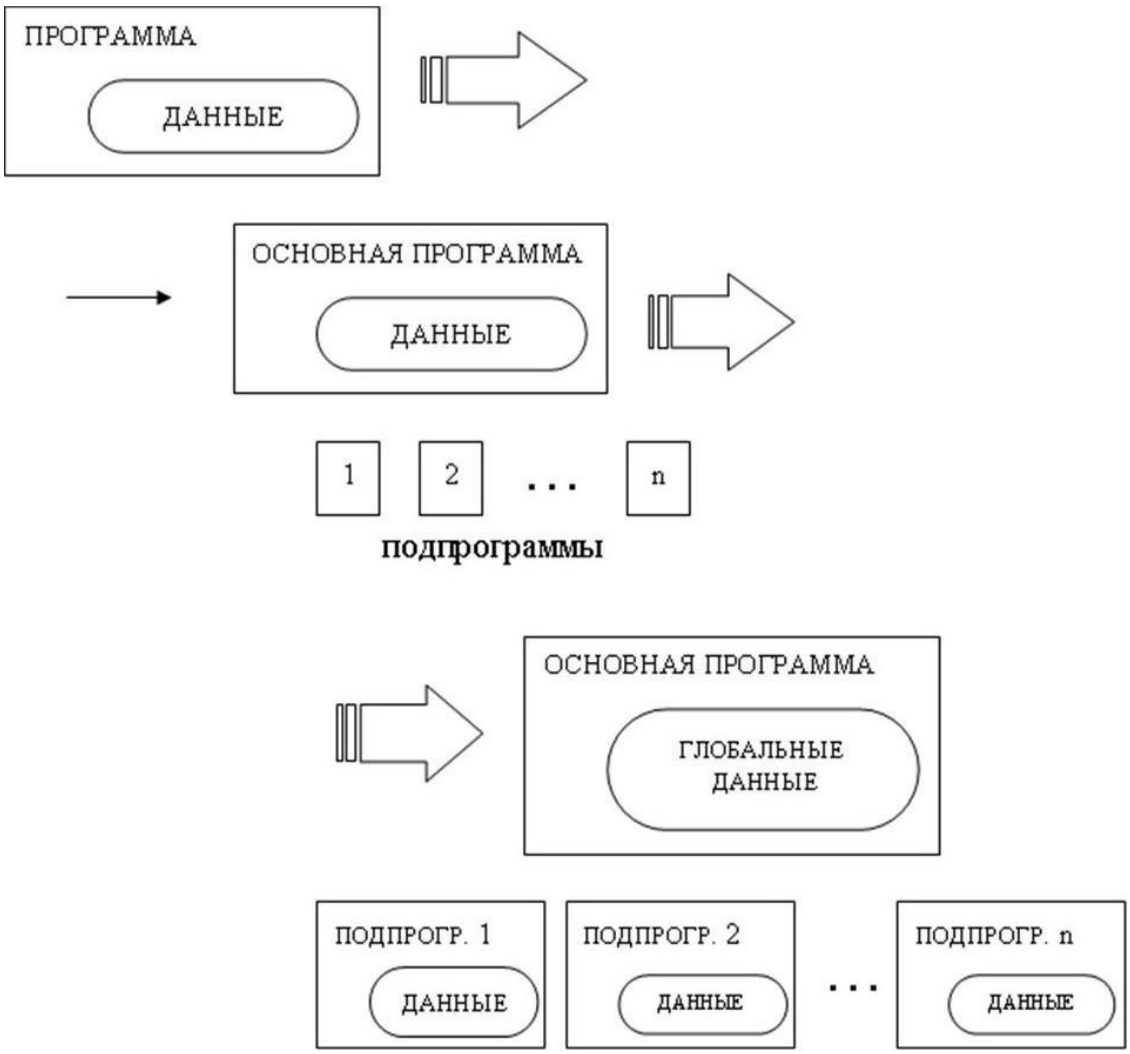


Рисунок 1.8 – Развитие архитектуры программ при технологиях процедурного и структурного программирования

Кроме того, разрабатываемое ПО с локальными данными по прежнему ограничивалось возможностью программиста отслеживать процессы их обработки. Это предопределило возникновение первого «кризиса программирования» (60-е годы XX-го века) – колоссальные успехи в области развития средств вычислительной техники пришли в противоречие с низкой производительностью труда программистов, отсюда проекты устаревали раньше, чем были готовы к внедрению. Появление операционных систем снизило остроту проблем. Однако оставалась разработка ПО «снизу-вверх» – подход при котором сначала разрабатывались сравнительно простые подпрограммы, из которых затем пытались построить сложную программу.

Отсутствие четких моделей описания подпрограмм и методов их проектирования, создание каждой подпрограммы превращалось в непростую задачу: интерфейсы подпрограмм получались сложными и при сборке программного продукта выявлялось большое количество ошибок согласования (80 % времени разработки ПО уходило на тестирование).

Таким образом, эволюция технологий разработки ПО является объективной реальностью и определяется необходимостью преодоления проблем, связанных с ростом сложности ПО, отсутствием автоматизированных средств описания программных систем, потребностью в коллективной разработке и увеличению степени повторяемости программного кода.

1.2.3 Вопросы и задания для самоконтроля

1 Дайте определение технологии проектирования ПО?

2 Что понимают под архитектурой ПО? 3 Что представляют собой модели ПО?

4 В каких случаях строятся модели?

5 Что является центральным процессом моделирования? Что включает в себя язык моделирования?

6 Перечислите последовательность операций технологии процедурного программирования.

7 Какие объекты включает в себя технологические операции?

8 Дайте определение методу проектирования.

9 В чем заключается сущность стихийного программирования?

10 Перечислите и поясните последовательность операций технологий процедурного программирования и их исполнителей.

1.3 Базовые технологии разработки программного обеспечения

Программирование – сравнительно молодая и быстро развивающаяся отрасль науки и техники. Опыт ведения реальных разработок и совершенствование имеющихся программно-аппаратных средств постоянно переосмысливается, в результате чего появляются новые технологии и методы их реализации, которые, в свою очередь, служат основой более современных средств разработки ПО. Однако, в основе любой технологии лежат две базовые парадигмы: структурное и объектноориентированное программирование.

1.3.1 Технологии на основе парадигмы структурного программирования

Дальнейший рост сложности разрабатываемого ПО потребовал структурирования данных и, как следствие, в языках появилась возможность определения пользовательских типов данных. Кроме того, анализ причин возникновения большинства ошибок технологии процедурного программирования позволил сформулировать новый подход к программированию, который был назван «структурным».

Сущность структурного подхода к разработке АИС заключается в **декомпозиции** проектируемой системы на автоматизируемые функции: система разбивается на функциональные подсистемы, которые в свою очередь делятся на подфункции, подразделяемые на задачи и так далее. В отличии от используемого ранее процедурного подхода к декомпозиции, структурный подход предполагает представление задачи **в виде иерархии** подзадач простейшей структуры (40..50 операторов). Проектирование осуществляется «свреху-вниз» и подразумевает реализацию общей идеи за счет разработки интерфейсов подпрограмм, а также специальный метод проектирования алгоритмов – **метод пошаговой детализации**.

