Лекция №1

Содержание лекции

Информационный процесс

Все процессы в природе сопровождаются **сигналами**. Такие изменения можно наблюдать, измерять или фиксировать, при этом возникают и **регистрируются** новые сигналы, то есть, образуются данные.

Данные – это зарегистрированные сигналы.

Данные несут в себе **информацию** о событиях, произошедших в материальном мире, поскольку они являются регистрацией сигналов, возникших в результате этих событий. Однако данные **не тождественны** информации. Для того чтобы данные дали информацию необходимо наличие метода обработки данных.

Информация – это продукт взаимодействия данных и адекватных им методов.

Информация есть обработанные данные, а данные есть зарегистрированные сигналы. Таким образом, информацию можно считать некоторой материальной величиной, которую можно получать, хранить, передавать, обрабатывать, воспроизводить. Все перечисленные возможности работы с информацией являются основными составляющими **информационного процесса**.

Информационный процесс – это любой процесс, в котором присутствует хотя бы один из элементов: прием информации, ее хранение, обработка, передача, воспроизведение.

Так как понятие «данные» используется на самом низком уровне обработки, то в дальнейшем будем пользоваться только понятием «информация» – мало-мальски обработанные данные.

Информационная система

Информационная система – это любая система, реализующая или поддерживающая информационный процесс.

К информационным можно относить любые системы, включающие в себя работу с информацией. В настоящее время основным помощником человека при работе с информацией является компьютер, поэтому именно его мы и будем рассматривать в качестве источника, способа изменения и хранения информационных систем. А в качестве информационных систем будем рассматривать программное обеспечение компьютера.

В зависимости от предметной области информационные системы могут весьма значительно различаться по своим функциям, архитектуре, реализации. Однако можно выделить ряд свойств, которые являются общими.

- Информационные системы предназначены организации и поддержке информационного процесса, поэтому в основе любой из них лежит среда хранения и доступа к информации.
- Информационные системы ориентированы на конечного пользователя, не обладающего высокой квалификацией в области вычислительной техники. Поэтому клиентские приложения информационной системы должны обладать простым, удобным, легко осваиваемым интерфейсом.

Таким образом, при разработке информационной системы приходится решать две основные задачи:

- разработка базы данных, предназначенной для хранения информации;
- разработка графического интерфейса пользователя клиентских приложений.

Подавляющее большинство информационных систем работает в режиме диалога с пользователем.

В наиболее общем случае типовые программные компоненты, входящие в состав информационной системы, реализуют:

- диалоговый ввод-вывод;
- логику диалога;
- прикладную логику обработки данных;
- логику управления данными;
- операции манипулирования файлами и (или) базами данных.

Классификация информационных систем

Информационные системы классифицируются по разным признакам.

Классификация по масштабу

По масштабу информационные системы подразделяются на следующие группы (рис. 1):

- одиночные;
- групповые;
- корпоративные.

Одиночные информационные системы

Одиночные информационные системы реализуются, как правило, на автономном персональном компьютере (сеть не используется). Такая система может содержать несколько простых приложений, связанных общим информационным фондом, и рассчитана на работу одного пользователя или группы пользователей, разделяющих по времени одно рабочее место. Подобные приложения создаются с помощью так называемых настольных, или локальных, систем управления базами данных (СУБД). Среди локальных СУБД наиболее известными являются Clarion, Clipper, FoxPro, Paradox, dBase и Microsoft Access.

Групповые информационные системы

Групповые информационные системы ориентированы на коллективное использование информации членами рабочей группы и чаще всего строятся на базе локальной вычислительной сети. При разработке таких приложений используются серверы баз данных (называемые также SQL (Structured Query Language – структурированный язык запросов)-серверами) для рабочих групп. Существует довольно большое количество различных SQL-серверов как коммерческих, так и свободно распространяемых. Среди них наиболее известны такие серверы баз данных, как Oracle, DB2, Microsoft SQL Server, InterBase, Sybase, Informix.

Корпоративные информационные системы

Корпоративные информационные системы являются развитием систем для рабочих групп, они ориентированы на крупные компании и могут поддерживать территориально разнесенные узлы или сети. В основном они имеют иерархическую структуру из нескольких уровней. Для таких систем характерна архитектура клиент-сервер со специализацией серверов или же многоуровневая архитектура. При разработке таких систем могут использоваться те же серверы баз данных, что и при разработке групповых информационных систем. Однако в крупных информационных системах наибольшее распространение получили серверы Oracle, DB2 и Microsoft SQL Server.

Классификация по сфере применения

По сфере применения информационные системы обычно подразделяются на четыре группы (рис. 2):

- системы обработки транзакций (протоколов);
- системы поддержки принятия решений;
- информационно-справочные системы;
- офисные информационные системы.

Системы обработки транзакций, в свою очередь, по оперативности обработки данных разделяются на пакетные информационные системы и оперативные информационные системы. В информационных системах организационного управления преобладает режим оперативной обработки транзакций (OnLine Transaction Processing, OLTP) для отражения актуального состояния предметной области в любой момент времени, а пакетная обработка занимает весьма ограниченную часть. Для систем OLTP характерен регулярный (возможно, интенсивный) поток довольно простых транзакций, играющих роль заказов, платежей, запросов и т.п. Важными требованиями для них являются:

- высокая производительность обработки транзакций;

гарантированная доставка информации при удаленном доступе к БД по телекоммуникациям.

Рис. 2. Деление информационных систем по сфере применения.

Системы поддержки принятия решений (Decision Support System, DSS) представляют собой другой тип информационных систем, в которых с помощью довольно сложных запросов производится отбор и анализ данных в различных разрезах: временных, географических, по другим показателям.

Обширный класс **информационно-справочных систем** основан на гипертекстовых документах и мультимедиа. Наибольшее развитие такие информационные системы получили в Интернете.

Класс **офисных информационных систем** нацелен на перевод бумажных документов в электронный вид, автоматизацию делопроизводства и управление документооборотом.

Классификация по способу организации

По способу организации групповые и корпоративные информационные системы подразделяются на следующие классы (рис. 3):

- системы на основе архитектуры файл-сервер;
- системы на основе архитектуры клиент-сервер;
- системы на основе многоуровневой архитектуры;
- системы на основе Интернет/интранет-технологий.

Рис. 3. Деление информационных систем по способу организации.

В любой информационной системе можно выделить необходимые функциональные компоненты (табл. 1), которые помогают понять ограничения различных архитектур информационных систем. Рассмотрим более подробно особенности вариантов построения информационных приложений.

Таблица 1.1. Типовые функциональные компоненты информационной системы

Обозначение	Наименование	Характеристика
PS	Presentation Services (средства представления)	Обслуживает пользовательский ввод и отображает то, что сообщает ему компонент логики представления (PL), с использованием соответствующей программной поддержки
PL	Presentation Logic (логика представления)	Управляет взаимодействием между пользователем и ЭВМ. Обрабатывает действия пользователя при выборе команды в меню, щелчке на кнопке или выборе пункта в списке
BL	Business Logic (прикладная логика)	Набор правил для принятия решений, вычислений и операций, которые должно выполнить приложение
DL	Data Logic (логика управления данными)	Операции с базой данных (реализуемые SQL-операторами), которые нужно выполнить для реализации прикладной логики управления данными
DS	Data Services (операции с базой данных)	Действия СУБД, реализующие логику управления данными, такие как манипулирование данными, определение данных, фиксация или откат транзакций и т. п. СУБД обычно компилирует SQL-предложения
FS	File Services (файловые	Дисковые операции чтения и записи данных для СУБД и других компонентов. Обычно являются

	операции)	функциями операционной системы (ОС)

Архитектура файл-сервер

В архитектуре файл-сервер сетевое разделение компонентов диалога PS и PL отсутствует, а компьютер используется для функций отображения, что облегчает построение графического интерфейса. Файл-сервер только извлекает данные из файлов, так что дополнительные пользователи и приложения лишь незначительно увеличивают нагрузку на центральный процессор.

Объектами разработки в файл-серверном приложении являются компоненты приложения, определяющие логику диалога PL, а также логику обработки BL и управления данными DL. Разработанное приложение реализуется либо в виде законченного загрузочного модуля, либо в виде специального кода для интерпретации.

Однако такая архитектура имеет существенный недостаток: при выполнении некоторых запросов к базе данных клиенту могут передаваться большие объемы данных, загружая сеть и приводя к непредсказуемости времени реакции. Значительный сетевой трафик особенно сказывается при организации удаленного доступа к базам данных на файл-сервере через низкоскоростные каналы связи. Одним из вариантов устранения данного недостатка является удаленное управление файл-серверным приложением в сети. При этом в локальной сети размещается сервер приложений, совмещенный с телекоммуникационным сервером (обычно называемым сервером доступа), в среде которого выполняются обычные файл-серверные приложения. Особенность состоит в том, что диалоговый ввод-вывод поступает от удаленных клиентов через телекоммуникации. Приложения не должны быть слишком сложными, иначе велика вероятность перегрузки сервера или же нужна очень мощная платформа для сервера приложений.

Архитектура клиент-сервер

Архитектура клиент-сервер предназначена для разрешения проблем файл-серверных приложений путем разделения компонентов приложения и размещения их там, где они будут функционировать наиболее эффективно. Особенностью архитектуры клиент-сервер является наличие выделенных серверов баз данных, понимающих запросы на языке структурированных запросов (Structured Query Language, SQL) и выполняющих поиск, сортировку и агрегирование информации.

Отличительная черта серверов БД — наличие справочника данных, в котором записаны структура БД, ограничения целостности данных, форматы и даже серверные процедуры обработки данных по вызову или по событиям в программе. Объектами разработки в таких приложениях, помимо диалога и логики обработки, являются, прежде всего, реляционная модель данных и связанный с ней набор SQL-операторов для типовых запросов к базе данных.

Большинство конфигураций клиент-сервер использует двухуровневую модель, в которой клиент обращается к услугам сервера. Предполагается, что диалоговые компоненты PS и PL размещаются на клиенте, что позволяет реализовать графический интерфейс. Компоненты управления данными DS и FS размещаются на сервере, а диалог (PS, PL) и логика (BL, DL) — на клиенте. В двухуровневом определении архитектуры клиент-сервер используется именно этот вариант: приложение работает на клиенте, СУБД — на сервере (рис. 4).

Рис. 4. Классический вариант клиент-серверной системы.

Поскольку эта схема предъявляет наименьшие требования к серверу, она обладает наилучшей масштабируемостью. Однако сложные приложения, активно взаимодействующие с БД, могут жестко загрузить как клиент, так и сеть. Результаты SQL-запроса должны вернуться клиенту для обработки, потому что там реализована логика принятия решения. Такая схема приводит к дополнительному усложнению администрирования приложений, разбросанных по различным клиентским узлам.

Для сокращения нагрузки на сеть и упрощения администрирования приложений компонент BL можно разместить на сервере. При этом вся логика принятия решений оформляется в виде хранимых процедур и выполняется на сервере БД.

Хранимая процедура — процедура с SQL-операторами для доступа к БД, вызываемая по имени с передачей требуемых параметров и выполняемая на сервере БД. Хранимые процедуры могут компилироваться, что повышает скорость их выполнения и сокращает нагрузку на сервер.

Хранимые процедуры улучшают целостность приложений и БД, гарантируют актуальность коллективных операций и вычислений. Улучшается сопровождение таких процедур, а также безопасность (нет прямого доступа к данным).

Создание архитектуры клиент-сервер возможно и на основе многотерминальной системы. В этом случае в многозадачной среде сервера приложений выполняются программы пользователей, а клиентские узлы вырождены и представлены терминалами. Подобная схема информационной системы характерна для Unix.

Двухуровневые схемы архитектуры клиент-сервер могут привести к некоторым проблемам в сложных информационных приложениях с множеством пользователей и запутанной логикой. Решением этих проблем может стать применение многоуровневой архитектуры.

Многоуровневая архитектура

Многоуровневая архитектура стала развитием архитектуры клиент-сервер и в своей классической форме состоит из трех уровней:

- нижний уровень представляет собой приложения клиентов, выделенные для выполнения функций и логики представлений PS и PL и имеющие программный интерфейс для вызова приложения на среднем уровне;
- средний уровень представляет собой сервер приложений, на котором выполняется прикладная логика BL и с которого логика обработки данных DL выполняет операции с базой данных DS;
- верхний уровень представляет собой удаленный специализированный сервер базы данных, выделенный для услуг обработки данных DS и файловых операций FS (без использования хранимых процедур).

Подобную концепцию обработки данных пропагандируют, в частности, фирмы Oracle, Sun, Borland и др.

Трехуровневая архитектура позволяет еще больше сбалансировать нагрузку на разные узлы и сеть, а также способствует специализации инструментов для разработки приложений и устраняет недостатки двухуровневой модели клиент-сервер.

Централизация логики приложения упрощает администрирование и сопровождение. Четко разделяются платформы и инструменты для реализации интерфейса и прикладной логики, что позволяет с наибольшей отдачей реализовывать их специалистам узкого профиля. Наконец, изменения прикладной логики не затрагивают интерфейса, и наоборот. Но поскольку границы между компонентами PL, BL и DL размыты, прикладная логика может реализовываться на всех трех уровнях. Сервер приложений с помощью монитора транзакций обеспечивает интерфейс с клиентами и другими серверами, может управлять транзакциями и гарантировать целостность распределенной базы данных. Средства удаленного вызова процедур наиболее соответствуют идее распределенных вычислений: они обеспечивают из любого узла сети вызов прикладной процедуры, расположенной на другом узле, передачу параметров, удаленную обработку и возврат результатов. С ростом систем клиент-сервер необходимость трех уровней становится все более очевидной. Продукты для трехуровневой архитектуры, так называемые мониторы транзакций, являются

относительно новыми. Эти инструменты в основном ориентированы на среду Unix, однако прикладные серверы можно строить на базе Microsoft Windows NT с вызовом удаленных процедур для организации связи клиентов с сервером приложений. На практике в локальной сети могут использоваться смешанные архитектуры (двухуровневые и трехуровневые) с одним и тем же сервером базы данных. С учетом глобальных связей архитектура может иметь больше трех уровней.

Таким образом, многоуровневая архитектура распределенных приложений позволяет повысить эффективность работы корпоративной информационной системы и оптимизировать распределение ее программно-аппаратных ресурсов. Но пока на российском рынке по-прежнему доминирует архитектура клиент-сервер.

Интернет/интранет-технологии

В развитии Интернет/интранет-технологий основной акцент пока что делается на разработке инструментальных программных средств. В то же время наблюдается отсутствие развитых средств разработки приложений, работающих с базами данных. Компромиссным решением для создания удобных и простых в использовании и сопровождении информационных систем, эффективно работающих с базами данных, стало объединение Интернет/интранет-технологий с многоуровневой архитектурой. При этом структура информационного приложения приобретает следующий вид:

браузер — сервер приложений — сервер баз данных — сервер динамических страниц — веб-сервер.

Благодаря интеграции Интернет/интранет-технологий и архитектуры клиент-сервер, процесс внедрения и сопровождения корпоративной информационной системы существенно упрощается при сохранении достаточно высокой эффективности и простоты совместного использования информации.

Требования, предъявляемые к информационным системам

Информационная система должна соответствовать требованиям гибкости, надежности, эффективности и безопасности.

Гибкость

Гибкость, способность к адаптации и дальнейшему развитию подразумевает возможность приспособления информационной системы к новым условиям, новым потребностям предприятия. Выполнение этих условий возможно, если на этапе разработки информационной системы использовались общепринятые средства и методы документирования, так что по прошествии определенного времени сохранится возможность разобраться в структуре системы и внести в нее соответствующие изменения, даже если все разработчики или их часть по каким-либо причинам не смогут продолжить работу.

Любая информационная система рано или поздно морально устареет, и станет вопрос о ее модернизации или полной замене. Разработчики информационных систем, как правило, не являются специалистами в прикладной области, для которой разрабатывается система. Участие в модернизации или создании новой системы той же группы проектировщиков существенно сократит сроки модернизации. Вместе с тем возникает риск применения устаревших решений при модернизации системы. Рекомендация в таком случае одна — внимательнее относиться к подбору разработчиков информационных систем.

Надежность

Надежность информационной системы подразумевает ее функционирование без искажения информации, потери данных по «техническим причинам». Требование надежности обеспечивается созданием резервных копий хранимой информации, выполнения операций протоколирования, поддержанием качества каналов связи и

физических носителей информации, использованием современных программных и аппаратных средств. Сюда же следует отнести защиту от случайных потерь информации в силу недостаточной квалификации персонала.

Эффективность

Система является эффективной, если с учетом выделенных ей ресурсов она позволяет решать возложенные на нее задачи в минимальные сроки. В любом случае оценка эффективности будет производиться заказчиком, исходя из вложенных в разработку средств и соответствия представленной информационной системы его ожиданиям.

Негативной оценки эффективности информационной системы со стороны заказчика можно избежать, если представители заказчика будут привлекаться к проектированию системы на всех его стадиях. Такой подход позволяет многим конечным пользователям уже на этапе проектирования адаптироваться к изменениям условий работы, которые иначе были бы приняты враждебно.

Активное сотрудничество с заказчиком с ранних этапов проектирования позволяет уточнить потребности заказчика. Часто встречается ситуация, когда заказчик чего-то хочет, но сам не знает чего именно. Чем раньше будут учтены дополнения заказчика, тем с меньшими затратами и в более короткие сроки система будет создана.

Кроме того, заказчик, не являясь специалистом в области разработки информационных систем, может не знать о новых информационных технологиях. Контакты с заказчиком во время разработки для него информационной системы могут подтолкнуть заказчика к модернизации его аппаратных средств, применению новых методов ведения бизнеса, что отвечает потребностям, как заказчика, так и проектировщика. Заказчик получает рост эффективности своего предприятия, проектировщик — расширение возможностей, применяемых при проектировании информационной системы.

Эффективность системы обеспечивается оптимизацией данных и методов их обработки, применением оригинальных разработок, идей, методов проектирования.

Не следует забывать и о том, что работать с системой придется обычным людям, являющимся специалистами в своей предметной области, но зачастую обладающим весьма средними навыками в работе с компьютерами. Интерфейс информационных систем должен быть им интуитивно понятен. В свою очередь, разработчик-программист должен понимать характер выполняемых конечным пользователем операций. Рекомендациями в этом случае могут служить повышение эффективности управления разработкой информационных систем, улучшение информированности разработчиков о предметной области.

Безопасность

Под безопасностью, прежде всего, подразумевается свойство системы, в силу которого посторонние лица не имеют доступа к информационным ресурсам организации, кроме тех, которые для них предназначены. Защита информации от постороннего доступа обеспечивается управлением доступа к ресурсам системы, использованием современных программных средств защиты информации. В крупных организациях целесообразно создавать подразделения, основным направлением деятельности которых было бы обеспечение информационной безопасности, в менее крупных организациях назначать сотрудника, ответственного за данный участок работы.

Помимо злого умысла, при обеспечении безопасности информационных систем приходится сталкиваться еще с несколькими факторами. В частности, современные информационные системы являются достаточно сложными программными продуктами. При их проектировании с высокой вероятностью возможны ошибки, вызванные большим объемом программного кода, несовершенством компиляторов, человеческим фактором, несовместимостью с используемыми программами сторонних разработчиков в случае модификации этих программ и т.п. Поэтому за фазой разработки информационной

системы неизбежно следует фаза ее сопровождения в процессе эксплуатации, в которой происходит выявление скрытых ошибок и их исправление.

Требование безопасности обеспечивается современными средствами разработки информационных систем, современной аппаратурой, методами защиты информации, применением паролей и протоколированием, постоянным мониторингом состояния безопасности операционных систем и средств их защиты.

Лекция №2

Содержание лекции

Жизненный цикл информационных систем

Любую организацию можно рассматривать как совокупность взаимодействующих элементов (подразделений), каждый из которых может иметь свою, достаточно сложную, структуру. Взаимосвязи между подразделениями тоже достаточно сложны. В общем случае можно выделить три вида связей между подразделениями предприятия:

- **функциональные связи** — каждое подразделение выполняет определенные виды работ в рамках единого бизнес-процесса;
- **информационные связи** — подразделения обмениваются информацией (документами, факсами, письменными и устными распоряжениями и т. п.);
- **внешние связи** — некоторые подразделения взаимодействуют с внешними системами, причем их взаимодействие также может быть как информационным, так и функциональным.

В общем случае процесс разработки информационной системы может быть рассмотрен с двух точек зрения:

- по содержанию действий разработчиков (групп разработчиков) — в данном случае рассматривается статический аспект процесса разработки, описываемый в терминах основных потоков работ (исполнители, действия, последовательность действий и т. п.);
- по времени или по стадиям жизненного цикла разрабатываемой системы — в данном случае рассматривается динамическая организация процесса разработки, описываемая в терминах циклов, стадий, итераций и этапов.

Общие сведения об управлении проектами

Информационная система предприятия разрабатывается как некоторый проект. Многие особенности управления проектами и фазы разработки проекта (фазы жизненного цикла) являются общими, не зависящими не только от предметной области, но и от характера проекта (не важно, инженерный это проект или экономический). Поэтому имеет смысл вначале рассмотреть ряд общих вопросов управления проектами.

Проект — это ограниченное по времени целенаправленное изменение отдельной системы с изначально четко определенными целями, достижение которых означает завершение проекта, а также с установленными требованиями к срокам, результатам, риску, рамкам расходования средств и ресурсов, организационной структуре.