При этом последовательность технологических операций, характерная для технологий структурного программирования, практически не изменилась (см. рисунок 1.8).

Все наиболее распространенные технологии структурного подхода базируются на ряде общих принципов:

– принцип "разделяй и властвуй" – принцип решения сложных проблем путем их разбиения на множество меньших независимых задач, легких для понимания и решения;

– принцип иерархического упорядочивания – принцип организации составных частей проблемы в иерархические древовидные структуры с добавлением новых деталей на каждом уровне.

– принцип абстрагирования – заключается в выделении существенных аспектов системы и отвлечения от несущественных;

– принцип структурирования данных – заключается в том, что данные должны быть структурированы и иерархически организованы.

Поддержка принципов структурного программирования заложено в основу так называемых структурных языков программирования – Pascal, C, PL/1.

В структурном анализе используются модели, иллюстрирующих функции, выполняемые системой, и модели, описывающие отношения между данными:

– SADT (Structured Analysis and Design Technique) – функциональные модели;

– DFD (Data Flow Diagrams) – диаграммы потоков данных;

– ERD (Entity-Relationship Diagrams) – диаграммы «сущность-связь».

Технология SADT разработана Дугласом Россом и на ее основе принят известный стандарт IDEF0 (Icam DEFinition), который является основной частью программы ICAM (Интеграция компьютерных и промышленных технологий). Технология возникла под влиянием PLEX, концепции клеточной модели человекоориентированных функций Хори, общей теории систем, технологий программирования и кибернетики.

SADT представляет собой совокупность концепций, нотаций и правил, предназначенных для построения функциональной модели объекта предметной области.

Элементы этой технологии основываются на концепции графического представление блочного моделирования – SADT-диаграммы отображают функции в виде блоков, взаимодействующих друг с другом посредством интерфейсных дуг. Место соединения дуги с блоком определяет тип интерфейса. Управляющая информация входит в блок сверху, в то время как информация, которая подвергается обработке, показана с левой стороны блока, а результаты показаны с правой стороны. Механизм (человек или автоматизированная система), который осуществляет операцию, представляется дугой, входящей в блок снизу.

На рисунке 1.9 представлена основная **нотация** SADT-модели любой предметной области.

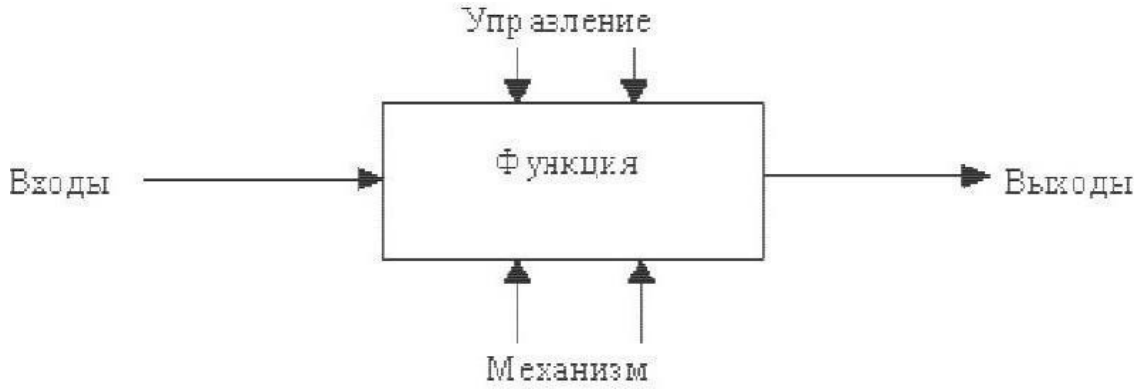


Рисунок 1.9 – Функциональный блок и интерфейсные дуги

Правила SADT включают:

– ограничение количества блоков на каждом уровне декомпозиции (правило 3-6 блоков);

– связность диаграмм (номера блоков), уникальность меток и наименований (отсутствие повторяющихся имен);

– разделение входов и управлений (правило определения роли данных) и отделение организации от функций, т.е. исключение влияния организационной структуры на функциональную модель.

Одной из наиболее важных особенностей технологии SADT является постепенное введение все больших уровней детализации по мере создания диаграмм, отображающих модель. На рисунке 1.10 приведены четыре уровня диаграммы SADT-модели и их взаимосвязи.

Рисунок 1.10 – Декомпозиция диаграмм SADT-модели

Каждый компонент модели может быть декомпозирован на другой диаграмме. Каждая диаграмма иллюстрирует «внутреннее строение» блока на родительской диаграмме. Пример SADT-модели показан на рисунках 1.11 и 1.12, полученные в CASE-среде BPwin.

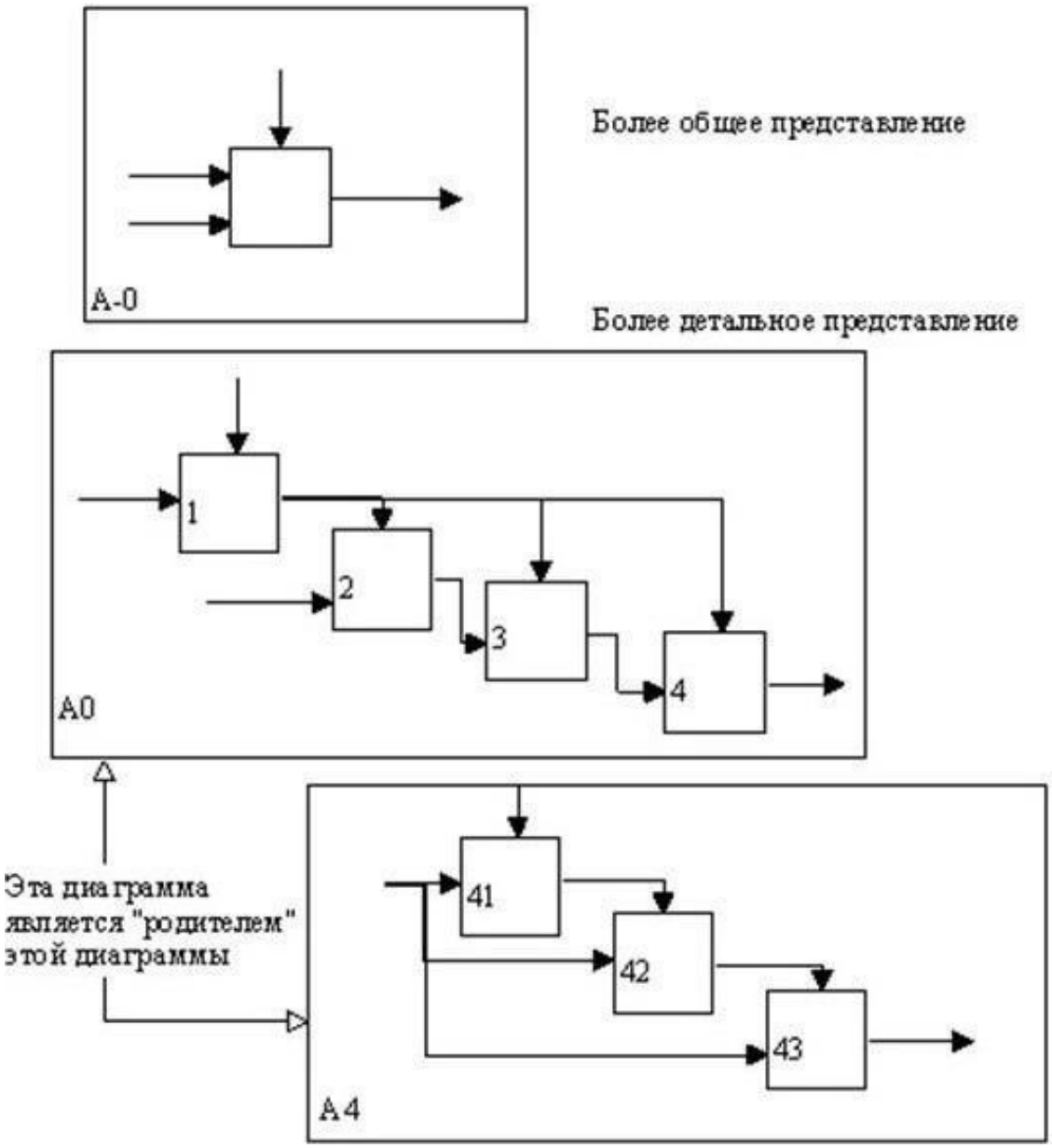


Рисунок 1.11. – Исходная диаграмма SADT-модели

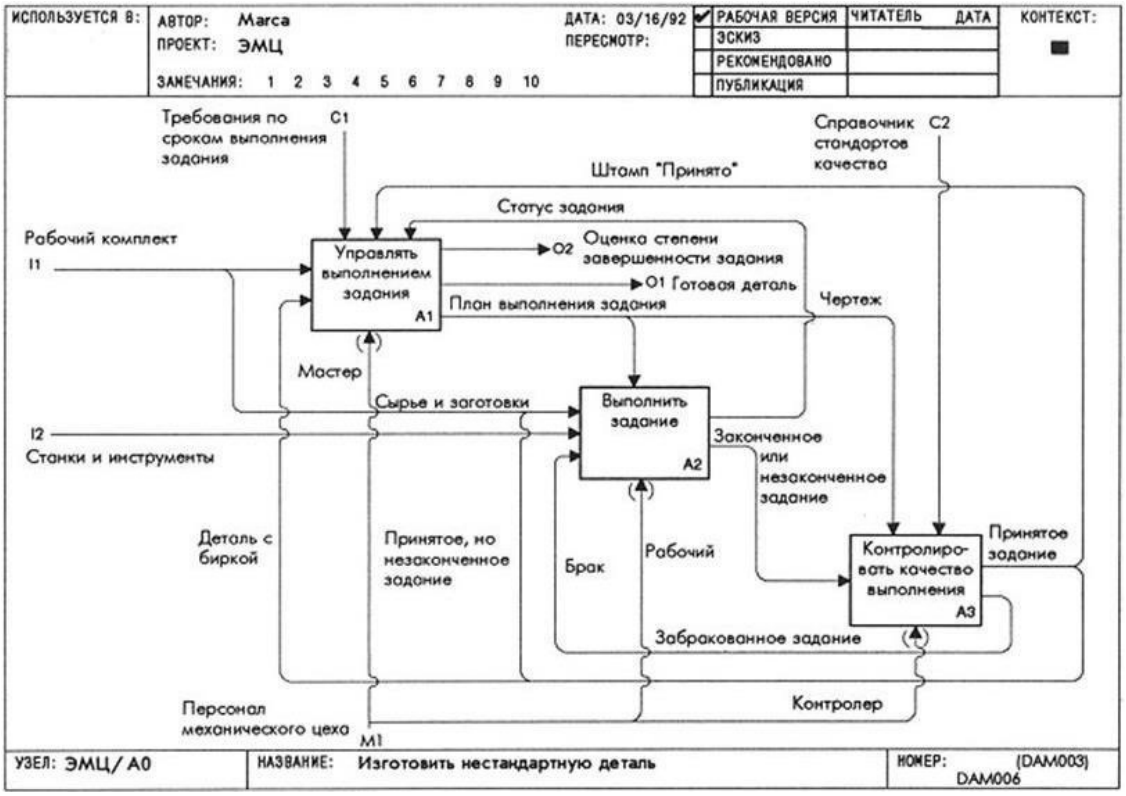


Рисунок 1.12 – Декомпозиция исходной диаграммы SADT-модели

Технология DFD. Графическая модель системы определяется как иерархия диаграмм потоков данных (ДПД), описывающих асинхронный процесс преобразования информации от ее ввода в систему до выдачи пользователю. Диаграммы верхних уровней иерархии (контекстные диаграммы) определяют основные процессы или подсистемы АИС с внешними входами и выходами. Они детализируются при помощи диаграмм нижнего уровня. Такая декомпозиция продолжается, создавая многоуровневую иерархию диаграмм, до тех пор, пока не будет достигнут такой уровень декомпозиции, на котором процессы становятся элементарными и детализировать их далее нецелесообразно.

Основными нотациями ДПД являются: внешние сущности; системы/подсистемы; процессы; накопители данных; потоки данных. Источники информации (внешние сущности) порождают информационные потоки (потоки данных), переносящие информацию к подсистемам или процессам. Те в свою очередь преобразуют информацию и порождают новые потоки, которые переносят информацию к другим процессам или подсистемам, накопителям данных или внешним сущностям – потребителям информации.

Внешняя сущность представляет собой материальный предмет или физическое лицо, представляющее собой источник или приемник информации, например, как показано на рисунке 1.13, заказчики, поставщики, клиенты, склад.

Рисунок 1.13 – Внешняя сущность

Определение некоторого объекта или системы в качестве внешней сущности указывает на то, что она находится за пределами границ анализируемой АИС.

На рисунке 1.14 изображена контекстная диаграмма **подсистемы** (системы). Номер подсистемы служит для ее идентификации. В поле имени вводится наименование подсистемы – предложения с подлежащим и соответствующими определениями и дополнениями.