Можно выделить следующие основные отличительные признаки проекта как объекта управления:

- изменчивость — целенаправленный перевод системы из существующего в некоторое желаемое состояние, описываемое в терминах целей проекта;
- ограниченность конечной цели;
- ограниченность продолжительности;
- ограниченность бюджета;
- ограниченность требуемых ресурсов;
- новизна для предприятия, для которого реализуется проект;
- комплексность — наличие большого числа факторов, прямо или косвенно влияющих на прогресс и результаты проекта;
- правовое и организационное обеспечение — создание специфической организационной структуры на время реализации проекта.

Рассматривая планирование проектов и управление ими, необходимо четко осознавать, что речь идет об управлении неким динамическим объектом. Поэтому система управления проектом должна быть достаточно гибкой, чтобы допускать возможность модификации без глобальных изменений в рабочей программе.

В системном плане проект может быть представлен «черным ящиком», на входе которого располагаются технические требования и условия финансирования, а на выходе — требуемый результат (рис. 1). Выполнение работ обеспечивается наличием необходимых ресурсов:

- материалов;
- оборудования;
- человеческих ресурсов.

Рис. 1. Представление проекта в виде «черного ящика».

Для обоснования осуществимости, для анализа хода его реализации, а так же для заключительной оценки проекта существует ряд характеристик проекта. К важнейшим из них относятся следующие технико-экономические показатели:

- объем работ;
- сроки выполнения;
- себестоимость;
- экономическая эффективность, обеспечиваемая реализацией проекта;
- социальная и общественная значимость проекта.

Классификация проектов

Проекты могут быть классифицированы по самым различным признакам. Отметим основные из них.

- **Класс проекта** определяется по составу и структуре проекта. Обычно различают:
  - монопроект (отдельный проект, который может быть любого типа, вида и масштаба);
  - мультипроект (комплексный проект, состоящий из ряда монопроектов.
- **Тип проекта** определяется по основным сферам деятельности, в которых осуществляется проект. Можно выделить пять основных типов проекта:
  - технический;
  - организационный;
  - экономический;
  - социальный;
  - смешанный.
- **Масштаб проекта** определяется размером бюджета и количеством участников. Бывают большие и малые проекты. Масштабы проектов рассматривают в конкретной форме — отраслевые, корпоративные, ведомственные проекты, проекты одного предприятия.

Основные фазы проектирования информационной системы

Каждый проект, независимо от сложности и объема работ, необходимых для его выполнения, проходит в своем развитии определенные состояния: от состояния, когда «проекта еще нет», до состояния, когда «проекта уже нет».

Можно выделить следующие фазы развития информационной системы:

- формирование концепции;
- подготовка технического задания;
- проектирование;
- разработка;

- ввод системы в эксплуатацию.

Рассмотрим каждую из них более подробно.

Концептуальная фаза

Главным содержанием работ на концептуальной фазе является определение проекта, разработка его концепции, включающая:

- формирование идеи, постановку целей;
- формирование ключевой команды проекта;
- изучение мотивации и требований заказчика и других участников;
- сбор исходных данных и анализ существующего состояния;
- определение основных требований и ограничений, требуемых материальных, финансовых и трудовых ресурсов;
- сравнительную оценку альтернатив;
- представление предложений, их экспертизу и утверждение.

Подготовка технического предложения

Главным содержанием фазы подготовки технического предложения является уточнение технического предложения в ходе переговоров с заказчиком о заключении контракта. Общее содержание работ этой фазы:

- разработка основного содержания, базовой структуры проекта;
- разработка и утверждение технического задания;
- планирование, декомпозиция базовой структуры модели проекта;
- составление сметы и бюджета проекта, определение потребности в ресурсах;
- разработка календарных планов и укрупненных графиков работ;
- подписание контракта с заказчиком;
- ввод в действие средств коммуникации участников проекта и средств контроля за ходом работ.

Проектирование

На фазе проектирования определяются подсистемы, их взаимосвязи, выбираются наиболее эффективные способы выполнения проекта и использования ресурсов. Характерные работы этой фазы:

- выполнение базовых проектных работ;
- разработка частных технических заданий;
- выполнение концептуального проектирования;
- составление технических спецификаций и инструкций;
- представление проектной разработки, экспертиза и утверждение.

Разработка

На фазе разработки производятся координация и оперативный контроль работ по проекту, осуществляется изготовление подсистем, их объединение и тестирование. Основное содержание:

- выполнение работ по разработке программного обеспечения;
- подготовка к внедрению системы;
- контроль и регулирование основных показателей проекта.

Ввод системы в эксплуатацию

На фазе ввода системы в эксплуатацию проводятся испытания, идет опытная эксплуатация системы в реальных условиях, ведутся переговоры о результатах выполнения проекта и о возможных новых контрактах. Основные виды работ:

- комплексные испытания;
- подготовка кадров для эксплуатации создаваемой системы;
- подготовка рабочей документации, сдача системы заказчику и ввод ее в эксплуатацию;
- сопровождение, поддержка, сервисное обслуживание;
- оценка результатов проекта и подготовка итоговых документов;
- разрешение конфликтных ситуаций и закрытие работ по проекту;
- накопление опытных данных для последующих проектов, анализ опыта, состояния, определение направлений развития.

Следует иметь в виду, что на обнаружение ошибок, допущенных на стадии системного проектирования, расходуется примерно в два раза больше времени, чем на последующих фазах, а их исправление обходится в пять раз дороже. Поэтому на начальных стадиях проекта разработку следует выполнять особенно тщательно. Наиболее часто на начальных фазах допускаются следующие ошибки:

- ошибки в определении интересов заказчика;
- концентрация на маловажных, сторонних интересах;
- неправильная интерпретация исходной задачи;
- неправильное или недостаточное понимание деталей;
- неполнота функциональных спецификаций (системных требований);
- ошибки в определении требуемых ресурсов и сроков;
- редкая проверка на согласованность этапов и отсутствие контроля со стороны заказчика (нет привлечения заказчика).

Процессы, протекающие на протяжении жизненного цикла информационной системы

Жизненный цикл информационной системы представляет собой непрерывный процесс, начинающийся с момента принятия решения о создании информационной системы и заканчивающийся в момент полного изъятия ее из эксплуатации.

Существует международный стандарт, регламентирующий жизненный цикл информационных систем — ISO/IEC 12207.

ПРИМЕЧАНИЕ

ISO расшифровывается как International Organization of Standardization (международная организация по стандартизации), IEC — как International Electrotechnical Commission (международная комиссия по электротехнике).

Стандарт ISO/IEC 12207 определяет структуру жизненного цикла, включая процессы, действия и задачи, которые должны быть выполнены во время создания информационной системы. Согласно данному стандарту, структура жизненного цикла основывается на трех группах процессов:

- основные процессы жизненного цикла (приобретение, поставка, разработка, эксплуатация, сопровождение);
- вспомогательные процессы, обеспечивающие выполнение основных процессов (документирование, управление конфигурацией, обеспечение качества, верификация, аттестация, оценка, аудит, разрешение проблем);
- организационные процессы (управление проектами, создание инфраструктуры проекта, определение, оценка и улучшение самого жизненного цикла, обучение).

Рассмотрим каждую из указанных групп более подробно.

Основные процессы жизненного цикла

Среди основных процессов жизненного цикла наибольшую важность имеют три: разработка, эксплуатация и сопровождение.

Разработка

Разработка информационной системы включает в себя все работы по созданию информационного программного обеспечения и его компонентов в соответствии с заданными требованиями. Разработка информационного программного обеспечения также включает:

- оформление проектной и эксплуатационной документации;
- подготовку материалов, необходимых для тестирования разработанных программных продуктов;
- разработку материалов, необходимых для обучения персонала.

Разработка является одним из важнейших процессов жизненного цикла информационной системы и, как правило, включает в себя стратегическое планирование, анализ, проектирование и реализацию (программирование).

Эксплуатация

Эксплуатационные работы можно подразделить на подготовительные и основные. К подготовительным относятся:

- конфигурирование базы данных и рабочих мест пользователей;
- обеспечение пользователей эксплуатационной документацией;
- обучение персонала.

Основные эксплуатационные работы включают:

- непосредственно эксплуатацию;
- локализацию проблем и устранение причин их возникновения;
- модификацию программного обеспечения;
- подготовку предложений по совершенствованию системы;
- развитие и модернизацию системы.

Сопровождение

Службы технической поддержки играют весьма заметную роль в жизни любой корпоративной информационной системы. Наличие квалифицированного технического обслуживания на этапе эксплуатации информационной системы является необходимым условием решения поставленных перед ней задач, причем ошибки обслуживающего персонала могут приводить к явным или скрытым финансовым потерям, сопоставимым со стоимостью самой информационной системы.

Основными предварительными действиями при подготовке к организации технического обслуживания информационной системы являются:

- выделение наиболее ответственных узлов системы и определение для них критичности простоя (это позволит выделить наиболее критичные составляющие информационной системы и оптимизировать распределение ресурсов для технического обслуживания);
- определение задач технического обслуживания и их разделение на внутренние, решаемые силами обслуживающего подразделения, и внешние, решаемые специализированными сервисными организациями (таким образом, производится четкое определение круга исполняемых функций и разделение ответственности);
- проведение анализа имеющихся внутренних и внешних ресурсов, необходимых для организации технического обслуживания в рамках описанных задач и разделения компетенции (основные критерии для анализа: наличие гарантии на оборудование, состояние ремонтного фонда, квалификация персонала);
- подготовка плана организации технического обслуживания, в котором необходимо определить этапы исполняемых действий, сроки их исполнения, затраты на этапах, ответственность исполнителей.

Обеспечение качественного технического обслуживания информационной системы требует привлечения специалистов высокой квалификации, которые в состоянии не только решать каждодневные задачи администрирования, но и быстро восстанавливать работоспособность системы при сбоях.

Вспомогательные процессы жизненного цикла

Среди вспомогательных процессов одно из главных мест занимает **управление конфигурацией**. Это тот вспомогательный процесс, который поддерживает основные процессы жизненного цикла информационной системы, прежде всего процессы разработки и сопровождения.

При разработке проектов сложных информационных систем, состоящих из многих компонентов, каждый из которых может разрабатываться независимо и, следовательно, иметь несколько вариантов реализации и/или несколько версий одной реализации, возникает проблема учета их связей и функций, создания единой структуры и обеспечения развития всей системы. **Управление конфигурацией** позволяет организовывать, систематически учитывать и контролировать внесение изменений в различные компоненты информационной системы на всех стадиях ее жизненного цикла.

Организационные процессы

Управление проектом связано с вопросами планирования и организации работ, создания коллективов разработчиков и контроля за сроками и качеством выполняемых работ. Техническое и организационное обеспечение проекта включает:

- выбор методов и инструментальных средств для реализации проекта;
- определение методов описания промежуточных состояний разработки;
- разработку методов и средств испытаний созданного программного обеспечения;
- обучение персонала.

Обеспечение качества проекта связано с проблемами верификации, проверки и тестирования компонентов информационной системы.

Верификация — это процесс определения соответствия текущего состояния разработки, достигнутого на данном этапе, требованиям этого этапа.

Проверка — это процесс определения соответствия параметров разработки исходным требованиям. Проверка отчасти совпадает с тестированием, которое проводится для определения различий между действительными и ожидавшимися результатами и оценки соответствия характеристик информационной системы исходным требованиям.

Структура жизненного цикла информационной системы

Полный жизненный цикл информационной системы включает в себя, как правило, стратегическое планирование, анализ, проектирование, реализацию, внедрение и эксплуатацию. В общем случае жизненный цикл можно, в свою очередь, разбить на ряд стадий. Рассмотрим один из вариантов деления, предлагаемый корпорацией Rational Software — одной из ведущих фирм на рынке программного обеспечения средств разработки информационных систем (среди которых большой популярностью заслуженно пользуется универсальное CASE-средство Rational Rose).

ПРИМЕЧАНИЕ

Термин CASE (Computer Aided Software/System Engineering) используется в настоящее время в весьма широком смысле. Первоначальное значение термина CASE ограничивалось лишь вопросами автоматизации разработки программного обеспечения. Однако в дальнейшем значение этого термина расширилось. Теперь под термином «CASE-средства» понимаются программные средства, поддерживающие процессы создания и сопровождения информационных систем, включая анализ и формулировку требований, проектирование прикладного программного обеспечения и баз данных, генерацию кода, тестирование, документирование, обеспечение качества, конфигурационное управление и управление проектом, а также другие процессы.

Согласно методологии, предлагаемой Rational Software, жизненный цикл информационной системы подразделяется на четыре стадии:

- начало;
- уточнение;
- конструирование;
- передача в эксплуатацию.

Границы каждой стадии определены некоторыми моментами времени, в которые необходимо принимать определенные критические решения и, следовательно, достигать определенных ключевых целей.

Начальная стадия

На начальной стадии устанавливается область применения системы и определяются граничные условия. Для этого необходимо идентифицировать все внешние объекты, с которыми должна взаимодействовать разрабатываемая система, и определить характер этого взаимодействия. На начальной стадии идентифицируются все

функциональные возможности системы, и производится описание наиболее существенных из них.

Деловое применение начальной стадии включает:

- критерии успеха разработки;
- оценку риска;
- оценку ресурсов, необходимых для выполнения разработки;
- календарный план с указанием сроков завершения основных этапов.

Стадия уточнения

На стадии уточнения проводится анализ прикладной области, разрабатывается архитектурная основа информационной системы.

В конце стадии уточнения проводится анализ архитектурных решений и способов устранения главных факторов риска в проекте.

Стадия конструирования

На стадии конструирования разрабатывается законченное изделие, готовое к передаче пользователю. По окончании этой стадии определяется работоспособность разработанного программного обеспечения.

Стадия передачи в эксплуатацию

На стадии передачи в эксплуатацию разработанное программное обеспечение передается пользователям. При эксплуатации разработанной системы в реальных условиях часто возникают различного рода проблемы, которые требуют дополнительных работ по внесению корректив в разработанный продукт. В конце стадии передачи в эксплуатацию необходимо определить, достигнуты цели разработки или нет.

Лекция №3

Содержание лекции

Жизненный цикл информационных систем

Модели жизненного цикла информационной системы

Моделью жизненного цикла информационной системы будем называть некоторую структуру, определяющую последовательность осуществления процессов, действий и задач, выполняемых на протяжении жизненного цикла информационной системы, а также взаимосвязи между этими процессами, действиями и задачами.

В стандарте ISO/IEC 12207 не конкретизируются в деталях методы выполнения действий и решения задач, входящих в процессы жизненного цикла информационной системы, а лишь описываются структуры этих процессов. Это вполне понятно, так как регламенты стандарта являются общими для любых моделей жизненного цикла, методологий и технологий разработки. Модель же жизненного цикла зависит от специфики информационной системы и условий, в которых она создается и функционирует.

К настоящему времени наибольшее распространение получили две основные (классические) модели жизненного цикла:

- каскадная модель, иногда также называемая моделью водопада (waterfall);
- спиральная модель.

Каскадная модель жизненного цикла информационной системы

Каскадная модель предусматривает последовательную организацию работ. При этом основной особенностью является разбиение всей разработки на этапы, причем переход с одного этапа на следующий происходит только после того, как полностью завершены все работы на предыдущем этапе. Каждый этап завершается выпуском полного комплекта документации, достаточной для того, чтобы разработка могла быть продолжена другой командой разработчиков.

Основные этапы разработки по каскадной модели

Можно выделить следующий ряд устойчивых этапов разработки, практически не зависящих от предметной области (рис. 1):

- анализ требований заказчика;
- проектирование;

- разработка;
- тестирование и опытная эксплуатация;
- сдача готового продукта.

Рис. 1. Каскадная модель разработки.

Результатом, получаемым на первом этапе, является техническое задание (задание на разработку), согласованное со всеми заинтересованными сторонами.

Результатом второго этапа является комплект проектной документации, содержащей все необходимые данные для реализации проекта.

Результатом выполнения третьего этапа является готовый программный продукт.

Результатом выполнения четвертого этапа являются различного рода скрытые недостатки, проявляющиеся в реальных условиях работы информационной системы.

Главная задача пятого этапа — убедить заказчика, что все его требования выполнены в полной мере.

В действительности жизненный цикл самой системы существенно сложнее и длиннее. Он может включать в себя произвольное число циклов уточнения, изменения и дополнения уже принятых и реализованных проектных решений. В этих циклах происходит развитие информационной системы и модернизация отдельных ее компонентов.

Основные достоинства каскадной модели

Рассмотрим ее основные достоинства.

- На каждом этапе формируется законченный набор проектной документации, отвечающей критериям полноты и согласованности. На заключительных этапах также разрабатывается пользовательская документация, охватывающая все предусмотренные стандартами виды обеспечения информационной системы (организационное, методическое, информационное, программное, аппаратное).
- Выполняемые в логичной последовательности этапы работ позволяют планировать сроки завершения и соответствующие затраты.

Каскадный подход хорошо зарекомендовал себя при разработке определенных информационных систем. Имеются в виду системы, для которых в самом начале разработки можно достаточно точно и полно сформулировать все требования, с тем, чтобы предоставить разработчикам свободу выбора реализации, наилучшей с технической точки зрения. К таким информационным системам, в частности, относятся сложные расчетные системы, системы реального времени.

Недостатки каскадной модели

Перечень недостатков каскадной модели при ее использовании для разработки информационных систем достаточно обширен:

- существенная задержка в получении результатов;
- ошибки и недоработки на любом из этапов проявляются, как правило, на последующих этапах работ, что приводит к необходимости возврата назад;
- сложность параллельного ведения работ по проекту;
- чрезмерная информационная перенасыщенность каждого из этапов;
- сложность управления проектом;

- высокий уровень риска и ненадежность инвестиций.

Задержка в получении результатов обычно считается главным недостатком каскадной схемы. Данный недостаток проявляется в основном в том, что из-за последовательного подхода к разработке согласование результатов с заинтересованными сторонами производится только после завершения очередного этапа работ.

Используемые при разработке информационной системы модели автоматизируемого объекта, отвечающие критериям внутренней согласованности и полноты, могут в силу различных причин устареть за время разработки (например, из-за внесения изменений в законодательство, колебания курса валют и т.п.). Это относится и к функциональной модели, и к информационной модели, и к проектам интерфейса пользователя, и к пользовательской документации.

Возврат на более ранние стадии. Данный недостаток каскадной модели, в общем-то, является одним из проявлений предыдущего. Возврат может служить причиной срыва графика работ и усложнения взаимоотношений между группами разработчиков, выполняющих отдельные этапы работы.

Самым же неприятным является то, что недоработки предыдущего этапа могут обнаруживаться не сразу на последующем этапе, а позднее (например, на стадии опытной эксплуатации могут проявиться ошибки в описании предметной области). Это означает, что часть проекта должна быть возвращена на начальный этап работы. Вообще, работа может быть возвращена с любого этапа на любой предыдущий этап, поэтому в реальности каскадная схема разработки выглядит так, как показано на рис. 2.

Рис. 2. Реальный процесс разработки по каскадной схеме.

Сложность параллельного ведения работ. Проблемы возникают вследствие того, что работа над проектом строится в виде цепочки последовательных шагов. Причем даже в том случае, когда разработку некоторых частей проекта (подсистем) можно вести параллельно, при использовании каскадной схемы распараллеливание работ весьма затруднительно. Сложности параллельного ведения работ связаны с необходимостью постоянного согласования различных частей проекта. Чем сильнее взаимозависимость отдельных частей проекта, тем чаще и тщательнее должна выполняться синхронизация, тем сильнее зависят друг от друга группы разработчиков.

Информационная перенасыщенность. Проблема информационной перенасыщенности возникает вследствие сильной зависимости между различными группами разработчиков. Данная проблема заключается в том, что при внесении изменений в одну из частей проекта необходимо оповещать всех разработчиков, которые использовали или могли бы использовать эту часть в своей работе. Как следствие, объем документации по мере разработки проекта растет очень быстро, так что требуется все больше времени для составления документации и ознакомления с ней.

Следует также отметить, что, помимо изучения нового материала, не отпадает необходимость в изучении старой информации. Это связано с тем, что вполне вероятна ситуация, когда в процессе разработки изменяется состав группы разработчиков (этот процесс носит название **ротации кадров**).

Сложность управления проектом при использовании каскадной схемы в основном обусловлена строгой последовательностью стадий разработки и наличием сложных взаимосвязей между различными частями проекта.

Последовательность разработки проекта приводит к тому, что одни группы разработчиков должны ожидать результатов работы других команд. Поэтому требуется административное вмешательство для согласования сроков работы и состава передаваемой документации.

Высокий уровень риска. Чем сложнее проект, тем больше продолжительность каждого из этапов разработки и тем сложнее взаимосвязи между отдельными частями проекта, количество которых также увеличивается. Возврат на предыдущие стадии может быть связан не только с ошибками, но и с изменениями, произошедшими в предметной

области или в требованиях заказчика за время разработки. Причем возврат проекта на доработку вследствие этих причин не гарантирует, что предметная область снова не изменится к тому моменту, когда будет готова следующая версия проекта. Фактически это означает, что существует вероятность того, что процесс разработки «зациклится» и система никогда не дойдет до сдачи в эксплуатацию.

Поэтому можно утверждать, что сложные проекты, разрабатываемые по каскадной схеме, имеют повышенный уровень риска.

Спиральная модель жизненного цикла

Спиральная модель, в отличие от каскадной, предполагает итерационный процесс разработки информационной системы. При этом возрастает значение начальных этапов жизненного цикла, таких как анализ и проектирование. На этих этапах проверяется и обосновывается реализуемость технических решений путем создания прототипов.