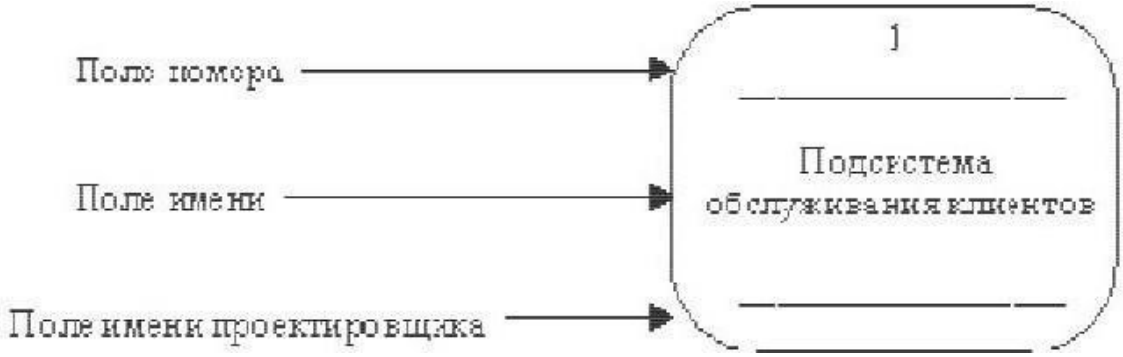


Рисунок 1.14 – Подсистема

Процесс представляет собой преобразование входных потоков данных в выходные на основе определенного алгоритма. Физически процесс может быть реализован различными способами: это может быть подразделение организации (отдел), выполняющее обработку входных документов и выпуск отчетов, программа, аппаратно реализованное логическое устройство и т.д. Процесс на диаграмме потоков данных изображается, как показано на рисунке 1.15.

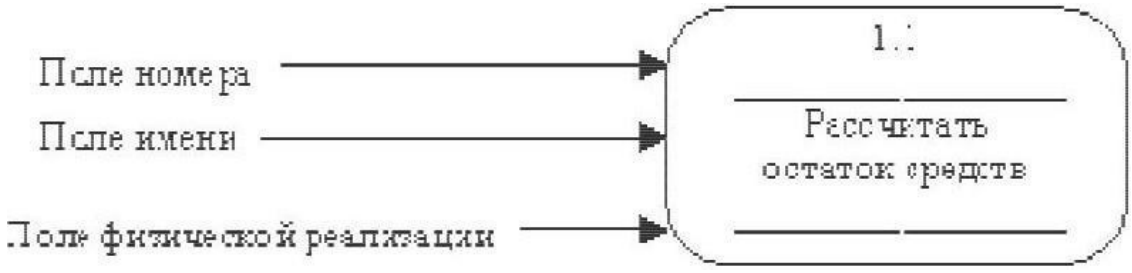


Рисунок 1.15 – Процесс

Номер процесса служит для его идентификации. В поле имени вводится наименование процесса в виде предложения с активным недвусмысленным глаголом в неопределенной форме (вычислить, рассчитать, проверить, определить, создать, получить), за которым следуют существительные в винительном падеже. Информация в поле физической реализации показывает, какое подразделение организации, программа или аппаратное устройство выполняет данный процесс.

Накопитель данных представляет собой абстрактное устройство для хранения информации, которую можно в любой момент поместить в накопитель и через некоторое время извлечь, причем способы помещения и извлечения могут быть любыми. Накопитель данных может быть реализован физически в виде: таблицы в оперативной памяти, файла и т.д. Накопитель на диаграмме потоков данных изображается, как показано на рисунке 1.16.

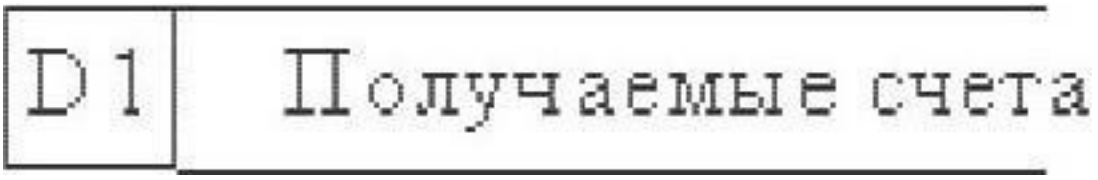


Рисунок 1.16 – Накопитель данных

Накопитель данных идентифицируется буквой "D" и произвольным числом. Накопитель данных в общем случае является прообразом будущей базы данных и описание хранящихся в нем данных должно быть увязано с информационной моделью.

Поток данных определяет информацию, передаваемую через некоторое соединение от источника к приемнику. Реальный поток данных может быть информацией, передаваемой по кабелю между двумя устройствами, пересылаемыми по почте, переносимыми дискетами и т.д.

Поток данных на диаграмме изображается линией, оканчивающейся стрелкой, которая показывает направление потока (рисунок 1.17). Каждый поток данных имеет имя, отражающее его содержание. Средой, использующей DFD-модели, является BPwin, пример реализации которой показан на рисунке 1.17.

Рисунок 1.17 – Диаграмма потоков данных

Дальнейший рост сложности АИС потребовал разграничения доступа к глобальным данным программы. В результате технология структурного программирования получила развитие, отражением которого становится модульное программирования (70 гг. XX в.).

Технология модульного программирования предполагает выделение группы подпрограмм, использующих одни и те же глобальные данные в отдельно компилируемые модули (библиотеки подпрограмм).

Архитектура программы при технологии модульного программирования показана на рисунке 1.18.

Связи между модулями при использовании данной технологии осуществляются через специальный разрабатываемый интерфейс, в то время как доступ к реализации модуля (телу подпрограмм) запрещен. Использование модульного программирования существенно упростило разработку ПО несколькими программистами. Кроме того, модули в дальнейшем без изменений можно было использовать в других проектах.

Эту технологию поддерживают современные версии высокоуровневых языков Turbo Pascal, C + и др.

Практика показала, что структурный подход в сочетании с модульным программированием позволяет получать достаточно надежные программные продукты, размер которых не превышает 100 000 операторов.

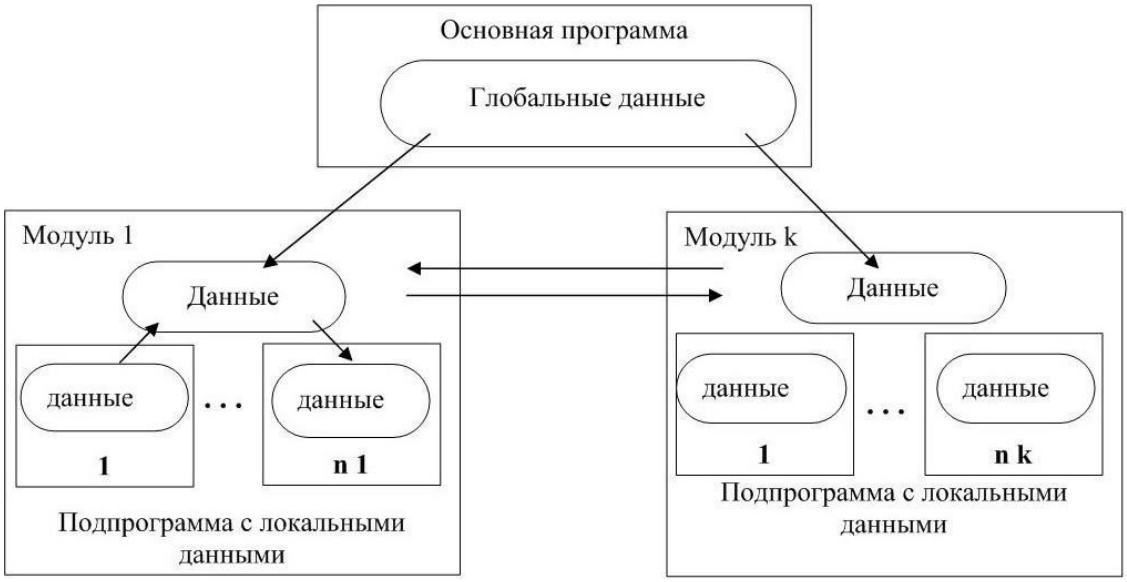


Рисунок 1.18 – Архитектура программы при технологии модульного программирования

Таким образом, узким местом технологии модульного программирования является то, что при увеличении размера программы обычно возрастает сложность межмодульных интерфейсов, и, с некоторого момента, предусмотреть взаимовлияние отдельных частей программы становиться практически невозможным.