Итерации

Каждая итерация представляет собой законченный цикл разработки, приводящий к выпуску внутренней или внешней версии изделия (или подмножества конечного продукта), которое совершенствуется от итерации к итерации, чтобы стать закопченной системой (рис. 3).

Таким образом, каждый виток спирали соответствует созданию фрагмента или версии программного изделия, на нем уточняются цели и характеристики проекта, определяется его качество, планируются работы на следующем витке спирали. На каждой итерации углубляются и последовательно конкретизируются детали проекта, в результате чего выбирается обоснованный вариант, который доводится до окончательной реализации.

Использование спиральной модели позволяет осуществлять переход на следующий этап выполнения проекта, не дожидаясь полного завершения текущего — недоделанную работу можно будет выполнить на следующей итерации. Главная задача каждой итерации — как можно быстрее создать работоспособный продукт, который можно показать пользователям системы. Таким образом, существенно упрощается процесс внесения уточнений и дополнений в проект.

Рис. 3. Спиральная модель жизненного цикла информационной системы.

Преимущества спиральной модели

Спиральный подход к разработке программного обеспечения позволяет преодолеть большинство недостатков каскадной модели и, кроме того, обеспечивает ряд дополнительных возможностей, делая процесс разработки более гибким.

Рассмотрим преимущества итерационного подхода более подробно.

- Итерационная разработка существенно упрощает внесение изменений в проект при изменении требований заказчика.
- При использовании спиральной модели отдельные элементы информационной системы интегрируются в единое целое постепенно. При итерационном подходе интеграция производится фактически непрерывно. Поскольку интеграция начинается с меньшего количества элементов, то возникает гораздо меньше проблем при ее проведении.
- Уменьшение уровня рисков. Данное преимущество является следствием предыдущего, так как риски обнаруживаются именно во время интеграции. Поэтому уровень рисков максимален в начале разработки проекта. По мере

продвижения разработки ожидаемый уровень рисков снижается. На рис. 4 приведены в сравнении графики зависимости уровня рисков от времени разработки для каскадного и спирального подходов.

Рис. 4. Зависимость рисков от времени разработки.

- Итерационная разработка обеспечивает большую гибкость в управлении проектом, давая возможность внесения тактических изменений в разрабатываемое изделие.
- Итерационный подход упрощает повторное использование компонентов. Это обусловлено тем, что гораздо проще выявить общие части проекта, когда они уже частично разработаны, чем пытаться выделить их в самом начале проекта. Анализ проекта после проведения нескольких начальных итераций позволяет выявить общие многократно используемые компоненты, которые на последующих итерациях будут совершенствоваться.
- Спиральная модель позволяет получить более надежную и устойчивую систему. Это связано с тем, что по мере развития системы ошибки и слабые места обнаруживаются и исправляются на каждой итерации.
- Итерационный подход дает возможность совершенствовать процесс разработки — анализ, проводимый в конце каждой итерации, позволяет проводить оценку того, что должно быть изменено в организации разработки, и улучшить ее на следующей итерации.

Недостатки спиральной модели

Основная проблема спирального цикла — определение момента перехода на следующий этап. Для ее решения необходимо ввести временные ограничения на каждый из этапов жизненного цикла. Иначе процесс разработки может превратиться в бесконечное совершенствование уже сделанного. При итерационном подходе полезно следовать принципу «лучшее — враг хорошего». Поэтому завершение итерации должно производиться строго в соответствии с планом, даже если не вся запланированная работа закончена.

Лекция №4

Содержание лекции

Методология и технология разработки информационных систем

Методология создания информационных систем заключается в организации процесса построения информационной системы и в управлении этим процессом для того, чтобы гарантировать выполнение требований, как к самой системе, так и к характеристикам процесса разработки.

Основными задачами, решение которых должна обеспечивать методология создания информационных систем (с помощью соответствующего набора инструментальных средств), являются:

- соответствие создаваемой информационной системы целям и задачам предприятия, а также предъявляемым к ней требованиям по автоматизации бизнес-процессов;
- гарантирование создания системы с заданными параметрами в течение заданного времени в рамках оговоренного заранее бюджета;
- простота сопровождения, модификации и расширения системы с целью обеспечения ее соответствия изменяющимся условиям работы предприятия;
- соответствие создаваемой корпоративной информационной системы требованиям открытости, переносимости и масштабируемости;

- возможность использования в создаваемой системе разработанных ранее и применяемых на предприятии средств информационных технологий (программного обеспечения, баз данных, средств вычислительной техники, телекоммуникаций).

Методологии, технологии и инструментальные средства проектирования (CASE-средства) составляют основу проекта любой информационной системы. Методология реализуется через конкретные технологии и поддерживающие их стандарты, методики и инструментальные средства, которые обеспечивают выполнение процессов жизненного цикла информационных систем.

Основное содержание технологии проектирования составляют технологические инструкции, состоящие из описания последовательности технологических операций, условий, в зависимости от которых выполняется та или иная операция, и описаний самих операций.

Технология проектирования может быть представлена как совокупность трех составляющих:

- заданной последовательности выполнения технологических операций проектирования;
- критериев и правил, используемых для оценки результатов выполнения технологических операций;
- графических и текстовых средств (нотаций), используемых для описания проектируемой системы.

Каждая технологическая операция должна обеспечиваться следующими материальными, информационными и людскими ресурсами:

- данными, полученными на предыдущей операции (или исходными данными), представленными в стандартном виде;
- методическими материалами, инструкциями, нормативами и стандартами;
- программными и техническими средствами;
- исполнителями.

Результаты выполнения операции должны представляться в некотором стандартном виде, обеспечивающем их адекватное восприятие при выполнении следующей технологической операции (на которой они будут использоваться в качестве исходных данных).

Можно сформулировать ряд общих требований, которым должна удовлетворять технология проектирования, разработки и сопровождения информационных систем:

- поддерживать полный жизненный цикл информационной системы;
- обеспечивать гарантированное достижение целей разработки системы с заданным качеством и в установленное время;

- обеспечивать возможность разделения (**декомпозиции**) крупных проектов на ряд подсистем — составных частей, разрабатываемых группами исполнителей ограниченной численности, с последующей интеграцией этих частей;

ПРИМЕЧАНИЕ

Декомпозиция проекта позволяет повысить эффективность работ. Подсистемы, на которые разбивается проект, должны быть слабо связаны поданным и функциям. Каждая подсистема разрабатывается отдельной группой разработчиков. При этом необходимо обеспечить координацию работ и исключить дублирование результатов, получаемых каждой проектной группой.

- обеспечивать возможность ведения работ по проектированию отдельных подсистем небольшими группами, что обусловлено принципами управляемости коллектива и повышения производительности за счет минимизации числа внешних связей;
- обеспечивать минимальное время получения работоспособной системы;
- предусматривать возможность управления конфигурацией проекта, ведения версий проекта и его составляющих, автоматического выпуска проектной документации и синхронизацию ее версий с версиями проекта;
- обеспечивать независимость выполняемых проектных решений от средств реализации системы — системы управления базами данных, операционной системы, языка и системы программирования.

Методология RAD

На начальном этапе существования компьютерных информационных систем их разработка велась на традиционных языках программирования. Однако по мере возрастания сложности разрабатываемых систем и запросов пользователей (чему в значительной степени способствовал прогресс в области вычислительной техники, а также появление удобного графического интерфейса пользователя в системном программном обеспечении) требовались новые средства, обеспечивающие значительное сокращение сроков разработки. Это послужило предпосылкой к созданию целого направления в области программного обеспечения — инструментальных средств для быстрой разработки приложений. Развитие этого направления привело к появлению на рынке программного обеспечения средств автоматизации практически всех этапов жизненного цикла информационных систем.

Основные особенности методологии RAD

Методология создания информационных систем, основанная на использовании средств быстрой разработки приложений, получила в последнее время широкое распространение и приобрела название *методологии быстрой разработки приложений* (Rapid Application Development, RAD). Данная методология охватывает все этапы жизненного цикла современных информационных систем.

Методология RAD — это комплекс специальных инструментальных средств, позволяющих оперировать с определенным набором графических объектов, функционально отображающих отдельные информационные компоненты приложений.

Под методологией быстрой разработки приложений обычно понимается процесс разработки информационных систем, основанный на трех основных элементах:

- небольшой команде программистов (обычно от 2 до 10 человек);
- тщательно проработанном производственном графике работ, рассчитанном на сравнительно короткий срок разработки (от 2 до 6 мес.);
- итерационной модели разработки, основанной на тесном взаимодействии с заказчиком — по мере выполнения проекта разработчики уточняют и реализуют в продукте требования, выдвигаемые заказчиком.

Основные принципы методологии RAD можно свести к следующим:

- используется итерационная (спиральная) модель разработки;
- полное завершение работ на каждом из этапов жизненного цикла не обязательно;
- в процессе разработки информационной системы обеспечивается тесное взаимодействие с заказчиком и будущими пользователями;
- применяются CASE-средства и средства быстрой разработки приложений;
- применяются средства управления конфигурацией, облегчающие внесение изменений в проект и сопровождение готовой системы;
- используются прототипы, позволяющие полнее выяснить и реализовать потребности конечного пользователя;
- тестирование и развитие проекта осуществляются одновременно с разработкой;
- разработка ведется немногочисленной и хорошо управляемой командой профессионалов;
- обеспечиваются грамотное руководство разработкой системы, четкое планирование и контроль выполнения работ.

Объектно-ориентированный подход

Средства RAD позволили реализовать совершенно иную по сравнению с традиционной технологию создания приложений: информационные объекты формируются как некие действующие модели (прототипы), чье функционирование согласуется с пользователем, а затем разработчик может переходить непосредственно к формированию законченных приложений, не теряя из виду общей картины проектируемой системы.

Возможность использования подобного подхода в значительной степени является результатом применения принципов объектно-ориентированного проектирования. Эти принципы позволяют преодолеть одну из главных трудностей, возникающих при разработке сложных систем, — колоссальный разрыв между реальным миром (предметной областью) и имитирующей средой.

Использование объектно-ориентированных принципов позволяет создать описание (модель) предметной области в виде совокупности объектов — сущностей, объединяющих данные и методы обработки этих данных (процедуры). Каждый объект обладает собственным поведением и моделирует некоторый объект реального мира. С этой точки зрения объект является вполне осязаемым и демонстрирует определенное поведение.

В объектном подходе акцент переносится на конкретные характеристики физической или абстрактной системы, являющейся предметом программного моделирования. Объекты обладают целостностью, которая не может быть нарушена. Таким образом, свойства, характеризующие объект и его поведение, остаются неизменными. Объект может только менять состояние, управляться или становиться в определенное отношение к другим объектам.

Широкое распространение объектно-ориентированное программирование (ОПП) получило с появлением средств визуального программирования, которые обеспечивают слияние (инкапсуляцию) данных с процедурами, описывающими поведение реальных объектов, в объекты программ, которые могут быть отображены определенным образом в графической пользовательской среде. Это позволило приступить к созданию программных систем, максимально похожих на реальные, и добиваться наивысшего уровня абстракции.

При разработке приложений с помощью инструментов RAD используются множество готовых объектов, сохраняемых в общедоступном хранилище. Однако имеется также возможность разработки новых объектов. При этом новые объекты могут разрабатываться как на основе существующих, так и «с нуля».

Инструментальные средства RAD обладают удобным графическим интерфейсом пользователя и позволяют на основе стандартных объектов формировать простые

приложения без написания кода программы. Это является большим преимуществом RAD, так как в значительной степени сокращает рутинную работу по разработке интерфейсов пользователя (при использовании обычных средств разработка интерфейсов представляет собой достаточно трудоемкую задачу, решение которой отнимает много времени). Высокая скорость разработки интерфейсной части приложений позволяет быстро создавать прототипы и упрощает взаимодействие с конечными пользователями.

Таким образом, инструменты RAD позволяют разработчикам сконцентрировать усилия на сущности реальных бизнес-процессов предприятия, для которого создается информационная система. В итоге это приводит к повышению качества разрабатываемой системы.

Визуальное программирование

Применение принципов объектно-ориентированного программирования позволило создать принципиально новые средства проектирования приложений, называемые средствами *визуального программирования.* Визуальные инструменты RAD позволяют создавать сложные графические интерфейсы пользователя вообще без написания кода программы. При этом разработчик может на любом этапе наблюдать то, что закладывается в основу принимаемых решений.

Визуальные средства разработки оперируют, в первую очередь, со стандартными интерфейсными объектами — окнами, списками, текстами, которые легко можно связать с данными из базы данных и отобразить на экране монитора. Другая группа объектов представляет собой стандартные элементы управления — кнопки, переключатели, флажки, меню и т. п., с помощью которых осуществляется управление отображаемыми данными. Все эти объекты могут быть стандартным образом описаны средствами языка, а сами описания сохранены для многократного использования в будущем.

В настоящее время существует довольно много различных визуальных средств разработки приложений, но все они могут быть разделены на две группы — универсальные и специализированные.

Среди **универсальных** систем визуального программирования сейчас наиболее распространены такие, как Borland Delphi и Visual Basic. Универсальными мы их называем потому, что они не ориентированы исключительно на разработку приложений баз данных — с их помощью могут быть разработаны приложения почти любого типа, в том числе и информационные приложения. Причем программы, разрабатываемые с помощью универсальных систем, могут взаимодействовать практически с любыми системами управления базами данных.

Специализированные средства разработки ориентированы только на создание приложений баз данных. Причем, как правило, они привязаны к вполне определенным

системам управления базами данных. В качестве примера таких систем можно привести системы Power Builder фирмы Sybase (естественно, предназначенную для работы с СУБД Sybase Anywhere Server) и Visual FoxPro фирмы Microsoft.

Поскольку задачи создания прототипов и разработки пользовательского интерфейса, по существу, слились, программист получил непрерывную обратную связь с конечными пользователями, которые могут не только наблюдать за созданием приложения, но и активно участвовать в нем, корректировать результаты и свои требования. Это также способствует сокращению сроков разработки и является важным психологическим аспектом, который привлекает к RAD все большее число разработчиков.

Событийное программирование

Логика приложения, построенного средствами RAD, является событийно-ориентированной. Это означает, что каждый объект, входящий в состав приложения, может генерировать события и реагировать на события, генерируемые другими объектами. Примерами событий могут быть открытие и закрытие окон, щелчок на кнопке, нажатие клавиши клавиатуры, движение мыши, изменение данных в базе данных и т. п.

Разработчик реализует логику приложения путем определения обработчика каждого события — процедуры, выполняемой объектом при наступлении соответствующего события. Например, обработчик события «щелчок на кнопке» может открыть диалоговое окно. Таким образом, управление объектами осуществляется с помощью событий.

Фазы жизненного цикла в рамках методологии RAD

При использовании методологии быстрой разработки приложений жизненный цикл информационной системы состоит из четырех фаз:

- анализа и планирования требований;
- проектирования;
- построения;
- внедрения.

Рассмотрим каждую из них более подробно.

Фаза анализа и планирования требований

На фазе анализа и планирования требований определяются:

- функции, которые должна выполнять разрабатываемая информационная система;
- наиболее приоритетные функции, требующие разработки в первую очередь; Q информационные потребности;
- масштаб проекта;

- временные рамки для каждой из последующих фаз;
- сама возможность реализации данного проекта в установленных рамках финансирования на имеющихся аппаратных и программных средствах.

Если реализация проекта принципиально возможна, то результатом фазы анализа и планирования требований будет список функций разрабатываемой информационной системы с указанием их приоритетов, а также предварительные функциональные и информационные модели системы.

Фаза проектирования

На фазе проектирования необходимым инструментом являются CASE-средства, используемые для быстрого получения работающих прототипов приложений.

Прототипы, созданные с помощью CASE-средств, анализируются пользователями, которые уточняют и дополняют те требования к системе, которые не были выявлены на предыдущей фазе. Таким образом, на данной фазе также необходимо участие будущих пользователей в техническом проектировании системы.

Далее на этой фазе проводится анализ и, если требуется, корректировка функциональной модели системы. Детально рассматривается каждый процесс системы. При необходимости для каждого элементарного процесса создается частичный прототип: экран, диалоговое окно или отчет (это позволяет устранить неясности или неоднозначности). Затем определяются требования разграничения доступа к данным.

После детального рассмотрения процессов определяется количество функциональных элементов разрабатываемой системы. Это позволяет разделить информационную систему на ряд подсистем, каждая из которых реализуется одной командой разработчиков за приемлемое для RAD-проектов время (порядка полутора месяцев). С использованием CASE-средств проект распределяется между различными командами — делится функциональная модель.

На этой же фазе происходит определение набора необходимой документации.

Результатами данной фазы являются:

- общая информационная модель системы;
- функциональные модели системы в целом и подсистем, реализуемых отдельными командами разработчиков;
- точно определенные с помощью CASE-средства интерфейсы между автономно разрабатываемыми подсистемами;
- построенные прототипы экранов, диалоговых окон и отчетов.

Фаза построения

На фазе построения выполняется собственно быстрая разработка приложения. На данной фазе разработчики производят итеративное построение реальной системы на

основе полученных ранее моделей, а также требований нефункционального характера. Разработка приложения ведется средствами визуального программирования. Формирование программного кода частично выполняется с помощью автоматических генераторов кода, входящих в состав CASE-средств. Код генерируется на основе разработанных моделей.

На фазе построения также требуется участие пользователей системы, которые 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.

Завершается физическое проектирование системы, а именно:

- определяется необходимость распределения данных;
- производится анализ использования данных;
- производится физическое проектирование базы данных;
- определяются требования к аппаратным ресурсам;
- определяются способы повышения производительности;
- завершается разработка документации проекта.

Результатом реализации данной фазы является готовая информационная система, удовлетворяющая всем требованиям пользователей.

Фаза внедрения

Фаза внедрения в основном сводится к обучению пользователей разработанной информационной системы.

Так как фаза построения достаточно непродолжительна, планирование и подготовка к внедрению должны начинаться заранее, еще на этапе проектирования системы.

Ограничения методологии RAD

Несмотря на все свои достоинства, методология RAD (как, впрочем, и любая другая методология) не может претендовать на универсальность. Ее применение наиболее эффективно при создании сравнительно небольших систем, разрабатываемых для конкретного заказчика.

При разработке типовых систем, не являющихся законченным продуктом, а представляющих собой совокупность типовых элементов информационной системы, большое значение имеют такие показатели проекта, как управляемость и качество,

которые могут войти в противоречие с простотой и скоростью разработки. Это связано с тем, что типовые системы обычно централизованно сопровождаются и могут адаптироваться к различным программно-аппаратным платформам, системам управления базами данных, коммуникационным средствам, а также интегрироваться с существующими разработками. Поэтому для такого рода проектов необходимы высокий уровень планирования и жесткая дисциплина проектирования, строгое следование заранее разработанным протоколам и интерфейсам, что снижает скорость разработки.

Методология RAD не подходит для создания не только типовых информационных систем, но и сложных расчетных программ, операционных систем и программ управления сложными инженерно-техническими объектами, то есть программ, требующих написания большого объема уникального кода.

Методология RAD не может быть использована для разработки приложений, в которых интерфейс пользователя является вторичным, то есть отсутствует наглядное определение логики работы системы. Примерами таких приложений могут служить приложения реального времени, драйверы или службы.

Совершенно неприемлема методология RAD для разработки систем, от которых зависит безопасность людей, например систем управления транспортом или атомными электростанциями. Это обусловлено тем, что итеративный подход, являющийся одной из основ RAD, предполагает, что первые версии системы не будут полностью работоспособными, что в данном случае может привести к серьезнейшим катастрофам.

Лекция №5

Содержание лекции

Методология и технология разработки информационных систем

Профили открытых информационных систем

Создание, сопровождение и развитие современных сложных информационных систем базируется на методологии построения таких систем как открытых. *Открытые информационные системы* создаются в процессе информатизации всех основных сфер современного общества: органов государственного управления, финансово-кредитной сферы, информационного обслуживания предпринимательской деятельности, производственной сферы, науки, образования. Развитие и использование открытых информационных систем неразрывно связаны с применением стандартов на основе методологии функциональной стандартизации информационных технологий.

Понятие профиля информационной системы

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

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

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

Профиль не должен противоречить входящим в него базовым стандартам и нормативным документам. Применяемые в соответствии с профилем необязательные возможности и значения параметров, выбранные из альтернативных вариантов, должны оставаться в допустимых пределах.

На базе одной совокупности базовых стандартов могут формироваться и утверждаться различные профили для разных проектов информационных систем. Ограничения базовых документов профиля и их согласованность, контролируемая разработчиками профиля, должны обеспечивать качество, совместимость и корректное взаимодействие отдельных компонентов системы, соответствующих профилю, в заданной области его применения.

Базовые стандарты и профили в зависимости от проблемно-ориентированной области применения информационных систем могут использоваться как непосредственные директивные, руководящие или рекомендательные документы, а также как нормативная база, необходимая при выборе или разработке средств автоматизации технологических этапов или процессов создания, сопровождения и развития информационных систем.

Обычно рассматривают две группы профилей, регламентирующих:

- архитектуру и структуру информационной системы;
- процессы проектирования, разработки, применения, сопровождения и развития системы.

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

- профили конкретной информационной системы, определяющие стандартизованные проектные решения в пределах данного проекта;
- профили информационной системы, предназначенные для решения некоторого класса прикладных задач.

Профили информационных систем унифицируют и регламентируют только часть требований, характеристик, показателей качества объектов и процессов, выделенных и формализованных на базе стандартов и нормативных документов. Другая часть

функциональных и технических характеристик системы определяется заказчиками и разработчиками творчески, без учета положений нормативных документов.

Принципы формирования профиля информационной системы

Профили информационных систем призваны решить следующие задачи:

- снижение трудоемкости проектов;
- повышение качества компонентов информационных систем;
- обеспечение расширяемости и масштабируемости разрабатываемых систем;
- обеспечение возможности функциональной интеграции в информационную систему задач, которые раньше решались раздельно;
- обеспечение переносимости прикладного программного обеспечения.
- зависимости от того, какие из указанных задач являются наиболее приоритетными, производится выбор стандартов и документов для формирования профиля.

Актуальность использования профилей информационных систем обусловлена современным состоянием стандартизации информационных технологий, которое характеризуется следующими особенностями:

- существует множество международных и национальных стандартов, которые не полностью и неравномерно удовлетворяют потребностям в стандартизации объектов и процессов создания и применения сложных информационных систем;
- длительные сроки разработки, согласования и утверждения международных и национальных стандартов приводят к их консерватизму и хроническому отставанию от современных информационных технологий;
- функциональными стандартами поддержаны и регламентированы только самые простые объекты и рутинные, массовые процессы (телекоммуникации, программирование, документирование программ и данных), а наиболее сложные и творческие процессы создания и развития крупных распределенных информационных систем (системный анализ и проектирование, интеграция компонентов и систем, испытания и сертификация) почти не поддержаны требованиями и рекомендациями стандартов из-за трудности их формализации и унификации;
- совершенствование и согласование нормативных и методических документов в ряде случаев позволяют создать на их основе национальные и международные стандарты.

Подходы к формированию профилей информационных систем могут быть различными. В международной функциональной стандартизации информационных технологий принято довольно жесткое понятие профиля. Считается, что его основой могут быть только утвержденные международные и национальные стандарты. Использование стандартов де-факто и нормативных фирменных документов не допускается. При таком

подходе затруднены унификация, регламентирование и параметризация множества конкретных функций и характеристик сложных объектов архитектуры и структуры современных информационных систем.

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

Эталонная модель среды открытых систем определяет разделение любой информационной системы на две составляющие: *приложения* (прикладные программы и программные комплексы) и *среду,* в которой эти приложения функционируют.

Между приложениями и средой определяются стандартизованные прикладные программные интерфейсы (Application Programming Interface, API), которые являются необходимой частью профилей любой открытой системы. Кроме того, в профилях могут быть определены унифицированные интерфейсы взаимодействия функциональных частей друг с другом и интерфейсы взаимодействия между компонентами среды системы. Спецификации выполняемых функций и интерфейсов взаимодействия могут быть оформлены в виде профилей компонентов системы. Таким образом, профили информационной системы с иерархической структурой могут включать в себя:

- стандартизованные описания функций, выполняемых данной системой;
- функции взаимодействия системы с внешней для нее средой;
- стандартизованные интерфейсы между приложениями и средой информационной системы;
- профили отдельных функциональных компонентов, входящих в систему. Для эффективного использования конкретного профиля необходимо:
- выделить объединенные логической связью проблемно-ориентированные области функционирования, где могут применяться стандарты, общие для одной организации или группы организаций;
- идентифицировать стандарты и нормативные документы, варианты их использования и параметры, которые необходимо включить в профиль;
- документально зафиксировать части конкретного профиля, в которых требуется создание новых стандартов или нормативных документов, и идентифицировать характеристики, которые могут оказаться важными для разработки недостающих стандартов и нормативных документов этого профиля;
- формализовать профиль в соответствии с его категорией, включая стандарты, различные варианты нормативных документов и дополнительные параметры, которые непосредственно связаны с профилем;

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

Использование профилей способствует унификации при разработке тестов, проверяющих качество и взаимодействие компонентов проектируемой информационной системы. Профили должны определяться таким образом, чтобы тестирование их реализации можно было проводить в максимальной степени по стандартизованной методике. При этом возможно применение ранее разработанных методик, так как международные стандарты и профили являются основой для создания общепризнанных аттестационных тестов.

Структура профилей информационных систем

Разработка и применение профилей являются органической частью процессов проектирования, разработки и сопровождения информационных систем. Профили характеризуют каждую конкретную информационную систему на всех стадиях ее жизненного цикла, задавая согласованный набор базовых стандартов, которым должны соответствовать система и ее компоненты.

Стандарты, важные с точки зрения заказчика, должны задаваться в техническом задании на проектирование системы и составлять ее первичный профиль. То, что не задано в техническом задании, первоначально остается на усмотрение разработчика системы, который, руководствуясь требованиями технического задания, может дополнять и развивать профили системы и впоследствии согласовывать их с заказчиком. Таким образом, профиль конкретной системы не является статичным, он развивается и конкретизируется в процессе проектирования информационной системы и оформляется в составе документации проекта системы.

В профиль конкретной системы включаются спецификации компонентов, разработанных в составе данного проекта, и спецификации использованных готовых программных и аппаратных средств, если эти средства не специфицированы соответствующими стандартами. После завершения проектирования и испытаний системы, в ходе которых проверяется ее соответствие профилю, профиль применяется как основной инструмент сопровождения системы при эксплуатации, модернизации и развитии.

Формирование и применение профилей конкретных информационных систем реализуется на основе международных и национальных стандартов, ведомственных нормативных документов, а также стандартов де-факто при условии доступности соответствующих им спецификаций. Для обеспечения корректного применения профилей их описания должны содержать:

- определение целей использования профиля;

- точное перечисление функций объекта или процесса стандартизации, определяемого профилем;
- формализованные сценарии применения базовых стандартов и спецификаций, включенных в профиль;
- сводку требований к информационной системе или к ее компонентам, определяющих их соответствие профилю, и требований к методам тестирования соответствия;
- нормативные ссылки на конкретный набор стандартов и других нормативных документов, составляющих профиль, с точным указанием применяемых редакций и ограничений, способных повлиять на достижение корректного взаимодействия объектов стандартизации при использовании профиля;
- информационные ссылки на все исходные документы.

На стадиях жизненного цикла информационной системы выбираются и затем применяются следующие основные функциональные профили:

- прикладного программного обеспечения;
- среды информационной системы;
- защиты информации в информационной системе;
- инструментальных средств, встроенных в информационную систему.

Профиль прикладного программного обеспечения

Прикладное программное обеспечение всегда является проблемно-ориентированным и определяет основные функции информационной системы. Функциональные профили системы должны включать в себя согласованные базовые стандарты. При использовании функциональных профилей информационных систем следует еще иметь в виду согласование этих профилей между собой. При согласовании функциональных профилей возможны также уточнения профиля среды системы и профиля встраиваемых инструментальных средств создания, сопровождения и развития прикладного программного обеспечения.

Профиль среды информационной системы

Профиль среды информационной системы должен определять ее архитектуру в соответствии с выбранной моделью обработки данных.

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

- графического пользовательского интерфейса;

- реляционных или объектно-ориентированных СУБД (например, стандарта языка SQL-92 и спецификации доступа к разным базам данных);
- операционных систем с учетом сетевых функций, выполняемых на уровне операционной системы;
- телекоммуникационной среды в части услуг и служб прикладного уровня (электронной почты, доступа к удаленным базам данных, передачи файлов, доступа к файлам и управления файлами).

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

Профиль защиты информации

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

- функции, реализуемые операционной системой;
- функции защиты от несанкционированного доступа, реализуемые на уровне программного обеспечения промежуточного слоя;
- функции управления данными, реализуемые СУБД;
- функции защиты программных средств, включая средства защиты от вирусов;
- функции защиты информации при обмене данными в распределенных системах, включая криптографические функции;
- функции администрирования средств безопасности.

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

Профиль инструментальных средств

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

Состав инструментальных средств определяется на основании решений и нормативных документов об организации сопровождения и развития информационной системы. Функциональная область профиля инструментальных средств, встроенных в систему, охватывает функции централизованного управления и администрирования, связанные с:

- контролем производительности и корректности функционирования системы в целом;
- конфигурированием прикладного программного обеспечения, тиражированием версий;
- управлением доступом пользователей к ресурсам системы и конфигурированием ресурсов;
- перенастройкой приложений в связи с изменениями прикладных функций информационной системы;
- настройкой пользовательских интерфейсов (экранных форм и отчетов);
- ведением баз данных системы;
- восстановлением работоспособности системы после сбоев и аварий.

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

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

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

Лекция №6

Содержание лекции

Методология и технология разработки информационных систем

Стандарты и методики

Одним из важных условий эффективного использования информационных технологий является внедрение корпоративных стандартов. Корпоративные стандарты представляют собой соглашение о единых правилах организации технологии или управления. При этом за основу корпоративных стандартов могут приниматься отраслевые, национальные и даже международные стандарты.

Однако динамика развития информационных технологий приводит к быстрому устареванию существующих стандартов и методик разработки информационных систем. Так, в связи со значительным прогрессом в области программного обеспечения и средств вычислительной техники наблюдается рост размеров и сложности информационных систем. При этом существенно меняются требования как к основным функциям и сервисным возможностям систем, так и к динамике изменения этих функций. В этих условиях применение классических способов разработки и обеспечения качества информационных систем становится малоэффективным и не приводит к уровню качества, адекватному реальным требованиям.

Полезны в этом отношении стандарты открытых систем (в первую очередь, стандарты на интерфейсы различных видов, включая лингвистические, и на протоколы взаимодействия). Однако разработка систем в новых условиях требует также новых методов проектирования и новой организации проектных работ. Проектирование и методическая поддержка разработки информационных систем, включая программное

обеспечение и базы данных, традиционно поддерживаются многими стандартами и фирменными методиками. Вместе с тем известно, что требуется адаптивное планирование разработки, в том числе в динамике процесса ее выполнения. Одним из способов адаптивного проектирования является разработка и применение профилей жизненного цикла информационных систем и программного обеспечения. Корпоративные стандарты образуют целостную систему, которая включает три вида стандартов:

- на продукты и услуги;
- на процессы и технологии;
- на формы коллективной деятельности, или управленческие стандарты.

Виды стандартов

Существующие на сегодняшний день стандарты можно условно разделить на несколько групп.

- *По предмету стандартизации.* К этой группе можно отнести функциональные стандарты (стандарты на языки программирования, интерфейсы, протоколы) и стандарты на организацию жизненного цикла создания и использования информационных систем (ИС) и программного обеспечения (ПО).
- *По утверждающей организации.* Здесь можно выделить официальные международные, официальные национальные или ведомственные национальные стандарты (например, ГОСТ, ANSI, IDEFO/1), стандарты международных консорциумов и комитетов по стандартизации (например, OMG), стандарты де-факто — официально никем не утвержденные, но фактически действующие (например, стандартом де-факто долгое время были язык взаимодействия с реляционными базами данных SQL и язык программирования С), фирменные стандарты (например, Microsoft ODBC).
- *По методическому источнику.* К этой группе относятся различного рода методические материалы ведущих фирм-разработчиков программного обеспечения, фирм-консультантов, научных центров, консорциумов по стандартизации.

Вкратце рассмотрим методику CDM (Custom Development Method) фирмы Oracle по разработке прикладных ИС под заказ и Международный стандарт ISO/IEC 12207:1995-08-01 01 на организацию жизненного цикла продуктов программного обеспечения.

Методика CDM фирмы Oracle

Одним из уже сложившихся направлений деятельности фирмы Oracle стали разработка методологических основ и производство инструментальных средств для автоматизации процессов разработки сложных прикладных систем, ориентированных на интенсивное использование баз данных. Методика CDM является развитием давно

разработанной методики CASE-Method фирмы Oracle, применяемой в CASE-средстве Oracle CASE (в новых версиях — Designer/2000).

Перечислим основные составляющие CASE-технологии и инструментальной среды фирмы Oracle.

- Методология структурного нисходящего проектирования, при которой разработка прикладной системы представляется в виде последовательности четко определенных этапов.
- Поддержка всех этапов жизненного цикла прикладной системы, начиная с самых общих описаний предметной области до получения и сопровождения готового программного продукта.
- Ориентация на реализацию приложений в архитектуре клиент-сервер с использованием всех особенностей современных серверов баз данных, включая декларативные ограничения целостности, хранимые процедуры, триггеры баз данных, с поддержкой в клиентской части всех современных стандартов и требований к графическому интерфейсу конечного пользователя.
- Наличие централизованной базы данных — репозитария. Репозитарий предназначен для хранения спецификаций проекта прикладной системы на всех этапах ее разработки. Он представляет собой базу данных специальной структуры, работающую под управлением СУБД Oracle.
- Возможность одновременной работы с репозитарием многих пользователей. Такой многопользовательский режим почти автоматически обеспечивается стандартными средствами СУБД Oracle. Централизованное хранение проекта системы и управление одновременным доступом к нему всех участников разработки поддерживают согласованность действий разработчиков и не допускают ситуаций, в которых каждый проектировщик или программист работает со своей версией проекта и модифицирует ее независимо от других.
- Автоматизация последовательного перехода от одного этапа разработки к следующему. Для этого предусмотрены специальные утилиты, с помощью которых можно по спецификациям концептуального уровня (модели предметной области) автоматически получать первоначальный вариант спецификации уровня проектирования (описание структуры базы данных и состава программных модулей), чтобы на его основе после всех необходимых уточнений и дополнений автоматически генерировать готовые к выполнению программы.
- Автоматизация различных стандартных действий по проектированию и разработке приложения. Предусматривается генерация многочисленных отчетов по содержимому репозитария, обеспечивающих полное документирование текущей версии

системы на всех этапах ее разработки; с помощью специальных процедур предоставляется возможность проверки спецификаций на полноту и непротиворечивость.

Общая структура

Жизненный цикл формируется из определенных этапов (фаз) проекта и процессов, каждый из которых выполняется в течение нескольких этапов.

Методика CDM определяет следующие фазы ЖЦ ИС:

стратегию;

- анализ (формулирование детальных требований к прикладной системе);
- проектирование (преобразование требований в детальные спецификации системы);
- реализацию (написание и тестирование приложений);
- внедрение (установка новой прикладной системы, подготовка к началу эксплуатации);
- эксплуатацию (поддержка и сопровождение приложения, планирование будущих функциональных расширений).

Первый этап связан с моделированием и анализом процессов, описывающих деятельность организации, технологические особенности работы. Целью является построение моделей существующих процессов, выявление их недостатков и возможных источников совершенствования. Этот этап не является обязательным в случае, когда существующие технология и организационные структуры четко определены, хорошо понятны и не требуют дополнительного изучения и реорганизации.

На втором этапе разрабатываются детальные концептуальные модели предметной области, описывающие информационные потребности организации, особенности функционирования и т.п. Результатом являются модели двух типов:

- информационные, отражающие структуру и общие закономерности предметной области;
- функциональные, описывающие особенности решаемых задач.

На третьей стадии (этапе проектирования) на основании концептуальных моделей вырабатываются технические спецификации будущей прикладной системы – определяются структура и состав базы данных, специфицируется набор программных модулей. Первоначальный вариант проектных спецификаций может быть получен автоматически с помощью специальных утилит на основании данных концептуальных моделей.

На этапе реализации создаются программы, отвечающие всем требованиям проектных спецификаций.

Методика CDM выделяет следующие процессы, протекающие на протяжении ЖЦ ИС:

- определение производственных требований;
- исследование существующих систем;
- определение технической архитектуры;
- проектирование и построение базы данных;
- проектирование и реализацию модулей;
- конвертирование данных;
- документирование;
- тестирование;
- обучение;
- переход к новой системе;
- поддержку и сопровождение.

Особенности методики CDM

Отметим основные особенности методики CDM, определяющие область ее применения и присущие ей ограничения.

- Степень адаптивности CDM ограничивается тремя моделями жизненного цикла:
  - *классическая* модель предусматривает все этапы;
  - *быстрая разработка* ориентирована на использование инструментов моделирования и программирования Oracle;
  - *облегченный подход* рекомендуется в случае малых проектов и возможности быстро прототипировать приложения.
- Методика не предусматривает включение дополнительных задач, которые не оговорены в CDM, и их привязку к остальным. Также исключено удаление задачи (и порождаемых ею документов), не предусмотренное ни одной из трех моделей жизненного цикла, и изменение предложенной последовательности выполнения задач.
- Все модели жизненного цикла являются по сути каскадными. Даже «облегченный подход», несмотря на итерационность действий по прототипированию, сохраняет общий последовательный и детерминированный порядок выполнения задач.
- Методика не является обязательной, но может считаться фирменным стандартом. При формальном применении степень обязательности полностью соответствует ограничениям возможностей адаптации.
- Прикладная система рассматривается в основном как программно-техническая система, например, возможность выполнения организационно-структурных

преобразований, практически всегда происходящих при переходе к новой информационной системе, в этой методике отсутствует.

- CDM теснейшим образом опирается на инструментарий Oracle, несмотря на утверждения о простоте адаптации CDM к проектам, в которых используется другой комплект инструментальных средств.
- Методика CDM представляет собой вполне конкретный материал, детализированный до уровня заготовок проектных документов, рассчитанных на прямое использование в проектах информационных систем с опорой на инструментальные средства и СУБД фирмы Oracle.

Международный стандарт ISO/IEC 12207: 1995-08-01

Первая редакция ISO Г2207 была подготовлена в 1995 г. подкомитетом SC7 (Проектирование программного обеспечения) объединенного технического комитета JTC1 (Информационные технологии) ISO/IEC.

По определению, ISO 12207 — базовый стандарт процессов жизненного цикла ПО, ориентированный на различные виды ПО и типы проектов автоматизированных систем, в которых ПО является одной из составных частей. Стандарт определяет стратегию и общий порядок создания и эксплуатации ПО, он охватывает жизненный цикл от концептуализации идей до завершения проекта.

Целесообразность совместного использования стандартов на ИС и на ПО обусловливается одним из положений ISO 12207, согласно которому процессы, протекающие во время жизненного цикла ПО, должны быть совместимы с процессами, протекающими во время жизненного цикла автоматизированной системы.

Согласно ISO 12207, система — это объединение одного или нескольких процессов, аппаратных средств, программного обеспечения, оборудования и людей для удовлетворения определенным потребностям или целям.

Общая структура

В стандарте ISO 12207 не предусмотрено каких-либо этапов (фаз или стадий) ЖЦ ИС. Данный стандарт определяет лишь ряд процессов, причем по сравнению с CDM стандарт ISO 12207 состоит из гораздо более крупных обобщенных процессов (приобретение, поставка, разработка и т.п.). Несколько утрируя, можно сказать, что один процесс ISO 12207 сопоставим со всеми процессами CDM вместе взятыми.

Согласно ISO 12207, каждый процесс подразделяется на ряд действий, а каждое действие — на ряд задач.

Очень важной особенностью ISO 12207 по сравнению с CDM является то, что каждый процесс, действие или задача инициируется и выполняется другим процессом по

мере необходимости, причем нет заранее определенных последовательностей (естественно, при сохранении логики связей по исходным сведениям задач и т.п.).

Основные и вспомогательные процессы ЖЦ

В стандарте ISO 12207 описаны пять основных процессов ЖЦ программного обеспечения.

- *Процесс приобретения* определяет действия предприятия-покупателя, которое приобретает информационную систему, программный продукт или службу.
- *Процесс поставки* определяет действия предприятия-поставщика, которое снабжает покупателя системой, программным продуктом или службой.
- *Процесс разработки* определяет действия предприятия-разработчика, которое разрабатывает принцип построения программного изделия и программный продукт.
- *Процесс функционирования* определяет действия предприятия-оператора, которое обеспечивает обслуживание системы в целом (а не только программного обеспечения) в процессе ее функционирования в интересах пользователей. В отличие от действий, перечисленных разработчиком в инструкциях по эксплуатации (эта деятельность разработчика предусмотрена во всех трех рассматриваемых стандартах), определяются действия оператора по консультированию пользователей, получению обратной связи и др., которые он планирует сам и берет на себя соответствующие обязанности.
- *Процесс сопровождения* определяет действия персонала, обеспечивающего сопровождение программного продукта, то есть управление модификациями программного продукта, поддержку его текущего состояния и функциональной пригодности; сюда же относятся установка программного изделия на вычислительной системе и его удаление.

Помимо основных, стандарт ISO 12207 оговаривает 8 вспомогательных процессов, которые являются неотъемлемой частью всего ЖЦ программного изделия и обеспечивают должное качество проекта ПО. К вспомогательным процессам относятся:

- решения проблем;
- документирование;
- управление конфигурацией;
- обеспечение качества;
- верификация;
- аттестация;
- совместная оценка;
- аудит.

В стандарте ISO 12207 также определяются четыре организационных процесса:

управление;

создание инфраструктуры;

усовершенствование;

обучение.

ПРИМЕЧАНИЕ

Под процессом усовершенствования в стандарте ISO 12207 понимается не усовершенствование информационной системы или программного обеспечения, а улучшение самих процессов приобретения, разработки, обеспечения качества и т.д., реально осуществляемых в организации.

И, наконец, в стандарте ISO 12207 определен один особый процесс, называемый процессом адаптации, который определяет основные действия, необходимые для адаптации этого стандарта к условиям конкретного проекта.

Особенности стандарта ISO 12207

Все сказанное выше позволяет сформулировать некоторые особенности стандарта ISO 12207.