1.3.2 Технологии на основе парадигмы объектно-ориентированного программирования

В 1980-90 гг. для проектирования ПО большого объема предложена к использованию технология объектно-ориентированная программирования (ООП). ООП определяется как технология, основанная на представлении программной архитектуры в виде совокупности

объектов, каждый из которых является экземпляром определенного типа (класса), а классы образуют иерархию объектов.

Такая технология требует переосмысления роли фундаментальных понятий прикладных информационных технологий – **модели и алгоритма** (рисунок 1.19).

Модель является базовым понятием для любых областей знаний, поскольку каждая попытка работать в точных терминах с реальным явлением должна начинаться с описания его формальной модели.

Именно модель представляет объект исследования и определяет характер формального аппарата, используемого для описания задачи и выполнения необходимых преобразований информации. Модель объекта вычислений определяет *ЧТО* надо вычислить, а алгоритм определяет *КАК* нужно вычислять. Простая истина – прежде, чем определить *КАК*, необходимо сформулировать *ЧТО* является объектом решения, т.е. построить модель, очевидна для всякой науки, использующей математику.

Рисунок 1.19 – Определения модели и алгоритма

Отсюда, особенностью последовательности технологических операций ООП, изображенной на рисунке 1.19, является появление этапов моделирования и документирования, характерных для сложных программных проектов.

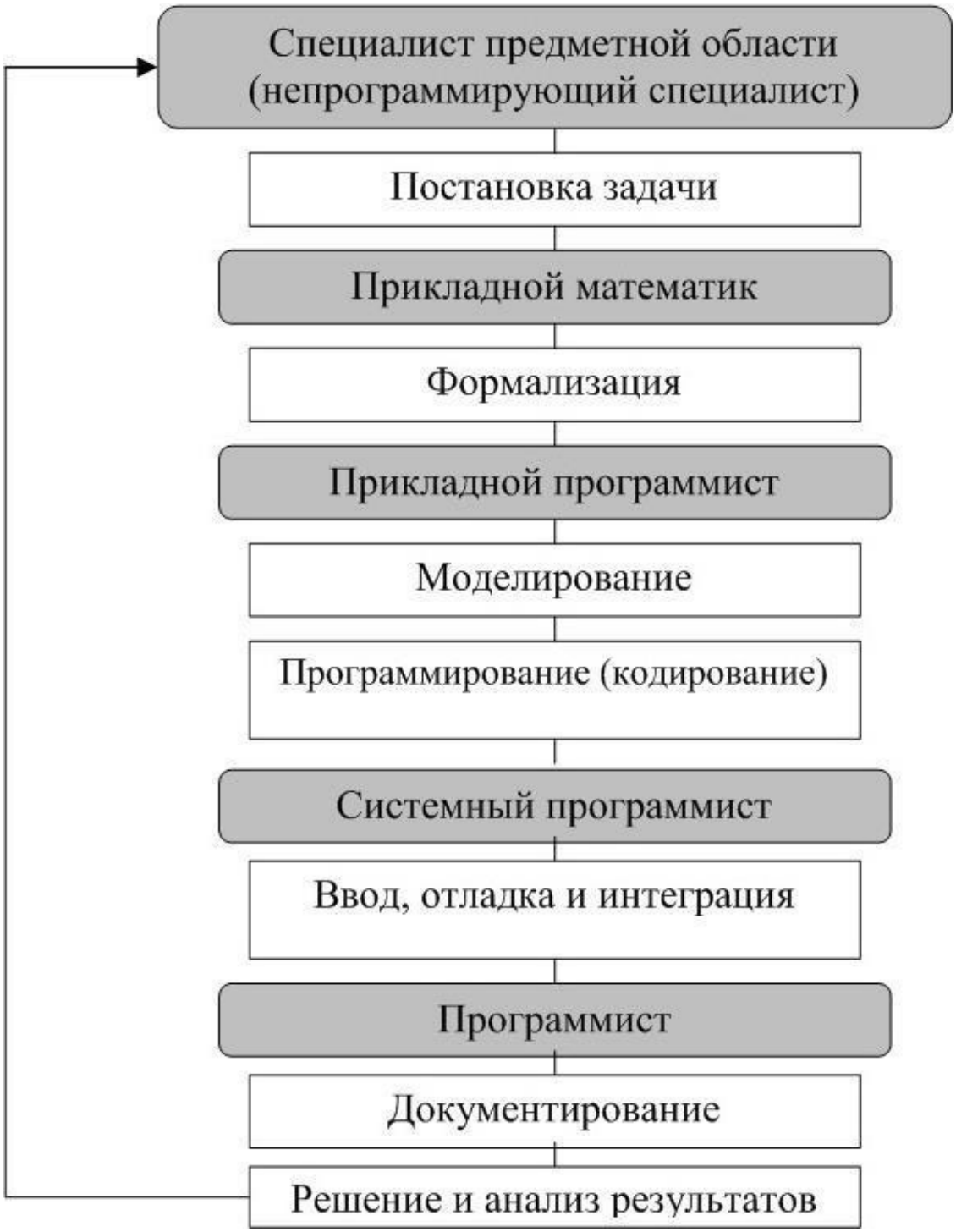


Рисунок 1.20 – Последовательность операций технологии ООП

Этап характеризуется появлением объектных языков программирования – Object Pascal, C++, в основе которых лежат следующие основные концепции:

– **класс** является описываемой на языке терминологии исходного кода моделью ещё не существующей сущности, т.е. объекта. Класс можно сравнить с чертежом, согласно которому создаются объекты. Обычно классы разрабатывают таким образом, чтобы их объекты соответствовали объектам предметной области.

– **объект** является сущностью в адресном пространстве вычислительной системы, появляющаяся при создании экземпляра класса (например, после запуска результатов компиляции исходного кода на выполнение).

Взаимодействие программных объектов в такой системе осуществляется путем передачи сообщений. Объект класса при этом обладает рядом характерных свойств (механизмов): абстрагирование, наследование, инкапсуляция, полиморфизм, существенно снижающая сложность проектирования ПО.

Абстрагирование – это способ выделить набор значимых характеристик объекта, исключая из рассмотрения незначимые. Соответственно, абстракция – это набор таких характеристик.

Инкапсуляция – это свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе и скрыть детали реализации от пользователя.

Наследование – это свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью.

Полиморфизм – это свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

В результате существенно увеличивается показатель повторяемости использования кода и появляется возможность создания библиотек классов для различных применений.