- Стандарт ISO 12207 имеет динамический характер, обусловленный способом определения последовательности выполнения процессов и решения задач, при котором один процесс при необходимости вызывает другой или его часть. Такой характер позволяет реализовать любую модель жизненного цикла.
- Стандарт ISO 12207 обеспечивает максимальную степень адаптивности. Множество процессов и задач сконструировано так, что возможна их адаптация в соответствии с конкретными проектами ИС. Адаптация сводится к исключению процессов, видов деятельности и задач, неприменимых в конкретном проекте.
- Стандарт принципиально не содержит описания конкретных методов действий, а тем более заготовок решений или документации. Он лишь описывает архитектуру процессов ЖЦ ПО, но не конкретизирует в деталях, как предоставлять услуги или решать задачи, включенные в процессы. Данный стандарт не предписывает имена, форматы или точное содержание получаемой документации. Решения такого типа принимаются сторонами, использующими стандарт.
- Качество обеспечивается с помощью различных процессов, выполняемых с разной степенью независимости контролирующей деятельности, вплоть до обязательных требований к полной независимости проверяющего персонала от какой-либо прямой ответственности за проверяемые объекты.

- Степень обязательности рассматриваемого стандарта следующая: после решения организации о соответствии торговых отношений стандарту ISO 12207 в качестве условия оговаривается ее ответственность за минимальный набор процессов и задач, которые обеспечивают согласованность с этим стандартом.
- Стандарт содержит предельно мало описаний, направленных на проектирование базы данных. Это можно считать оправданным, так как разные системы и разные прикладные комплексы ПО могут не только использовать весьма специфические типы баз данных, но и вообще обходиться без них.

Ценность стандарта ISO 12207 в том, что в нем представлены наборы задач, характеристики качества, критерии оценки и т.п., обеспечивающие всесторонний охват проектных ситуаций. Например, при анализе требований к системе предусматривается, что:

- рассматривается область применения системы для определения требований, предъявляемых к системе;
- спецификация требований системы должна описывать функции и возможности системы, области применения системы, организационные требования и требования пользователя, безопасность, защищенность, человеческие факторы, эргономику, связи, операции и требования сопровождения; проектные ограничения и квалификационные требования.

При анализе требований к ПО предусмотрено 11 характеристик качества, позволяющих обеспечить заданный уровень качества. При этом разработчик должен установить и документировать в виде требований к ПО следующие спецификации и характеристики:

- функциональные и возможные спецификации, включая исполнение, физические характеристики и условия среды эксплуатации, при которых единица ПО должна быть выполнена;
- внешние связи (интерфейсы) с единицей ПО;
- квалификационные требования;
- спецификации надежности, включая спецификации, связанные с методами функционирования и сопровождения, воздействия окружающей среды и вероятностью травмы персонала;
- спецификации защищенности, включая спецификации, связанные с компрометацией точности информации;
- человеческие факторы спецификаций по инженерной психологии (эргономике), включая связанные с ручным управлением, взаимодействием человека и оборудования, ограничениями на персонал и областями, нуждающимися в концентрированном

человеческом внимании, которые являются чувствительными к ошибкам человека и обучению;

- определение данных и требований к базе данных;
- установочные и приемочные требования поставляемого программного продукта в местах функционирования и сопровождения (эксплуатации);
- документацию пользователя;
- требования к интерфейсу пользователя.

ПРИМЕЧАНИЕ

Согласно стандарту ISO 12207, квалификационные требования — это набор критериев, или условий, которые должны быть удовлетворены для того, чтобы квалифицировать программный продукт как подчиняющийся (удовлетворяющий условиям) его спецификациям и готовый для использования в целевой окружающей среде.

Хотя стандарт не предписывает конкретной модели ЖЦ или метода разработки, он определяет, что стороны-участники при использовании стандарта ответственны:

- за выбор модели ЖЦ для разрабатываемого проекта;
- за адаптацию процессов и задач стандарта к этой модели;
- за выбор и применение методов разработки ПО;
- за выполнение действий и решение задач, подходящих для проекта ПО.

Лекция №7

Содержание лекции

CASE-технологии проектирования информационных систем

За последнее десятилетие сформировалось новое направление в программотехнике — CASE (Computer-Aided Software/System Engineering) — в дословном переводе — разработка программного обеспечения информационных систем при поддержке (с помощью) компьютера. В настоящее время не существует общепринятого определения CASE, термин CASE используется в весьма широком смысле. Первоначальное значение термина CASE, ограниченное вопросами автоматизации разработки только лишь программного обеспечения, в настоящее время приобрело новый смысл, охватывающий процесс разработки сложных автоматизированных информационных систем в целом. Теперь под термином CASE-средства понимаются программные средства, поддерживающие процессы создания и сопровождения ИС, включая анализ и формулировку требований, проектирование прикладного программного обеспечения (ПО) (приложений) и баз данных, генерацию кода, тестирование, документирование, обеспечение качества, конфигурационное управление и управление проектом, а также другие процессы. CASE-средства вместе с системным ПО и техническими средствами образуют полную среду разработки ИС.

CASE-средства позволяют не только создавать "правильные" продукты, но и обеспечить "правильный" процесс их создания. Основная цель CASE состоит в том, чтобы отделить проектирование ИС от его кодирования и последующих этапов разработки, а также скрыть от разработчиков все детали среды разработки и функционирования ИС. При использовании CASE-технологий изменяются все этапы жизненного цикла программного обеспечения (подробнее об этом будет сказано ниже) информационной системы, при этом наибольшие изменения касаются этапов анализа и проектирования. Большинство существующих CASE-средств основано на методологиях структурного (в основном) 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. Такие методологии обеспечивают строгое и наглядное описание проектируемой системы, которое начинается с ее общего обзора и затем детализируется, приобретая иерархическую структуру со все большим числом уровней. CASE-технологии успешно применяются для построения практически всех типов ИС, однако устойчивое положение они занимают в следующих областях:

- обеспечение разработки деловых и коммерческих ИС, широкое применение CASE-технологий обусловлены массовостью этой прикладной области, в которой CASE применяется не только для разработки ИС, но и для создания моделей систем, помогающих решать задачи стратегического планирования, управления финансами, определения политики фирм, обучения персонала и др. (это направление получило свое собственное название — бизнес-анализ);
- разработка системного и управляющих ИС. Активное применение CASE-технологий связано с большой сложностью данной проблематики и со стремлением повысить эффективность работ.

CASE — не революция в программотехнике, а результат естественного эволюционного развития всей отрасли средств, называемых ранее инструментальными или технологическими. С самого начала CASE-технологии развивались с целью преодоления ограничений при использовании структурных методологий проектирования 60—70-х гг. XX в. (сложности понимания, большой трудоемкости и стоимости использования, трудности внесения изменений в проектные спецификации и т. д.) за счет их автоматизации и интеграции поддерживающих средств. Таким образом, CASE-технологии не могут считаться самостоятельными методологиями, они только развивают структурные методологии и делают более эффективным их применение за счет автоматизации.

Помимо автоматизации структурных методологий и, как следствие, возможности применения современных методов системной и программной инженерии, CASE-средства обладают *следующими основными достоинствами:*

- улучшают качество создаваемых ИС за счет средств автоматического контроля (прежде всего контроля проекта);
- позволяют за короткое время создавать прототип будущей системы, что позволяет на ранних этапах оценить ожидаемый результат;
- ускоряют процесс проектирования и разработки;
- освобождают разработчика от рутинной работы, позволяя ему целиком сосредоточиться на творческой части разработки;
- поддерживают развитие и сопровождение разработки;

- поддерживают технологии повторного использования компонента разработки.

Появлению CASE-технологии и CASE-средств предшествовали исследования в области методологии программирования. Программирование обрело черты системного подхода с разработкой и внедрением языков высокого уровня, методов структурного и модульного программирования, языков проектирования и средств их поддержки, формальных и неформальных языков описаний системных требований и спецификаций и т. д. В 70—80-х гг. стала на практике применяться структурная методология, предоставляющая в распоряжение разработчиков строгие формализованные методы описания ИС и принимаемых технических решений. Она основана на наглядной графической технике: для описания различного рода моделей ИС используются схемы и диаграммы. Наглядность и строгость средств структурного анализа позволяла разработчикам и будущим пользователям системы с самого начала неформально участвовать в ее создании, обсуждать и закреплять понимание основных технических решений. Однако широкое применение этой методологии и следование ее рекомендациям при разработке контактных ИС встречалось достаточно редко, поскольку при неавтоматизированной (ручной) разработке это практически невозможно. Это и способствовало появлению программно-технических средств особого класса — CASE-средств, реализующих CASE-технологию создания и сопровождения ИС.

Необходимо понимать, что успешное применение CASE-средств невозможно без понимания базовой технологии, на которой эти средства основаны. Сами по себе программные CASE-средства являются средствами автоматизации процессов проектирования и сопровождения информационных систем. Без понимания методологии проектирования ИС невозможно применение CASE-средств.

Характеристика современных CASE-средств

Современные CASE-средства охватывают обширную область поддержки многочисленных технологий проектирования ИС: от простых средств анализа и документирования до полномасштабных средств автоматизации, покрывающих весь жизненный цикл (ЖЦ) ИС.

Наиболее трудоемкими этапами разработки ИС являются этапы анализа и проектирования, в процессе которых CASE-средства обеспечивают качество принимаемых технических решений и подготовку проектной документации. При этом большую роль играют методы визуального представления информации. Это предполагает построение структурных или иных диаграмм в реальном масштабе времени, использование многообразной цветовой палитры, сквозную проверку синтаксических правил. Графические средства моделирования предметной области позволяют

разработчикам в наглядном виде изучать существующую ИС, перестраивать ее в соответствии с поставленными целями и имеющимися ограничениями.

В разряд CASE-средств попадают как относительно дешевые системы для персональных компьютеров с весьма ограниченными возможностями, так и дорогостоящие системы для неоднородных вычислительных платформ и операционных сред. Так, современный рынок программных средств насчитывает около 300 различных CASE-средств, наиболее мощные из которых, так или иначе, используются практически всеми ведущими западными фирмами.

Обычно к CASE-средствам относят любое программное средство, автоматизирующее ту или иную совокупность процессов жизненного цикла ИС и обладающее следующими основными характерными особенностями:

- мощными графическими средствами для описания и документирования ИС, обеспечивающими удобный интерфейс с разработчиком и развивающими его творческие возможности;
- интеграцией отдельных компонент CASE-средств, обеспечивающей управляемость процессом разработки ИС;
- использованием специальным образом организованного хранилища проектных метаданных (репозитория). Интегрированное CASE-средство (или комплекс средств, поддерживающих полный ЖЦ ИС) содержит следующие компоненты:
- репозиторий, являющийся основой CASE-средства. Он должен обеспечивать хранение версий проекта и его отдельных компонентов, синхронизацию поступления информации от различных разработчиков при групповой разработке, контроль метаданных на полноту и непротиворечивость;
- графические средства анализа и проектирования, обеспечивающие создание и редактирование иерархически связанных диаграмм (DFD, ERD и др.), образующих модели ИС;
- средства разработки приложений, включая языки 4GL и генераторы кодов;
- средства конфигурационного управления;
- средства документирования;
- средства тестирования;
- средства управления проектом;
- средства реинжиниринга.

Все современные CASE-средства могут быть классифицированы в основном по типам и категориям. Классификация по типам отражает функциональную ориентацию CASE-средств на те или иные процессы ЖЦ. Классификация по категориям определяет степень интегрированности по выполняемым функциям и включает отдельные локальные средства, решающие небольшие автономные задачи (tools), набор частично

интегрированных средств, охватывающих большинство этапов жизненного цикла ИС (toolkit) и полностью интегрированные средства, поддерживающие весь ЖЦ ИС и связанные общим репозиторием. Помимо этого, CASE-средства можно классифицировать по следующим признакам:

- применяемым методологиям и моделям систем и БД;
- степени интегрированности с СУБД;
- доступным платформам.

Классификация по типам в основном совпадает с компонентным составом CASE-средств и включает следующие основные типы (после названия средства в скобках указана фирма-разработчик):

- **средства анализа** (Upper CASE), предназначенные для построения и анализа моделей предметной области (Design/IDEF (Meta Software), BPWin (Logic Works));
- **средства анализа и проектирований** (Middle CASE), поддерживающие наиболее распространенные методологии проектирования и использующиеся для создания проектных спецификаций (Vantage Team Builder (Cayenne), Designer/2000 (Oracle), Silverrun (CSA), PRO-IV (McDonnell Douglas), CASE. Аналитик (Макро-Проджект)). Выходом таких средств являются спецификации компонентов и интерфейсов системы, архитектуры системы, алгоритмов и структур данных;
- **средства проектирования баз данных**, обеспечивающие моделирование данных и генерацию схем баз данных (как правило, на языке SQL) для наиболее распространенных СУБД. К ним относятся ERwin (Logic Works). S-Designor (SDP) и DataBase Designer (Oracle). Средства проектирования баз данных имеются также в составе CASE-средств Vantage Team Builder, Designer/2000, Silverrun и PRO-IV;
- **средства разработки приложений**. К ним относятся средства 4GL (Uniface (Compuware), JAM (JYACC), PowerBuilder (Sybase), Developer/2000 (Oracle), New Era (Informix), SQL Windows (Gupta), Delphi (Borland) и др.) и генераторы кодов, входящие в состав Vantage Team Builder, PRO-IV и частично — в Silverrun;
- **средства реинжиниринга**, обеспечивающие анализ программных кодов и схем баз данных и формирование на их основе различных моделей и проектных спецификаций. Средства анализа схем БД и формирования ERD входят в состав Vantage Team Builder, PRO-IV, Silverrun, Designer/2000, ERwin и S-Designor. В области анализа программных кодов наибольшее распространение получают объектно-ориентированные CASE-средства, обеспечивающие реинжиниринг программ на языке C++ (Rational Rose (Rational Software), Object Team (Cayenne)). Вспомогательные типы включают:
- средства планирования и управления проектом (SE Companion, Microsoft Project и др.);
- средства конфигурационного управления (PVCS (Intersolv));

- средства тестирования (Quality Works (Segue Software));
- средства документирования (SoDA (Rational Software)).

На сегодняшний день российский рынок программного обеспечения располагает следующими наиболее развитыми CASE-средствами:

- Silverrun;
- Designer/2000;
- Vantage Team Builder (Westmount I-CASE);
- ERwin+BPwin;
- S-Designor;
- CASE-Аналитик.

Кроме того, на рынке постоянно появляются как новые для отечественных пользователей системы (например, CASE/ 4/0, PRO-IV, System Architect, Visible Analyst Workbench, EasyCASE), так и новые версии и модификации перечисленных систем.

Охарактеризуем основные возможности CASE-средств на примере имеющей широкое распространение системы Silverrun.

CASE-средство Silverrun американской фирмы Computer Systems Advisers, Inc. (CSA) используется для анализа и проектирования ИС бизнес-класса и ориентировано в большей степени на спиральную модель ЖЦ. Оно применимо для поддержки любой методологии, основанной на раздельном построении функциональной и информационной моделей (диаграмм потоков данных и диаграмм "сущность—связь").

Настройка на конкретную методологию обеспечивается выбором требуемой графической нотации моделей и набора правил проверки проектных спецификаций. В системе имеются готовые настройки для наиболее распространенных методологий: DATARUN (основная методология, поддерживаемая Silverrun), Gane/Sarson, Yourdon/DeMarco, Merise, Ward/Mellor, Information Engineering. Для каждого понятия, введенного в проекте, имеется возможность добавления собственных описателей. Архитектура Silverrun позволяет наращивать среду разработки по мере необходимости.

Silverrun имеет модульную структуру и состоит из четырех модулей, каждый из которых является самостоятельным продуктом и может приобретаться и использоваться без связи с остальными *модулями.*

Модуль *построения моделей бизнес-процессов* в форме диаграмм потоков данных (BPM — Business Process Modeler) позволяет моделировать функционирование обследуемой организации или создаваемой ИС. В модуле BPM обеспечена возможность работы с моделями большой сложности: автоматическая перенумерация, работа с деревом процессов (включая визуальное перетаскивание ветвей), отсоединение и присоединение частей модели для коллективной разработки. Диаграммы могут изображаться в нескольких предопределенных нотациях, включая Yourdon/DeMarco и Gane/Sarson.

Имеется также возможность создавать собственные нотации, в том числе добавлять в число изображаемых на схеме дескрипторов определенные пользователем поля.

Модуль *концептуального моделирования данных* (ERX — Entity-Relationship eXpert) обеспечивает построение моделей данных "сущность—связь", не привязанных к конкретной реализации. Этот модуль имеет встроенную экспертную систему, позволяющую создать корректную нормализованную модель данных посредством ответов на содержательные вопросы о взаимосвязи данных. Возможно автоматическое построение модели данных из описаний структур данных. Анализ функциональных зависимостей атрибутов дает возможность проверить соответствие модели требованиям третьей нормальной формы и обеспечить их выполнение. Проверенная модель передается в модуль RDM.

Модуль *реляционного моделирования* (RDM— Relational Data Modeler) позволяет создавать детализированные модели "сущность—связь", предназначенные для реализации в реляционной базе данных. В этом модуле документируются все конструкции, связанные с построением базы данных: индексы, триггеры, хранимые процедуры и т. д. Гибкая изменяемая нотация и расширяемость репозитория позволяют работать по любой методологии. Возможность создавать подсхемы соответствует подходу ANSI SPARC к представлению схемы базы данных. На языке подсхем моделируются как узлы распределенной обработки, так и пользовательские представления. Этот модуль обеспечивает проектирование и полное документирование реляционных баз данных.

Менеджер репозитория рабочей группы (WRM — Workgroup Repository Manager) применяется как словарь данных для хранения общей для всех моделей информации, а также обеспечивает интеграцию модулей Silverrun в единую среду проектирования.

Платой за высокую гибкость и разнообразие изобразительных средств построения моделей является такой недостаток Silverrun, как отсутствие жесткого взаимного контроля между компонентами различных моделей (например, возможности автоматического распространения изменений между DFD различных уровней декомпозиции). Следует, однако, отметить, что этот недостаток может иметь существенное значение только в случае использования каскадной модели ЖЦ ИС.

Для автоматической генерации схем баз данных у Silverrun существуют мосты к наиболее распространенным СУБД: Oracle, Informix, DB2, Ingres, Progress, SQL Server, SQLBase, Sybase. Для передачи данных в средства разработки приложений имеются мосты к языкам 4GL: JAM, PowerBuilder, SQL Windows, Uniface, NewEra, Delphi. Все мосты позволяют загрузить в Silverrun RDM информацию из каталогов соответствующих СУБД или языков 4GL. Это позволяет документировать, перепроектировать или переносить на новые платформы уже находящиеся в эксплуатации базы данных и прикладные системы. При использовании моста Silverrun расширяет свой внутренний

репозиторий специфичными для целевой системы атрибутами. После определения значений этих атрибутов генератор приложений переносит их во внутренний каталог среды разработки или использует при генерации кода на языке SQL. Таким образом, можно полностью определить ядро базы данных с использованием всех возможностей конкретной СУБД: триггеров, хранимых процедур, ограничений ссылочной целостности. При создании приложения на языке 4GL данные, перенесенные из репозитория Silverrun, используются либо для автоматической генерации интерфейсных объектов, либо для быстрого их создания вручную.

Для обмена данными с другими средствами автоматизации проектирования, создания специализированных процедур анализа и проверки проектных спецификаций, составления специализированных отчетов в соответствии с различными стандартами в системе Silverrun имеются три способа выдачи проектной информации во внешние файлы:

- система отчетов. Можно, определив содержимое отчета по репозиторию, выдать отчет в текстовый файл. Этот файл можно затем загрузить в текстовый редактор или включить в другой отчет;
- система экспорта/импорта. Для более полного контроля над структурой файлов в системе экспорта/импорта имеется возможность определять не только содержимое экспортного файла, но и разделители записей, полей в записях, маркеры начала и конца текстовых полей. Файлы с указанной структурой можно не только формировать, но и загружать в репозитории. Это дает возможность обмениваться данными с различными системами: другими CASE-средствами, СУБД, текстовыми редакторами и электронными таблицами;
- хранение репозитория во внешних файлах через ODBC-драйверы. Для доступа к данным репозитория из наиболее распространенных систем управления базами данных обеспечена возможность хранить всю проектную информацию непосредственно в формате этих СУБД.

Групповая работа поддерживается в системе Silverrun двумя способами:

- в стандартной однопользовательской версии имеется механизм контролируемого разделения и слияния моделей. Разделив модель на части, можно раздать их нескольким разработчикам. После детальной доработки модели объединяются в единые спецификации;
- сетевая версия Silverrun позволяет осуществлять одновременную групповую работу с моделями, хранящимися в сетевом репозитории на базе СУБД Oracle, Sybase или Informix. При этом несколько разработчиков могут работать с одной и той же моделью, так как блокировка объектов происходит на уровне отдельных элементов модели.

Имеются реализации Silverrun трех платформ — MS Windows, Macintosh и OS/2 Presentation Manager – с возможностью обмена проектными данными между ними.

Помимо системы Silverrun, укажем назначение и других популярных CASE-средств и их групп.

Vantage Team Builder представляет собой интегрированный программный продукт, ориентированный на реализацию каскадной модели ЖЦ ИС и поддержку полного ЖЦ ИС.

Uniface 6.1 – продукт фирмы Compuware (США) — представляет собой среду разработки крупномасштабных приложений в архитектуре "клиент—сервер".

CASE-средство Designer/2000 2.0 фирмы Oracle является интегрированным CASE-средством, обеспечивающим в совокупности со средствами разработки приложений Developer/ 2000 поддержку полного ЖЦ ИС для систем, использующих СУБД Oracle.

Пакет CASE/4/0 (microTOOL GmbH), включающий структурные средства системного анализа, проектирования и программирования, обеспечивает поддержку всего жизненного цикла разработки (вплоть до сопровождения), на основе сетевого репозитория, контролирующего целостность проекта и поддерживающего согласованную работу всех его участников (системных аналитиков, проектировщиков, программистов).

Локальные средства

Пакет ERWin (Logic Works) используется при моделировании и создании баз данных произвольной сложности на основе диаграмм "сущность—связь". В настоящее время ERWin является наиболее популярным пакетом моделирований данных благодаря поддержке широкого спектра СУБД самых различных классов — SQL-серверов (Oracle, Informix, Sybase SQL Server, MS SQL Server, Progress, DB2, SQLBase, Ingress, Rdb и др.) и "настольных" СУБД типа xBase (Clipper, dBase, FoxPro, MS Access, Paradox и др.).

BPWin — средство функционального моделирования, реализующее методологию IDEFO. Модель в BPWin представляет собой совокупность SADT-диаграмм, каждая из которых описывает отдельный процесс, разбивая его на шаги и подпроцессы.

S-Designer 4.2 (Sybase/Powersoft) представляет собой CASE-средство для проектирования реляционных баз данных. По своим функциональным возможностям и стоимости он близок к CASE-средству ERWin, отличаясь внешне используемой на диаграммах нотацией. S-Designer реализует стандартную методологию моделирования данных и генерирует описание БД для таких СУБД, как Oracle, Informix, Ingres, Sybase, DB2, Microsoft SQL Server и др.

CASE-Аналитик 1.1 (Эйтекс) является практически единственным в настоящее время конкурентоспособным отечественным CASE-средством функционального моделирования и реализует построение диаграмм потоков данных в соответствии с описанной ранее методологией.

Объектно-ориентированные CASE-средства

Rational Rose — CASE-средство фирмы Rational Software Corporation (США) — предназначено для автоматизации этапов анализа и проектирования ИС, а также для генерации кодов на различных языках и выпуска проектной документации. Rational Rose использует синтез-методологию объектно-ориентированного анализа и проектирования, основанную на подходах трех ведущих специалистов в данной области: Буча, Рамбо и Джекобсона. Разработанная ими универсальная нотация для моделирования объектов (язык UML — Unified Modeling Language) является в настоящее время и, очевидно, останется в будущем общепринятым стандартом в области объектно-ориентированного анализа и проектирования. Конкретный вариант Rational Rose определяется языком, на котором генерируются коды программ (C++, Smalltalk, PowerBuilder, Ada, SQLWindows и ObjectPro). Основной вариант – Rational Rose/C++ — позволяет разрабатывать проектную документацию в виде диаграмм и спецификаций, а также генерировать программные коды на C++. Кроме того, Rational Rose содержит средства реинжиниринга программ, обеспечивающие повторное использование программных компонент в новых проектах.

Средства конфигурационного управления

Цель конфигурационного управления (КУ) — обеспечить управляемость и контролируемость процессов разработки и сопровождения ИС. Для этого необходима точная и достоверная информация о состоянии ИС и его компонент в каждый момент времени, а также о всех предполагаемых и выполненных изменениях.

Для решения задач КУ применяются методы и средства, обеспечивающие идентификацию состояния компонент, учет номенклатуры всех компонент и модификаций системы в целом, контроль за вносимыми изменениями в компоненты, структуру системы и ее функции, а также координированное управление развитием функций и улучшением характеристик системы.

Наиболее распространенным средством КУ является PVCS фирмы Intersolv (США), включающее ряд самостоятельных продуктов: PVCS Version Manager, PVCS Tracker, PVCS Configuration Builder и PVCS Notify.

Средства документирования

Для создания документации в процессе разработки АИС используются разнообразные средства формирования отчетов, а также компоненты издательских систем. Обычно средства документирования встроены в конкретные CASE-средства. Исключением являются некоторые пакеты, предоставляющие дополнительный сервис при документировании. Из них наиболее активно используется SoDA (Software Document Automation).

Продукт SoDA предназначен для автоматизации разработки проектной документации на всех фазах ЖЦ ИС. Он позволяет автоматически извлекать разнообразную информацию, получаемую на разных стадиях разработки проекта, и включать ее в выходные документы. При этом контролируется соответствие документации проекту, взаимосвязь документов, обеспечивается их своевременное обновление. Результирующая документация автоматически формируется из множества источников, число которых не ограничено.

Пакет включает в себя графический редактор для подготовки шаблонов документов. Он позволяет задавать необходимый стиль, фон, шрифт, определять расположение заголовков, резервировать места, где будет размещаться извлекаемая из разнообразных источников информация. Изменения автоматически вносятся только в те части документации, на которые они повлияли в программе. Это сокращает время подготовки документации за счет отказа от перегенерации всей документации.

SoDA реализована на базе издательской системы FrameBuilder и предоставляет полный набор средств по редактированию и верстке выпускаемой документации.

Итоговым результатом работы системы SoDA является готовый документ (или книга). Документ может храниться в файле формата SoDA (Frame Builder), который получается в результате генерации документа. Вывод на печать этого документа (или его части) возможен из системы SoDA.

Среда функционирования SoDA — ОС типа UNIX на рабочих станциях Sun SPARCstation, IBM RISC System/6000 или Hewlett Packard HP 9000 700/800.

Средства тестирования

Под тестированием понимается процесс исполнения программы с целью обнаружения ошибок. Регрессионное тестирование — это тестирование, проводимое после усовершенствования функций программы или внесения в нее изменений.

Одно из наиболее развитых средств тестирования QA (новое название – Quality Works) представляет собой интегрированную, многоплатформенную среду для разработки автоматизированных тестов любого уровня, включая тесты регрессии для приложений с графическим интерфейсом пользователя.

QA позволяет начинать тестирование на любой фазе ЖЦ, планировать и управлять процессом тестирования, отображать изменения в приложении и повторно использовать тесты для более чем 25 различных платформ.

В заключение приведем пример комплекса CASE-средств, обеспечивающего поддержку полного ЖЦ ИС. Нецелесообразно сравнивать отдельно взятые CASE-средства, поскольку ни одно из них не решает в целом все проблемы создания и сопровождения ИС. Это подтверждается также полным набором критериев оценки и

выбора, которые затрагивают все этапы ЖЦ ИС. Сравниваться могут комплексы методологически и технологически согласованных инструментальных средств, поддерживающие полный ЖЦ ИС и обеспеченные необходимой технической и методической поддержкой со стороны фирм-поставщиков (отметим, что рациональное комплексирование инструментальных средств разработки ИС является важнейшим условием обеспечения качества этой ИС, причем это замечание справедливо для всех предметных областей).

Лекция №8

Содержание лекции

Принципы построения и этапы проектирования баз данных

Основные понятия и определения

В науке одним из наиболее сложных для строгого определения является понятие "информация". Согласно кибернетическому подходу "информация" — первоначальное сообщение данных, сведений, осведомление и т.п. Кибернетика вывела понятие информации за пределы человеческой речи и других форм коммуникации между людьми, *связала его с целенаправленными системами любой природы.* Информация выступает *в трех формах*: 1) *биологической* (биотоки; связи в генетических механизмах); 2) *машинной* (сигналы в электрических цепях); 3) *социальной* (движение знаний в общественных системах)".

Иными словами, "информация — связь в любых целенаправленных системах, определяющая их целостность, устойчивость, уровень функционирования". Содержание и особенности информации раскрываются указанием действий, в которых она участвует:

хранение (на некотором *носителе* информации);

преобразование (в соответствии с некоторым *алгоритмом*);

передача (с помощью *передатчика* и *приемника* по некоторой *линии связи*).

В соответствии с этим же подходом *"данные — факты и идеи,* представленные в *формализованном виде,* позволяющем *передавать* или *обрабатывать* их при помощи некоторого *процесса* и соответствующих *технических устройств"*.

Толковый словарь по информатике определяет понятия "информация" и "данные" несколько иначе:

"информация — 1) *совокупность знаний о фактических данных и связях между ними;* 2) в вычислительной технике — *содержание, присваиваемое данным* посредством соглашений, распространяющихся на эти данные; *данные,* подлежащие *вводу в ЭВМ, хранимые в ее памяти, обрабатываемые на ЭВМ и выдаваемые пользователям"*;

"данные — информация, представленная в виде, пригодном для обработки автоматическими средствами при возможном участии человека".

Как легко заметить, приведенные определения вынужденно используют такие сложно определяемые понятия, как *"факты", "идеи"* и, особенно, *"знания"*.

В дальнейшем *под информацией будем понимать любые сведения о процессах и явлениях, которые в той или иной форме передаются между объектами материального мира* (*людьми; животными; растениями; автоматами и др*.).

Если рассмотреть некоторый *объект материального мира,* информация о котором представляет интерес, и *наблюдателя* (в роли которого и выступают ИС), *способного фиксировать* эту информацию в определенной, понятной другим форме, то говорят, что *в памяти ("сознании") наблюдателя находятся данные,* описывающие состояние объекта. Таким образом, *данными будем называть формализованную информацию, пригодную для последующей обработки, хранения и передачи средствами автоматизации профессиональной деятельности.*

Информацию в ЭВМ можно хранить в виде различных данных (числовых; текстовых; визуальных и т.п.). Более того, для описания одной и той же информации можно предложить различные варианты их состава и структуры. Иными словами, *правомерно говорить о моделировании в ИС информации о некотором множестве объектов материального мира совокупностью взаимосвязанных данных*.

Информационное обеспечение (information support) ИС – совокупность единой *системы классификации и кодирования* информации; унифицированных *систем документации и используемых массивов информации*.

В этой связи *в качестве главных задач создания информационного обеспечения* ИС можно выделить:

- во-первых, *определение состава и структуры данных,* достаточно "хорошо" описывающих требуемую информацию;
- во-вторых, *обоснование способов хранения и переработки* данных с использованием ЭВМ.

Остановимся на понятиях и определениях, связанных с технологией банков данных.

Прежде чем определить понятие "банк данных", необходимо остановиться на другом ключевом понятии — "предметная область".

Под *предметной областью* будем понимать *информацию об объектах, процессах и явлениях окружающего мира,* которая, *с точки зрения потенциальных пользователей, должна храниться и обрабатываться в информационной системе*.

Банк *данных* (БнД) — *информационная система, включающая в свой состав комплекс специальных методов и средств для поддержания динамической информационной модели с целью обеспечения информационных потребностей пользователей*. Очевидно, что БнД может рассматриваться как специальная обеспечивающая подсистема в составе старшей по иерархии ИС.

Поддержание динамической модели предметной области (ПрОбл) предусматривает не только *хранение* информации о ней и своевременное *внесение изменений* в соответствии с реальным состоянием объектов, но и обеспечение возможности учета *изменений состава* этих объектов (в том числе появление новых) и *связей* между ними (т.е. изменений самой структуры хранимой информации).

Обеспечение информационных потребностей (запросов) пользователей имеет два аспекта:

- *определение границ* конкретной ПрОбл и *разработка описания* соответствующей информационной *модели*;
- *разработка БнД,* ориентированного на *эффективное обслуживание запросов* различных категорий пользователей.

С точки зрения *целевой направленности* профессиональной деятельности принято выделять пять основных категорий пользователей:

- аналитики;
- системные программисты;
- прикладные программисты;
- администраторы;
- конечные пользователи.

Кроме того, различают пользователей *постоянных и разовых;* пользователей-людей и пользователей-задач; пользователей с различным *уровнем компетентности (приоритетом) и* др., причем каждый класс пользователей предъявляет собственные специфические требования к своему обслуживанию (прежде всего – с точки зрения организации диалога "запрос—ответ"). Так, например, постоянные пользователи, как правило, обращаются в БнД с фиксированными по форме (типовыми) запросами; пользователи-задачи должны иметь возможность получать информацию из БнД в согласованной форме в указанные области памяти; пользователи с низким приоритетом могут получать ограниченную часть информации и т.д. Наличие столь разнообразного состава потребителей информации потребовало включения в БнД специального элемента — *словаря данных*.

Уровень сложности и важности задач информационного обеспечения ИС в рамках рассматриваемой технологии определяет ряд *основных требований* к БнД:

- *адекватность* информации состоянию предметной области;
- *быстродействие и производительность*;
- *простота и удобство* использования;
- *массовость* использования;
- *защита* информации;
- *возможность расширения* круга решаемых задач.

По сравнению с традиционным обеспечением монопольными файлами каждого приложения централизованное управление данными в БнД имеет *ряд важных преимуществ*:

- *сокращение избыточности* хранимых данных;
- *устранение противоречивости* хранимых данных;
- *многоаспектное использование* данных (при однократном вводе);
- *комплексная оптимизация* (с точки зрения удовлетворения разнообразных, в том числе и противоречивых, требований "в целом");
- обеспечение *возможности стандартизации*;
- обеспечение *возможности санкционированного доступа* к данным и др.

Все названные преимущества по существу связаны с такими основополагающими *принципами* концепции БнД, как *интеграция* данных, *централизация управления* ими и *обеспечение независимости прикладных программ* обработки данных и самих данных.

Структура типового БнД, удовлетворяющего предъявляемым требованиям, приведена на рис. 1, где представлены:

ВС — вычислительная система, включающая технические средства (ТС) и общее программное обеспечение (ОПО);

БД – базы данных;

СУБД – система управления БД;

АБД – администратор баз данных, а также обслуживающий *персонал* и *словарь данных*.

Рассмотрим составляющие БнД, представляющие наибольший интерес.

БД – *совокупность* специальным образом *организованных (структурированных)* данных и связей между ними. Иными словами, БД — это так называемое датологическое (от англ, data — данные) *представление информации о предметной области.* Если в состав БнД входит одна БД, банк принято называть *локальным*; если БД несколько — *интегрированным*.

Рис. 1. Основные компоненты БнД.

СУБД — специальный комплекс программ и языков, посредством которого организуется централизованное управление базами данных и обеспечивается доступ к ним. В состав любой СУБД входят *языки двух типов*: 1) язык *описания данных* (с его помощью описываются типы данных, их структура и связи); 2) язык *манипулирования данными* (его часто называют *язык запросов к* БД), предназначенный для организации работы с данными в интересах всех типов пользователей.

Словарь *данных* предназначен для хранения единообразной и централизованной *информации обо всех ресурсах данных* конкретного банка:

- об объектах, их свойствах и *отношениях* для данной ПрОбл;
- о *данных,* хранимых в БД (наименование; смысловое описание; структура; связи и т.п.);
- о возможных *значениях* и *форматах* представления данных;
- об *источниках* возникновения данных;
- о кодах *защиты* и *разграничении доступа* пользователей к данным и т. п.

Администратор баз данных — это лицо (группа лиц), реализующее *управление БД*. В этой связи сам БнД можно рассматривать как *автоматизированную систему управления базами данных.* Функции АБД являются долгосрочными; он координирует все виды работ на этапах создания и применения БнД. На стадии проектирования АБД выступает как идеолог и главный конструктор системы; на стадии эксплуатации он отвечает за нормальное функционирование БнД, управляет режимом его работы и обеспечивает безопасность данных.

Основные функции АБД:

- решать вопросы организации данных об объектах ПрОбл и установления связей между этими данными с целью объединения информации о различных объектах; согласовывать представления пользователей;
- координировать все действия по проектированию, реализации и ведению БД; учитывать текущие и перспективные требования пользователей; следить, чтобы БД удовлетворяли актуальным потребностям;
- решать вопросы, связанные с расширением БД в связи с изменением границ ПрОбл;
- разрабатывать и реализовывать меры по обеспечению защиты данных от некомпетентного их использования, от сбоев технических средств, по обеспечению секретности определенной части данных и разграничению доступа к ним;
- выполнять работы по ведению словаря данных; контролировать избыточность и противоречивость данных, их достоверность;
- следить за тем, чтобы БнД отвечал заданным требованиям по производительности, т. е. чтобы обработка запросов выполнялась за приемлемое время;

- выполнять при необходимости изменения методов хранения данных, путей доступа к ним, связей между данными, их форматов; определять степень влияния изменений в данных на всю БД;
- координировать вопросы технического обеспечения системы аппаратными средствами, исходя из требований, предъявляемых БД к оборудованию;
- координировать работы системных программистов, разрабатывающих дополнительное программное обеспечение для улучшения эксплуатационных характеристик системы;
- координировать работы прикладных программистов, разрабатывающих новые прикладные программы, и выполнять их проверку и включение в состав ПрОбл системы.

На рис. 2 представлен типовой состав группы АБД, отражающий основные направления деятельности специалистов.

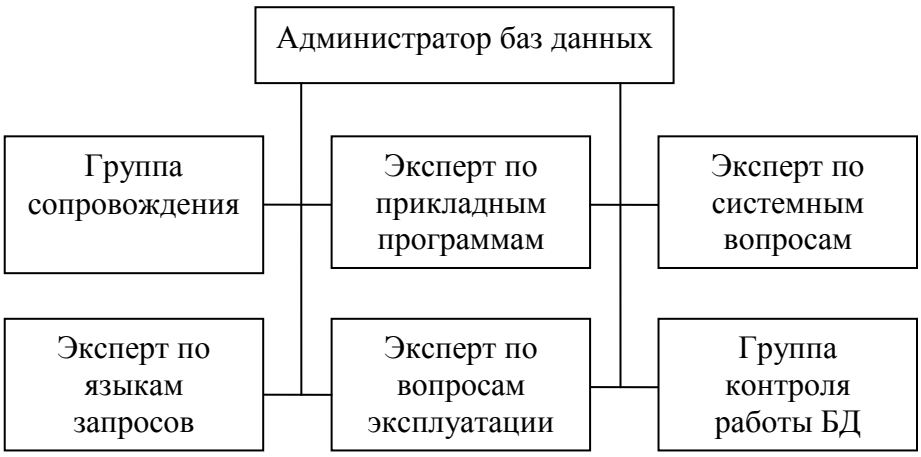

Рис. 2. Типовой состав группы АБД.

Описательная модель предметной области

Процесс проектирования БД является весьма сложным. По сути, он заключается в определении перечня данных, хранимых на физических носителях (магнитных дисках и т.п.), которые достаточно полно отражают информационные потребности потенциальных пользователей в конкретной ПрОбл. Проектирование БД начинается с анализа предметной области и возможных запросов пользователей. В результате этого анализа определяется перечень данных и связей между ними, которые адекватно — с точки зрения будущих потребителей — отражают ПрОбл. Завершается проектирование БД определением форм и способов хранения необходимых данных на физическом уровне.

Весь процесс проектирования БД можно разбить на ряд взаимосвязанных этапов, каждый из которых обладает своими особенностями и методами проведения. На рис. 3 представлены типовые этапы.

На этапе *инфологического (информационно-логического)* проектирования осуществляется построение семантической модели, описывающей сведения из

предметной области, которые могут заинтересовать пользователей БД. Семантическая модель *(semantic model)* — представление совокупности понятий о ПрОбл в виде *графа,* в вершинах которого расположены *понятия,* в терминальных вершинах — элементарные понятия, а *дуги* представляют *отношения между понятиями.*

Сначала из объективной реальности выделяется ПрОбл, т.е. очерчиваются ее границы. Логический анализ выделенной ПрОбл и потенциальных запросов пользователей завершается построением инфологической модели ПрОбл — перечня сведений об объектах ПрОбл, которые необходимо хранить в БД и связях между ними.

Анализ информационных потребностей потенциальных пользователей имеет два аспекта: 1) *определение* собственно *сведений об объектах* ПрОбл; 2) *анализ возможных запросов к БД* и требований по оперативности их выполнения.

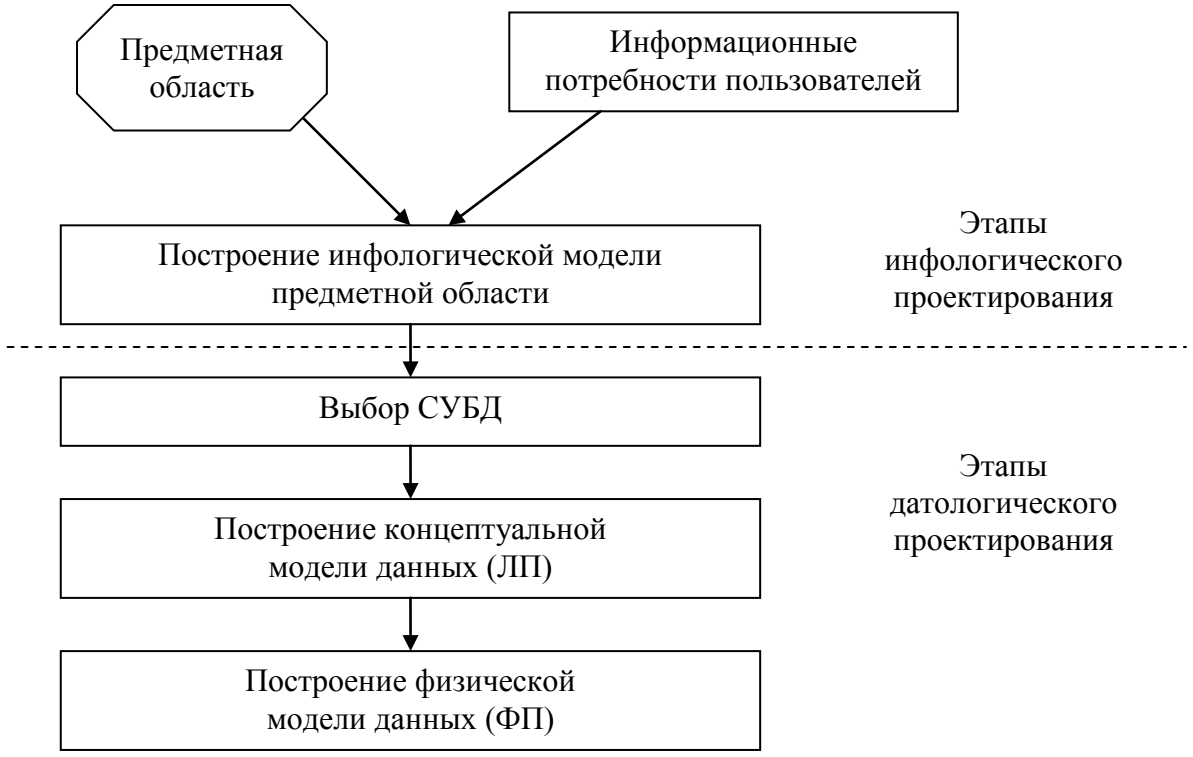

Рис. 3. Этапы проектирования БД.