Другой характерной особенностью технологии ООП является архитектура программы, представленная на рисунке 1.20.

Реализацией технологии ООП в рамках спиральной модели ЖЦ является получившая в последнее время широкое распространение технология быстрой разработки приложений RAD (Rapid Application Development).

Основные принципы (концепции) технологии RAD:

– разработка приложений итерациями;

– необязательность полного завершения работ на каждом из этапов ЖЦ;

– обязательное вовлечение пользователей в процесс разработки АИС;

– необходимое применение CASE-средств, обеспечивающих целостность проекта;

– применение средств управления конфигурацией, облегчающих внесение изменений в проект и сопровождение готовой системы;

– использование прототипирования, позволяющее полнее выяснить и удовлетворить потребности конечного пользователя;

– тестирование и развитие проекта одновременно с его разработкой;

– ведение разработки немногочисленной хорошо управляемой командой профессионалов;

– грамотное руководство разработкой системы, четкое планирование и контроль выполнения работ.

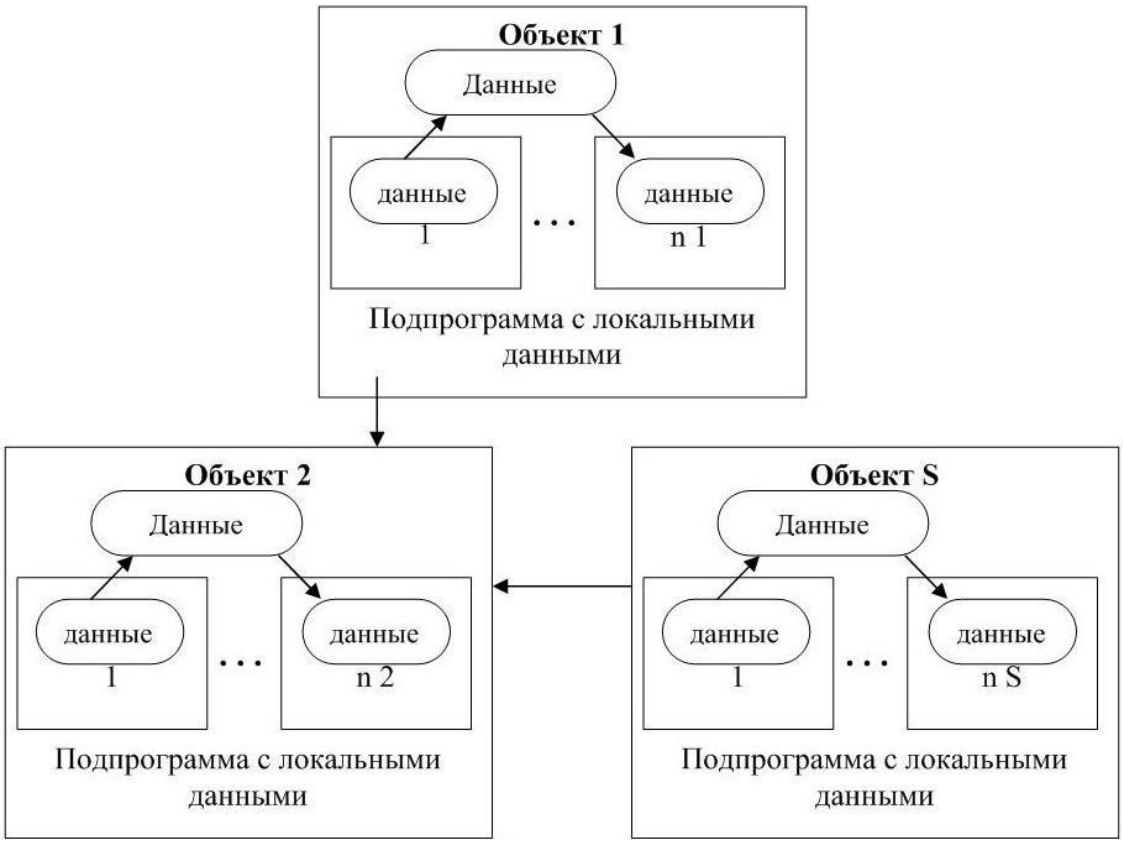


Рисунок 1.21 – Архитектура программы при технологии ООП

Процесс разработки программных систем по технологии RAD содержит следующие требования:

– небольшую команду программистов (от 2 до 10 человек);

– короткий производственный график (от 2 до 6 мес.);

– повторяющийся цикл, при котором разработчики, по мере того, как приложение начинает обретать форму, запрашивают и реализуют в продукте требования, полученные через взаимодействие с заказчиком.

Этапы спиральной модели ЖЦ программных систем, выполняемых в соответствии с технологией RAD, представлены на рисунке 1.22.

Рисунок 1.22 – ЖЦ АИС по технологии RAD

На этапе анализа и планирования пользователи системы определяют функции и требования АИС, выделяют наиболее приоритетные функции, описывают информационные потоки. Определение требований выполняется в основном силами пользователей под руководством специалистов-разработчиков. Ограничивается масштаб проекта, определяются временные рамки для каждого из последующих этапов. Результатом данного этапа являются техническое задание на разработку АИС.

На этапе проектирования пользователи принимают участие в техническом проектировании системы под руководством специалистов-разработчиков. CASEсредства используются для быстрого получения работающих прототипов приложений. Пользователи, непосредственно взаимодействуя с ними, уточняют и дополняют требования к системе. Более

подробно рассматриваются процессы системы. Анализируется и, при необходимости, корректируется функциональная схема (модель). Каждая функция рассматривается детально. При необходимости для каждого элементарного процесса создается частичный прототип: экран, диалог, отчет, устраняющий неясности или неоднозначности. Определяются требования разграничения доступа к данным. На этом этапе формируется список необходимой документации.

После детального определения состава процессов оценивается количество функциональных элементов разрабатываемой системы и принимается решение о разделении АИС на подсистемы, поддающиеся реализации одной командой разработчиков за приемлемое для RAD-проектов время (60 – 90 дней). Проект распределяется между различными командами (делится функциональная модель).

Результаты этапа:

– общая информационная модель системы;

– функциональные модели системы в целом и подсистем, реализуемых отдельными командами разработчиков;

– точно определенные интерфейсы между автономно разрабатываемыми подсистемами;

– прототипы экранов, отчетов, диалогов.

На этапе реализации выполняется непосредственно быстрая разработка приложения – разработчики производят итеративное построение реальной системы на основе полученных на предыдущем этапе моделей. Программный код частично формируется при помощи автоматических генераторов, получающих информацию непосредственно из репозитория CASE-средств. Конечные пользователи оценивают получаемые результаты и вносят коррективы, если в процессе разработки система перестает удовлетворять определенным ранее требованиям. Тестирование системы осуществляется непосредственно в процессе разработки.

После окончания работ каждой отдельной команды разработчиков производится постепенная интеграция данной части системы с остальными, формируется полный программный код, выполняется тестирование совместной работы данной части приложения с остальными, а затем тестирование системы в целом. Результатом этапа является готовая система, удовлетворяющая всем согласованным требованиям.

На этапе внедрения производится обучение пользователей, организационные изменения и опытная эксплуатация новой системы.

Оценка размера приложений производится на основе так называемых функциональных точек (экраны, сообщения, отчеты, файлы и т.п.) Подобная метрика не зависит от языка программирования, на котором ведется разработка. Размер приложений, разработанных по технологии RAD, для хорошо отлаженной среды разработки АИС с максимальным повторным использованием программных компонентов представлен в таблице 1.1.

Таблица 1.1 – Характеристика приложений, реализуемых по технологии RAD

Количество Функциональных точек	Характеристика группы Разработчиков
< 1000	один человек
1000-4000	одна команда разработчиков
> 4000	4000 функциональных точек на одну команду разработчиков

Технология RAD, соответствующая парадигме ООП, наряду с неоспоримыми преимуществами, обладает рядом существенных недостатков:

– отсутствие стандартов компоновки двоичных результатов компиляции объектов в единое целое даже в рамках одного языка программирования;

– взаимодействия между объектами требует разработки интерфейса, а, следовательно, дополнительных затрат времени и возникновение возможности ошибки в коде;

– изменение реализации одного объекта требует перекомпиляции всего программного продукта.

Таким образом, технология RAD эффективна для программных проектов средней сложности под конкретного заказчика. Разработка сложных программных систем (операционные системы, системы реального масштаба времени), т.е. программы с большим процентом уникального кода, требуют более высокого уровня планирования и жесткой дисциплины проектирования.

Для преодоления указанных недостатков ООП получил развитие компонентно-ориентированная парадигма программирования.

1.3.3 Вопросы и задания для самоконтроля

1 Что послужило формированию нового дохода к программированию который был назван «структурным».

2 В чем заключается сущность структурного подхода?

3 Охарактеризуйте технологию SADT. Перечислите правила SADT.

4 Охарактеризуйте технологию DFD. Дайте определение внешней сущности.

5 В чем заключается технология модульного программирования? Поясните архитектуру при технологии модульного программирования.

6 Поясните архитектуру программы при объектно – ориентированной технологии.

7 Дайте определение понятиям *модель* и *алгоритм.*

8 Перечислите последовательность операций технологии ООП.

9 Перечислите этапы спиральной модели ЖЦ АИС по технологии RAD. Охарактеризуйте каждый этап ЖЦ.

10 Перечислите недостатки характерные технологии RAD.

1.4 Современные технологии разработки программного обеспечения

Разработка программного обеспечения является молодой и быстро развивающейся отраслью инженерной науки, которая подвержена постоянным и быстрым изменениям. Так, лишь в начале 90-х годов Британское сообщество вычислительной техники (British Computer Society) начало присваивать разработчикам программ квалификацию инженера (Chartered Engineer), а в Соединенных Штатах (в штате Техас) только в 1998 году стало возможным зарегистрироваться в качестве профессионального инженера программного обеспечения. Но по-прежнему, даже в начале 21-го века, ***общепризнанным остается тот факт, что разработке программного обеспечения не достает развитой научной базы.*** По некоторым оценкам, 75 % организаций программной индустрии занимаются разработкой программ на интуитивном уровне. С другой стороны, в этой области сформировалось немало интересных идей и знакомство с ними является содержанием настоящей лекции.

1.4.1 Технологии компонентно-ориентированного программирования

Технологии компонентно-ориентированного программирования (КОП) определяет **стандартный механизм**, с помощью которого одна часть ПО предоставляет свои услуги другой части. Организация предоставления услуг в библиотеках, приложениях, системном и сетевом программном обеспечении позволяет изменить технологию создания программ.

Наиболее известные технологии КОП представлены на рисунке 1.23.

COM (Component Object Model) – компонентная модель объектов	CORBA (Common Object Request Broker Architecture) - общая архитектура с посредником обработки запросов объектов

Рисунок 1.23 – Технологии компонентно-ориентированного программирования

Концепция технологии COM для семейства операционных систем Windows заключается в построении программ из компонент, которые состоят из объектов, представляющих собой непосредственно **исполняемый двоичный код**. Важнейший признак «компонентности» – исполняемую программу можно собирать из отдельных частей **без операций сборки** (модуля).

COM устанавливает **понятия и правила**, необходимые для определения **объектов и интерфейсов**; кроме того, в ее состав входят программы, реализующие ключевые функции. В COM любая часть ПО реализует свои услуги с помощью объектов COM. Каждый объект COM поддерживает несколько интерфейсов. Клиенты могут получить доступ к услугам объекта COM только через вызовы операций его интерфейсов – нет непосредственного доступа к телу объекта.

Отличие COM от привычных объектов в стиле ООП состоит в том, что объекты ООП известны только компилятору. Это абстракции, в которых мыслит программист и которые компилятор превращает в двоичные структуры «данные + код». Технология COM есть технология, которая переносит все преимущества ООП, доступные программисту на уровне исходного текста, на **двоичный уровень**. Если в исходном тексте технологии ООП про-

граммист волен использовать любые объекты, но теряет всяческий контроль над тем, что сделано, как только исходный текст скомпилирован, то при использовании COM эти возможности сохраняются на протяжении всего жизненного цикла программы. Кроме того добавляются возможности разделения проекта на отдельные, **повторно используемые, двоичные компоненты.**

Хронология развития технологии COM показана на рисунке 1.24.

Рисунок 1.24 – Хронология развития технологии COM

Использованные сокращения: Dynamic Link Libraries (DLL) – динамически подключаемые библиотеки, Open DataBase Connectivity (ODBC) – открытый интерфейс доступа к базам данных, встроенный в Windows и Windows NT, Dynamic Data Exchange (DDE) – динамический обмен данными, Object Linking and Embedding (OLE1.0) – внедрение и связывание объектов, **OLE2.0 –** OLE **на базе COM**, Distributed COM (DCOM) – распределенная модель компонентных объектов, COM+ – новейшая технология COM.

Главная идея технологии **ODBC** заключается в создании **промежуточного программного слоя**, который определяет стандартный интерфейс для приложений. На уровне вызовов этот интерфейс использует язык SQL, а реализация взаимодействия с БД обеспечивается драйверами, поставляемые в форме DLL. Таким образом, ODBC располагается между приложением и источниками данных различных форматов.

Технологию динамического обмена данными **DDE** можно рассматривать как попытку **стандартизации обмена данными** между приложениями. Концепция Windows основана на обработке сообщений (messages). Отсюда приложения могут обмениваться друг с другом сообщениями, используя общую очередь сообщений. Проблема состоит в том, что каждое из приложений должно знать протокол обмена данными, т.е. **формат сообщений**. Технология DDE как раз и предложила такой **стандартный протокол**, реализованный во многих приложениях. Недостатками DDE является сложность программирования, невысокая надежность и то обстоятельство, что приложения должны знать формат передаваемых данных.