Анализ возможных запросов к БД позволяет уточнить связи между сведениями, которые необходимо хранить. Пусть, например, в БД по учебному процессу института хранятся сведения об учебных группах, читаемых курсах и кафедрах, а также связи "учебные группы—читаемые курсы" и "читаемые курсы—кафедры". Тогда запрос о том, проводит ли некоторая кафедра занятия в конкретной учебной группе может быть выполнен только путем перебора всех читаемых в данной группе курсов.

Хранение большого числа связей усложняет БД и приводит к увеличению потребной памяти ЭВМ, но часто существенно ускоряет поиск нужной информации. Поэтому разработчику БД (АБД) приходится принимать компромиссное решение, причем

процесс определения перечня хранимых связей, как правило, имеет итерационный характер.

Датологическое проектирование подразделяется на *логическое* (построение концептуальной модели данных) и *физическое* (построение физической модели) проектирование.

Главной задачей логического проектирования (ЛП) БД является представление выделенных на предыдущем этапе сведений в виде данных в форматах, поддерживаемых выбранной СУБД.

Задача физического проектирования (ФП) — выбор способа хранения данных на физических носителях и методов доступа к ним с использованием возможностей, предоставляемых СУБД.

Инфологическая модель "сущность—связь" (entity relationship model; ER-model) П. Чена (P. Chen) представляет собой описательную (неформальную) модель ПрОбл, семантически определяющую в ней сущности и связи.

Относительная простота и наглядность описания ПрОбл позволяет использовать ее в процессе диалога с потенциальными пользователями с самого начала инфологического проектирования. Построение инфологической модели П. Чена, как и любой другой модели, является творческим процессом, по этому единой методики ее создания нет. Однако при любом подходе к построению модели используют три основных конструктивных элемента:

- сущность;
- атрибут;
- связь.

Сущность — это собирательное понятие некоторого повторяющегося объекта, процесса или явления окружающего мира, о котором необходимо хранить информацию в системе. Сущность может определять как материальные (например, "студент", "грузовой автомобиль" и т.п.), так и нематериальные объекты (например, "экзамен", "проверка" и т.п.). Главной особенностью сущности является то, что вокруг нее сосредоточен сбор информации в конкретной ПрОбл. Тип сущности определяет *набор однородных объектов,* а *экземпляр сущности — конкретный объект* в наборе. Каждая сущность в модели Чена именуется. Для идентификации конкретного экземпляра сущности и его описания используется один или несколько атрибутов.

Атрибут — это поименованная характеристика сущности, которая принимает значения из некоторого множества значений. Например, у сущности "студент" могут быть атрибуты "фамилия", "имя", "отчество", "дата рождения", "средний балл за время обучения" и т.п.

Связи в инфологической модели выступают в качестве средства, с помощью которого представляются отношения *между сущностями,* имеющими место в ПрОбл. При анализе связей между сущностями могут встречаться бинарные (между двумя сущностями) и, в общем случае, *n*-арные (между *n* сущностями) связи. Например, сущности "отец", "мать" и "ребенок" могут находиться в триарном отношении "семья" ("является членом семьи").

Связи должны быть поименованы; между двумя типами сущностей могут существовать несколько связей.

Наиболее распространены бинарные связи. Любую *n*-арную связь можно представить в виде нескольких бинарных.

Различают четыре типа связей:

- связь один к одному (1:1);
- связь один ко многим (1:M);
- связь многие к одному (M:1);
- связь многие ко многим (M:N).

Связь *один к одному* определяет такой тип связи между типами сущностей A и B, при которой каждому экземпляру сущности A соответствует один и только один экземпляр сущности B, и наоборот. Таким образом, имея некоторый экземпляр сущности A, можно однозначно идентифицировать соответствующий ему экземпляр сущности B, а по экземпляру сущности B — экземпляр сущности A. Например, связь типа 1:1 ("имеет") может быть определена между сущностями "автомобиль" и "двигатель", так как на конкретном автомобиле может быть установлен только один двигатель, и этот двигатель, естественно, нельзя установить сразу на несколько автомобилей.

Связь *один ко многим* определяет такой тип связи между типами сущностей A и B, для которой одному экземпляру сущности A может соответствовать 0, 1 или несколько экземпляров сущности B, но каждому экземпляру сущности B соответствует один экземпляр сущности A. При этом однозначно идентифицировать можно только экземпляр сущности A по экземпляру сущности B. Примером связи типа 1:M является связь "учится" между сущностями "учебная группа" и "студент". Для такой связи, зная конкретного студента, можно однозначно идентифицировать учебную группу, в которой он учится, или, зная учебную группу, можно определить всех обучающихся в ней студентов.

Связь *многие к одному* по сути эквивалентна связи один ко многим. Различие заключается лишь в том, с точки зрения какой сущности (A или B) данная связь рассматривается.

Связь *многие ко многим* определяет такой тип связи между типами сущностей A и B, при котором каждому экземпляру сущности A может соответствовать 0, 1 или несколько экземпляров сущности B, и наоборот. При такой связи, зная экземпляр одной

сущности, можно указать все экземпляры другой сущности, относящиеся к исходному, т.е. идентификация сущностей не уникальна в обоих направлениях. В качестве примера такой связи можно рассмотреть связь "изучает" между сущностями "учебная дисциплина" и "учебная группа".

Реально *все связи являются двунаправленными,* т.е., зная экземпляр одной из сущностей, можно идентифицировать (однозначно или многозначно) экземпляр (экземпляры) другой сущности. В некоторых случаях целесообразно рассматривать лишь однонаправленные связи между сущностями в целях экономии ресурсов ЭВМ. Возможность введения таких связей полностью определяется информационными потребностями пользователей. Различают простую и многозначную однонаправленные связи, которые являются аналогами связей типа 1:1 и 1:M с учетом направления идентификации. Так, для простой однонаправленной связи "староста" ("является старостой") между сущностями "учебная группа" и "студент" можно, зная учебную группу, однозначно определить ее старосту, но, зная конкретного студента, нельзя сказать, является ли он старостой учебной группы. Примером многозначной однонаправленной связи служит связь между сущностями "пациент" и "болезнь", для которой можно для каждого пациента указать его болезни, но нельзя выявить всех обладателей конкретного заболевания.

Графически типы сущностей, атрибуты и связи принято изображать прямоугольниками, овалами и ромбами соответственно. На рис. 4 представлены примеры связей различных типов; на рис. 5 и 6 – фрагменты инфологических моделей "военнослужащий" (без указания атрибутов) и "учебный процесс факультета".

Несмотря на то, что построение инфологической модели есть процесс творческий, можно указать *два основополагающих правила,* которыми следует пользоваться всем проектировщикам БД:

- при построении модели *должны использоваться только три типа конструктивных элементов: сущность, атрибут, связь*;
- *каждый компонент информации должен моделироваться только одним из приведенных выше конструктивных элементов* для исключения избыточности и противоречивости описания.

Моделирование ПрОбл *начинают с выбора сущностей,* необходимых для ее описания. Каждая сущность должна соответствовать некоторому объекту (или группе объектов) ПрОбл, о котором в системе будет накапливаться информация. Существует проблема выбора конструктивного элемента для моделирования той или иной "порции" информации, что существенно затрудняет процесс построения модели.

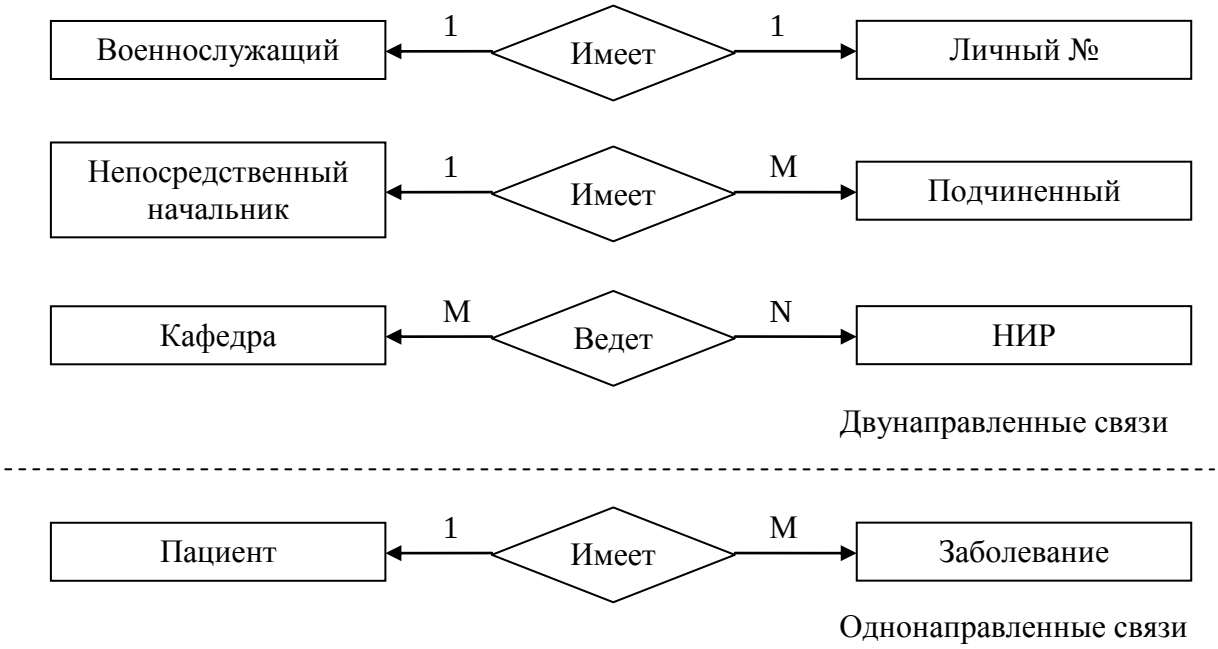

Рис. 4. Примеры связей между сущностями.

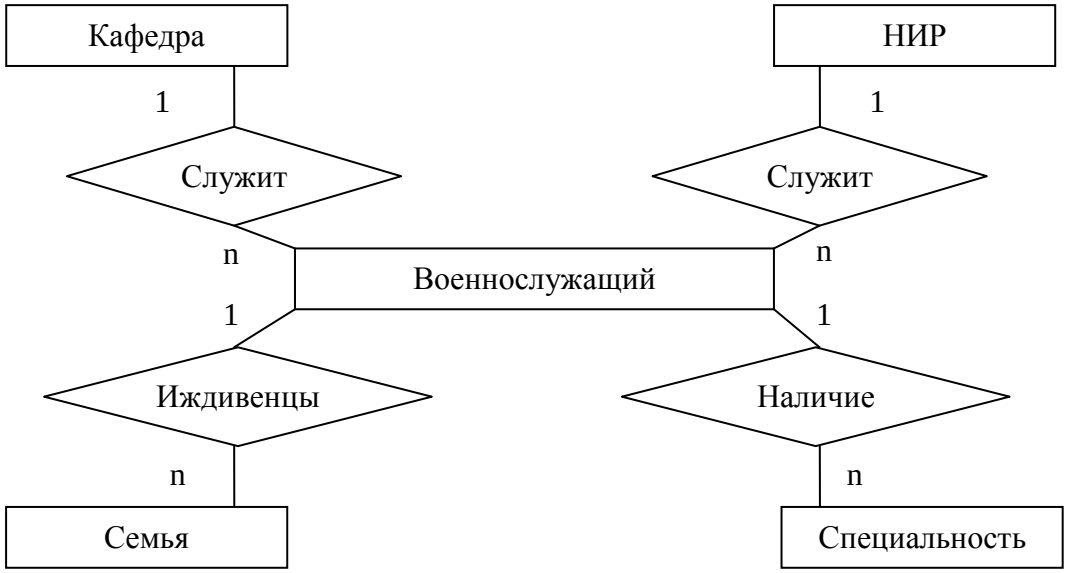

Рис. 5. Фрагмент ER-модели "Военнослужащий".

Так, информация о том, что некоторый студент входит в состав учебной группы (УГ) можно в модели представить:

- как связь "входит в состав" для сущностей "студент" и "УГ";
- как атрибут "имеет в составе "студента" сущности "УГ";
- как сущность "состав УГ".

В этих случаях приходится рассматривать несколько вариантов и с учетом информационных потребностей пользователей разбивать ПрОбл на такие фрагменты, которые с их точки зрения представляют самостоятельный интерес.

Рис. 6. Фрагмент ER-модели "Учебный процесс факультета".

При описании атрибутов сущности необходимо выбрать *ряд атрибутов, позволяющих однозначно идентифицировать экземпляр сущности. Совокупность идентифицирующих атрибутов называют ключом.*

Помимо идентифицирующих используются и *описательные атрибуты,* предназначенные для более полного определения сущностей. Число атрибутов (их тип) определяется единственным образом — на основе анализа возможных запросов пользователей. Существует ряд рекомендаций по "работе с атрибутами", например, по исключению повторяющихся групп атрибутов (см. рис. 7).

При определении связей между сущностями следует избегать связей типа M:N, так как они приводят к существенным затратам ресурсов ЭВМ. Устранение таких связей предусматривает введение других (дополнительных) элементов — сущностей и связей. На рис. 8 приведен пример исключения связи многие ко многим.

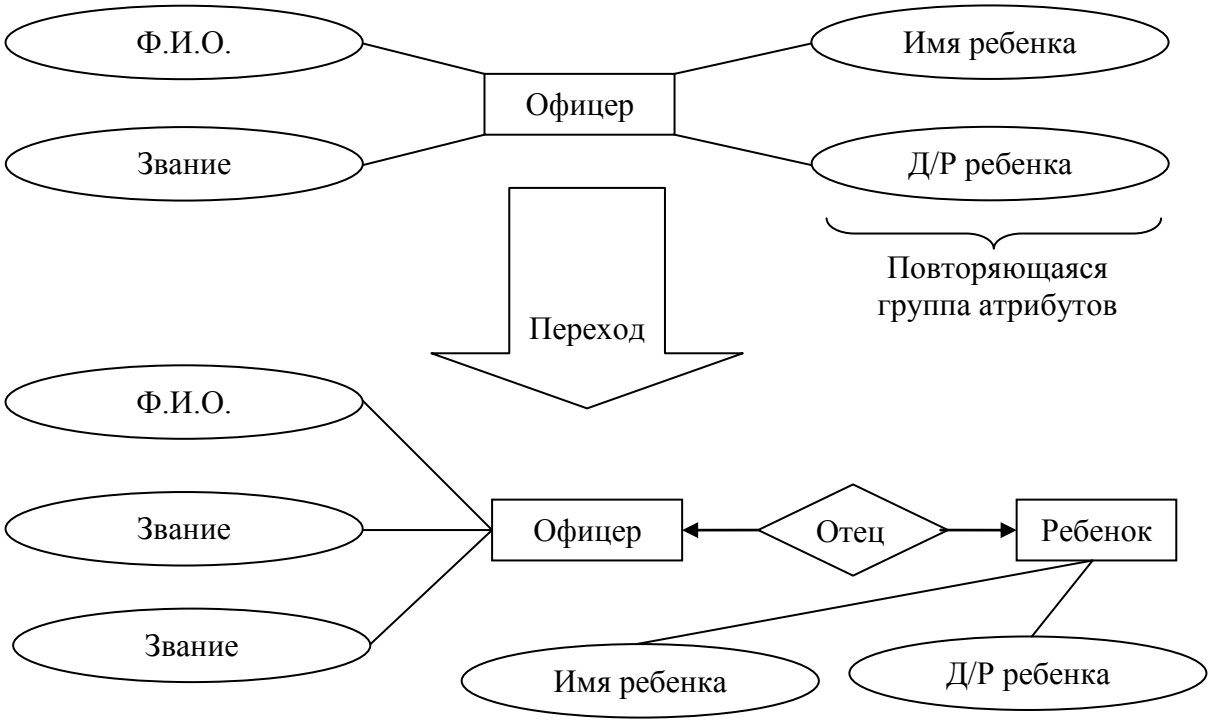

Рис. 7. Пример исключения повторяющейся группы атрибутов.

Приведем типовую последовательность работ (действий) по построению инфологической модели:

- *выделение* в ПрОбл *сущностей*;
- *введение множества атрибутов* для каждой сущности и выделение из них *ключевых*;

- *исключение множества повторяющихся атрибутов* (при необходимости);
- *формирование связей* между сущностями;
- *исключение связей типа M:N* (при необходимости);
- *преобразование связей в однонаправленные* (по возможности).

Помимо модели Чена существуют и другие инфологические модели. Все они представляют собой описательные (неформальные) модели, использующие различные конструктивные элементы и соглашения по их использованию для представления в БД информации о ПрОбл. Иными словами, первый этап построения БД всегда связан с моделированием предметной области.

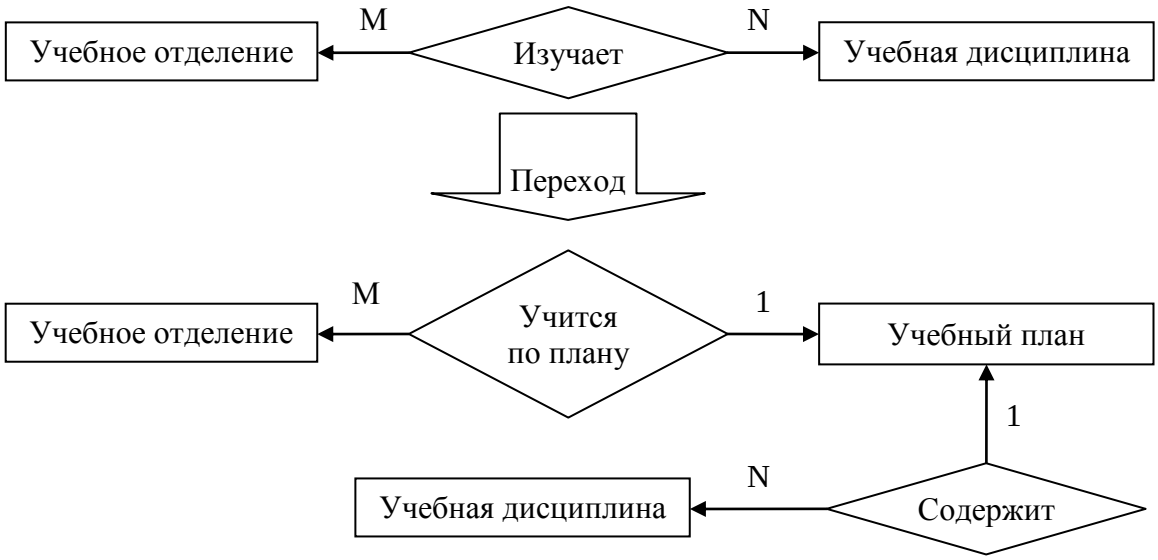

Рис. 8. Пример исключения связи типа M:N.

Лекция №9

Содержание лекции

Принципы построения и этапы проектирования баз данных

Концептуальные модели данных

В отличие от инфологической модели ПрОбл, описывающей по некоторым правилам сведения об объектах материального мира и связи между ними, которые следует иметь в БД, *концептуальная модель описывает хранимые в ЭВМ данные и связи. В силу этого каждая модель данных неразрывно связана с языком описания данных* конкретной СУБД (см. Лекция №8 рис. 3).

По существу модель данных — это совокупность трех составляющих:

- *типов (структур)* данных;
- *операций* над данными;
- *ограничений* целостности.

Другими словами, модель данных представляет собой некоторое интеллектуальное средство проектировщика, позволяющее реализовать интерпретацию сведений о ПрОбл в виде формализованных данных в соответствии с определенными требованиями, т.е. средство абстракции, которое дает возможность увидеть "лес" (информационное содержание данных), а не отдельные "деревья" (конкретные значения данных).

Типы структур данных

Среди широкого множества определений, обозначающих типы структур данных, наиболее распространена терминология КОДАСИЛ (Conference of DAta SYstems Language) – международной ассоциации по языкам систем обработки данных, созданной в 1959 г.

В соответствии с этой терминологией используют пять типовых структур (в порядке усложнения):

- *элемент данных*;
- *агрегат данных*;
- *запись*;
- *набор*;
- *база данных*.

Дадим краткие определения этих структур.

Элемент данных — наименьшая поименованная единица данных, к которой СУБД может адресоваться непосредственно и с помощью которой выполняется построение всех остальных структур данных.