Технологию внедрения и связывания объектов **OLE1.0** фирма Microsoft представила в 1991 г. как попытку реализации **объектно-ориентированного механизма взаимодействия приложений**. Главной идеей OLE является **концепция составного документа**, который может содержать объекты других приложений. До OLE приложения могли обмениваться **статическими снимками данных** через буфер обмена Windows. Однако редактирование таких данных должно выполняться тем приложением, которое их породило, а после редактирования они вновь должны быть вставлены в другой документ. Если изменяются исходные данные, то, очевидно, должны изменяться данные и в составном документе. Однако, системный буфер обмена не имеет никаких средств поддержания таких связей. Более того, проблемы возникают и при перемещении исходных данных в новое место. Внедренный с помощью OLE1.0 объект содержит статические данные и данные, необходимые для его редактирования. Для редактирования объекта пользователю приложенияконтейнера необходимо щелкнуть по объекту, вследствие чего в отдельном окне запускается исходное приложение, породившее эти данные. По окончании редактирования пользователь может сохранить данные, которые будут обновлены и в приложении-контейнере.

Недостатками технологии OLE1.0 являются:

– базовый механизм OLE1.0 – DDE по своей природе асинхронен, т.е. возврат управления при вызове любой функции происходит немедленно, но после завершения операции;

– для передачи данных между приложениями используется разделяемая глобальная память, т.е. данные сначала копируются в нее, а затем могут быть вытолкнуты Windows в **файл подкачки**, вследствие чего замедляется работа приложения;

– связи OLE1.0 легко разрываются при перемещении файлов;

– пользователю неудобно редактировать данные в отдельном окне.

Архитектура программных компонентов, разработанных по технологии COM, и взаимодействие COM-объектов показана на рисунке 1.25.

По технологии COM приложение представляет собой службы (функции) использующие специальные объекты – объекты COM, которые являются экземпляром класса COM. Объект COM включает **поля и методы**, но может реализовать **несколько интерфейсов**, обеспечивающих доступ к его полям и функциям (достигается за счет организации отдельной таблицы адресов методов для каждого интерфейса). При этом интерфейс объединяет несколько однотипных функций. Кроме того, классы COM поддерживают **наследование интерфейсов**, но не поддерживают наследование реализации, т.е. **не наследуют код методов**, хотя при необходимости объект класса-потомка может вызвать метод родителя.

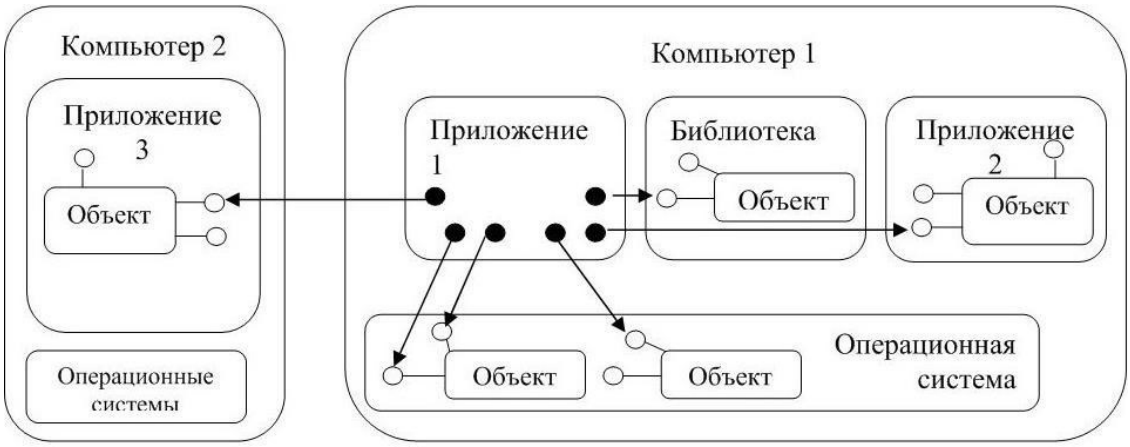


Рисунок 1.25 – Архитектура приложений на основе программных компонентов

Объекты COM всегда функционируют в составе сервера – динамической библиотеки и исполняемого файла. Различают три типа серверов, представленных на рисунке 1.26.

Внутренний сервер	**Локальный сервер**	**Удаленный сервер**
Реализуется динамическими библиотеками, которые подключаются к приложению - клиента и работают в одном адресном пространстве.	Реализуется отдельным процессом (модулем .exe) который работает на одном компьютере с клиентом.	Реализуется процессом, который работает на другом компьютере.

Рисунок 1.26 – Реализация технологии COM

Например, Microsoft Word является локальным сервером, включающим множество объектов, которые могут использоваться другими приложениями. Для обращения к службам клиент должен получить указатель на соответствующий интерфейс. Перед первым обращением клиент посылает запрос к библиотеке COM, хранящей информацию обо всех зарегистрированных в системе классах COM объектов, и передает ей имя класса, идентификатор

интерфейса и тип сервера. Библиотека запускает необходимый сервер, создает требуемые объекты и возвращает указатели на объекты и интерфейс. Получив указатели, клиент может вызывать необходимые функции объекта.

Модификация COM, обеспечивающая передачу вызовов между компьютерами, называется DCOM. При использовании удаленных серверов в адресном пространстве клиента создается proxy-объект (заместитель объекта COM), а в адресном пространстве сервера COM – заглушка, соответствующая клиенту. Получив задание от клиента, заместитель упаковывает его параметры и, используя службы ОС, передает вызов заглушке. Заглушка распаковывает задание и передает его объекту COM.

Результат возвращается объекту в обратном порядке.

Технология CORBA, разработанная группой компании OMG (группа внедрения объектной технологии программирования), реализует подход, аналогичный COM, но на базе объектов и интерфейсов CORBA. Программное ядро CORBA реализовано для всех основных аппаратных и программных платформ и потому эту технологию можно использовать для создания распределенного ПО **в гетерогенной (разнородной) вычислительной среде**. Организация взаимодействия между объектами клиента и сервера в CORBA осуществляется с помощью специального посредника, названного VisiBroker.

Последовательность операций технологии КОП представлена на рисунке 1.27.

Рисунок 1.27 – Последовательность технологических операций КОП

Таким образом, технология КОП существенно упрощает разработку, позволяет сократить сроки проектирования за счет повышения повторяемости элементов программных системы и автоматизации отдельных этапов разработки, т.е. реально перейти к индустриальному методу разработки программного обеспечения АИС

1.4.2 Case-технологии проектирования программного обеспечения

Современные методологии и реализующие их технологии поставляются в электронном виде вместе с CASE-средствами и включают библиотеки процессов, шаблонов, методов, моделей и других компонент, предназначенных для построения ПО того класса систем, на который ориентирована методология. CASE-средства поддерживают внедрение автоматизированных технологий на всех этапах жизненного цикла ПО – **CASE-технологии** (Computer-Aided Software/System Engineering – разработка программного обеспечения/программных систем с использованием компьютерной поддержки).

Существующие CASE-средства поддерживают как структурный, так и объектный подходы (в том числе и компонентный) к программированию и являются предметом изучения настоящей дисциплины.

Конец ознакомительного фрагмента.