Агрегат данных — поименованная совокупность элементов данных, которую можно рассматривать как единое целое. Агрегат может быть простым или составным (если он включает в себя другие агрегаты).

Запись — поименованная совокупность элементов данных и (или) агрегатов. Таким образом, запись – это агрегат, не входящий в другие агрегаты. Запись может иметь сложную иерархическую структуру, поскольку допускает многократное применение агрегации.

Набор — поименованная совокупность записей, образующих двухуровневую иерархическую структуру. Каждый тип набора представляет собой связь между двумя типами записей. Набор определяется путем объявления одного типа записи "записью-владельцем", а других типов записей — "записями-членами". При этом каждый экземпляр набора должен содержать один экземпляр "записи-владельца" и любое количество "записей-членов". Если запись представляет в модели данных сущность, то набор — связь между сущностями. Например, если рассматривать связь "учится" между сущностями "учебная группа" и "студент", то первая из сущностей объявляется "записью-владельцем" (она в экземпляре набора одна), а вторая – "записью-членом" (их в экземпляре набора может быть несколько).

База данных поименованная совокупность экземпляров записей различного типа, содержащая ссылки между записями, представленные экземплярами наборов.

Отметим, что структуры БД строятся на основании следующих основных композиционных правил:

- БД может содержать *любое количество типов записей и типов наборов*;
- между *двумя типами записей* может быть определено *любое количество наборов*;
- тип записи может быть *владельцем* и *одновременно членом* нескольких типов наборов.

Следование данным правилам позволяет моделировать данные о сколь угодно сложной ПрОбл с требуемым уровнем полноты и детализации.

Рассмотренные типы структур данных могут быть представлены в различной форме — графовой; табличной; в виде исходного текста языка описания данных конкретной СУБД.

Операции над данными

Операции, реализуемые СУБД, включают *селекцию* (поиск) данных; *действия* над данными.

Селекция данных выполняется с помощью критерия, основанного на использовании либо логической позиции данного (элемента; агрегата; записи), либо значения данного, либо связей между данными.

Селекция на основе логической позиции данного базируется на упорядоченности данных в памяти системы. При этом критерии поиска могут формулироваться следующим образом:

- найти следующее данное (запись);
- найти предыдущее данное;
- найти *n*-ое данное;
- найти первое (последнее) данное. Этот тип селекции называют *селекцией посредством текущей,* в качестве которой используется индикатор текущего состояния, автоматически поддерживаемый СУБД и, как правило, указывающий на некоторый экземпляр записи БД.

Критерий селекции по значениям данных формируется из простых или булевых условий отбора. Примерами простых условий поиска являются:

- ВУС = 200100;
- ВОЗРАСТ > 20;
- ДАТА < 19.04.2002 и т. п.

Булево условие отбора формируется путем объединения простых условий с применением логических операций, например:

- (ДАТА_РОЖДЕНИЯ < 28.12.1963) И (СТАЖ> 10);
- (УЧЕНОЕ_ЗВАНИЕ=ДОЦЕНТ:) ИЛИ (УЧЕНОЕЗВАНИЕ-ПРОФЕССОР).

Если модель данных, поддерживаемая некоторой СУБД, позволяет выполнить селекцию данных по связям, то можно найти данные, связанные с текущим значением какого-либо данного. Например, если в модели данных реализована двунаправленная связь "учится" между сущностями "студент" и "учебная группа", можно выявить учебные группы, в которых учатся юноши (если в составе описания студента входит атрибут "пол").

Как правило, большинство современных СУБД позволяет осуществлять различные комбинации описанных выше видов селекции данных.

Ограничения целостности

Ограничения целостности — логические ограничения на данные — используются для обеспечения непротиворечивости данных некоторым заранее заданным условиям при выполнении операций над ними. По сути ограничения целостности — это набор правил, используемых при создании конкретной модели данных на базе выбранной СУБД.

Различают *внутренние* и *явные* ограничения.

Ограничения, обусловленные возможностями конкретной СУБД, называют внутренними ограничениями целостности. Эти ограничения касаются типов хранимых данных (например, "текстовый элемент данных может состоять не более чем из 256 символов" или "запись может содержать не более 100 полей") и допустимых типов связей (например, СУБД может поддерживать только так называемые функциональные связи, т.е. связи типа 1:1, 1:М или М:1). Большинство существующих СУБД поддерживают, прежде всего, именно внутренние ограничения целостности, нарушения которых приводят к некорректности данных и достаточно легко контролируются.

Ограничения, обусловленные особенностями хранимых данных о конкретной ПрОбл, называют явными ограничениями целостности. Эти ограничения также поддерживаются средствами выбранной СУБД, но они формируются обязательно с участием разработчика БД путем определения (программирования) специальных процедур, обеспечивающих непротиворечивость данных. Например, если элемент данных "зачетная книжка" в записи "студент" определен как ключ, он должен быть уникальным, т.е. в БД не должно быть двух записей с одинаковыми значениями ключа. Другой пример: пусть в той же записи предусмотрен элемент "военно-учетная специальность" и для него отведено 6 десятичных цифр. Тогда другие представления этого элемента данных в БД невозможны. С помощью явных ограничений целостности можно организовать как "простой" контроль вводимых данных (прежде всего, на предмет принадлежности элементов данных фиксированному и заранее заданному множеству значений: например, элемент "ученое звание" не должен принимать значение "почетный доцент", если речь идет о российских ученых), так и более сложные процедуры (например, введение значения "профессор" элемента данных "ученое звание" в запись о преподавателе, имеющем возраст 25 лет, должно требовать, по крайней мере, дополнительного подтверждения).

Разнообразие существующих моделей данных соответствует разнообразию областей применения и предпочтений пользователей.

В специальной литературе встречается описание довольно большого количества различных моделей данных. Хотя наибольшее распространение получили иерархическая, сетевая и, бесспорно, реляционная модели, вместе с ними следует упомянуть и некоторые другие.

Используя в качестве классификационного признака особенности логической организации данных, можно привести следующий перечень известных моделей данных:

- иерархическая;
- сетевая;
- реляционная;
- бинарная;
- семантическая сеть.

Иерархическая модель данных

Наиболее давно используемой (можно сказать классической) является модель данных, в основе которой лежит *иерархическая структура типа дерева.* Дерево — орграф, в каждую вершину которого, кроме первой (корневой), входит только одна дуга, а из любой вершины (кроме конечных) может исходить произвольное число дуг. В иерархической структуре подчиненный элемент данных всегда связан только с одним исходным.

На рис. 1 показан фрагмент объектной записи в иерархической модели данных. Часто используется также "упорядоченное дерево", в котором значим относительный порядок поддеревьев.

Достоинства такой модели несомненны: простота представления предметной области, наглядность, удобство анализа структур и простота их описания. К *недостаткам* следует отнести сложность добавления новых и удаления существующих типов записей, невозможность отображения отношений, отличающихся от иерархических, громоздкость описания и информационную избыточность.

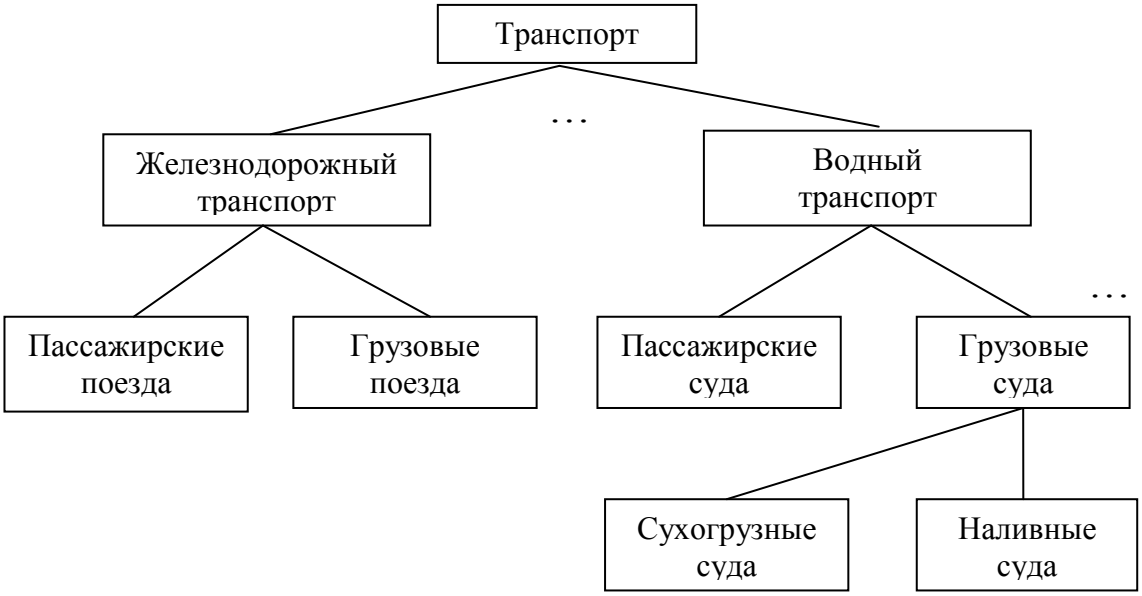

Рис. 1. Фрагмент иерархической модели данных.

Характерные примеры реализации иерархических структур — язык Кобол и СУБД семейства IMS (создана в рамках проекта высадки на Луну – "Аполлон") и System 2000 (S2K).

Сетевая модель данных

В системе баз данных, предложенных CODASYL, за основу была взята сетевая структура. Существенное влияние на разработку этой модели оказали ранние сетевые системы — IDS и Ассоциативный ПЛ/1. Необходимость в процессе получения одного отчета обрабатывать несколько файлов обусловила целесообразность установления перекрестных ссылок между файлами, что, в конце концов, и привело к сетевым структурам.

Сетевая модель данных основана на представлении информации в виде орграфа, в котором в каждую вершину может входить *произвольное число дуг.* Вершинам графа сопоставлены типы записей, дугам — связи между ними. На рис. 2. представлен пример структуры сетевой модели данных.

По сравнению с иерархическими сетевые модели обладают рядом существенных *преимуществ*: возможность отображения практически всего многообразия взаимоотношений объектов предметной области, непосредственный доступ к любой вершине сети (без указания других вершин), малая информационная избыточность. Вместе с тем в сетевой модели невозможно достичь полной независимости данных — с ростом объема информации сетевая структура становится весьма сложной для описания и анализа.

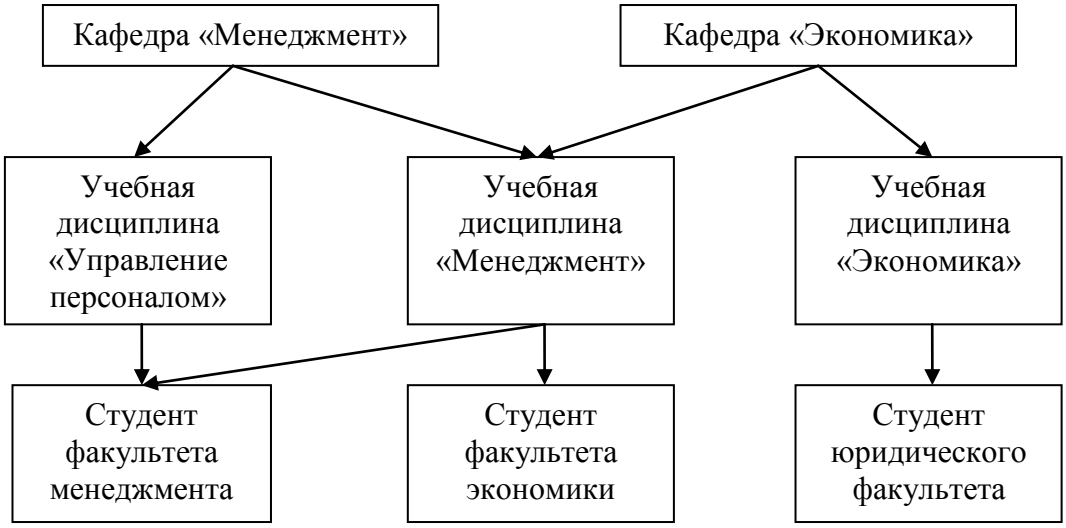

Рис. 2. Фрагмент сетевой модели данных.

Известно, что применение на практике иерархических и сетевых моделей данных в некоторых случаях требует разработки и сопровождения значительного объема кода приложения, что иногда может стать для информационной системы непосильным бременем.

Реляционная модель данных

В основе реляционной модели данных лежат не графические, а *табличные методы и средства представления данных* и манипулирования ими (рис. 3). В реляционной модели для отображения информации о предметной области используется таблица, называемая "отношением". Строка такой таблицы называется кортежем, столбец — атрибутом. Каждый атрибут может принимать некоторое подмножество значений из определенной области — домена.

Таблица организации БД позволяет реализовать ее важнейшее преимущество перед другими моделями данных, а именно — возможность использования точных математических методов манипулирования данными, и прежде всего — аппарата реляционной алгебры и исчисления отношений, К другим достоинствам реляционной модели можно отнести наглядность, простоту изменения данных и организации разграничения доступа к ним.

Основным недостатком реляционной модели данных является информационная избыточность, что ведет к перерасходу ресурсов вычислительных систем. Однако именно реляционная модель данных находит все более широкое применение в практике автоматизации информационного обеспечения профессиональной деятельности.

Вуз	Место расположения	Количество обучаемых

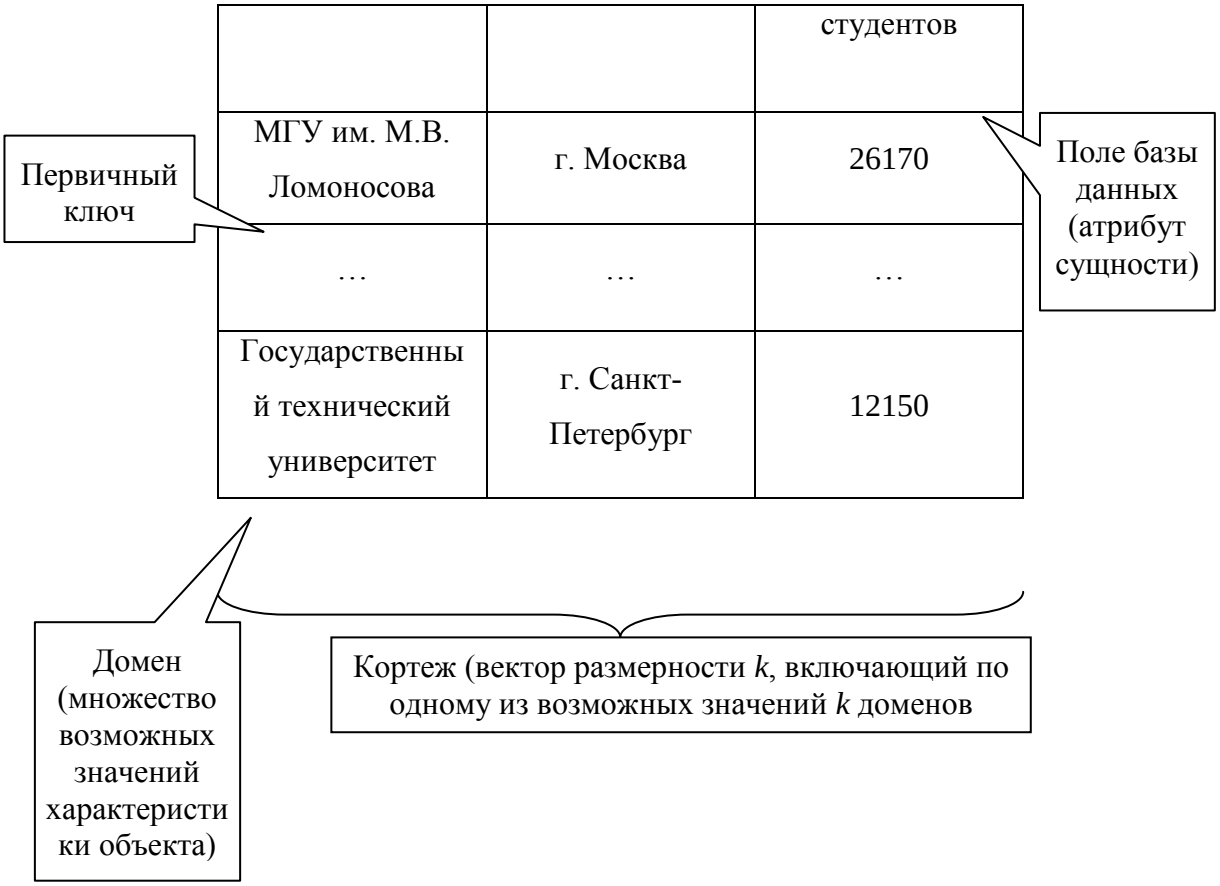

Рис. 3. Фрагмент реляционной модели данных.

Подавляющее большинство СУБД, ориентированных на персональные ЭВМ, являются системами, построенными на основе реляционной модели данных — реляционными СУБД.

Бинарная модель данных

Бинарная модель данных — это графовая модель, в которой *вершины являются представлениями простых однозначных атрибутов, а дуги представлениями бинарных связей между атрибутами* (см. рис. 4).

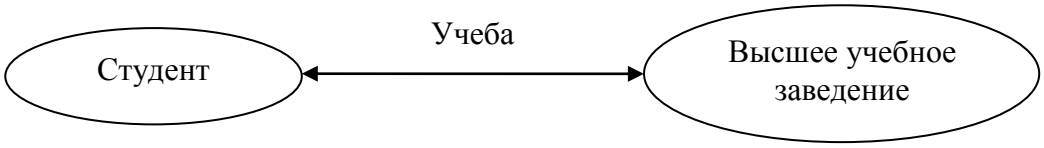

Рис. 4. Пример бинарного отношения.

Бинарная модель не получила особо широкого распространения, но в ряде случаев находит практическое применение. Так, в области искусственного интеллекта уже давно ведутся исследования с целью представления информации в виде бинарных отношений. Рассмотрим триаду (тройку) "объект—атрибут—значение". Триада "Кузнецов—возраст—

20" означает, что возраст некоего Кузнецова равен 20 годам. Эта же информация может быть выражена, например, бинарным отношением ВОЗРАСТ. Понятие бинарного отношения положено в основу таких моделей данных, как, например, Data Semantics (автор Абриал) и DIAM II (автор Сенко).

Бинарные модели данных обладают возможностью представления связки любой сложности (и это их несомненное преимущество), но вместе с тем их ориентация на пользователя недостаточна.

Семантическая сеть

Семантические сети как модели данных были предложены исследователями, работавшими над различными проблемами *искусственного интеллекта*. Так же, как в сетевой и бинарной моделях, базовые структуры семантической сети могут быть представлены графом, множество вершин и дуг которого образует сеть. Однако семантические сети предназначены для представления и систематизации знаний самого общего характера.

Таким образом, семантической сетью можно считать любую графовую модель (например — помеченный бинарный граф) если изначально четко определено, что обозначают вершины и дуги и как они используются.

Семантические сети являются богатыми источниками идей моделирования данных, чрезвычайно полезных в плане решения проблемы представления сложных ситуаций. Они могут быть использованы независимо или совместно с идеями, положенными в основу других моделей данных. Их интересной особенностью является то, что расстояние, измеренное на сети (семантическое расстояние или метрика), играет важную роль, определяя близость взаимосвязанных понятий. При этом предусмотрена возможность в явной форме подчеркнуть, что семантическое расстояние велико. Как показано на рис. 5, СПЕЦИАЛЬНОСТЬ соотносится с личностью ПРЕПОДАВАТЕЛЬ, и в то же время ПРЕПОДАВАТЕЛЮ присущ РОСТ. Взаимосвязь личности со специальностью очевидна, однако из этого не обязательно следует взаимосвязь СПЕЦИАЛЬНОСТИ и РОСТА.

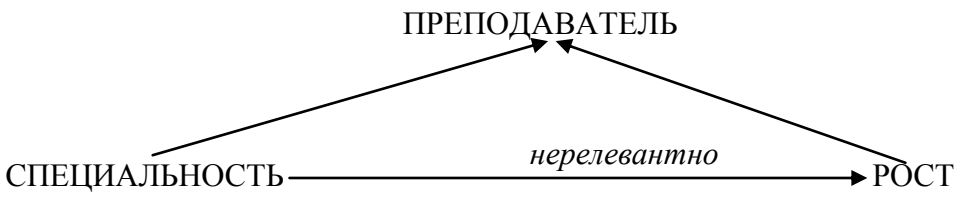

Рис. 5. Соотношение понятий в семантической сети.

Следует сказать, что моделям данных типа семантической сети при всем присущем им богатстве возможностей при моделировании сложных ситуаций присуща усложненность и некоторая неэкономичность в концептуальном плане.

Лекция № 10

Содержание лекции

Технология моделирования информационных систем

Методы моделирования систем

Понятие *модели* является ключевым в общей теории систем. Моделирование как мощный — а часто и единственный — метод исследования подразумевает замещение реального объекта другим — материальным или идеальным.

Важнейшими требованиями к любой модели являются ее *адекватность* изучаемому объекту в рамках конкретной задачи и реализуемость имеющимися средствами.

В *теории эффективности и информатике моделью объекта (системы, операции) называется материальная или идеальная (мысленно представимая) система, создаваемая и/или используемая при решении конкретной задачи с целью получения новых знаний об объекте-оригинале, адекватная ему с точки зрения изучаемых свойств и более простая, чем оригинал, в остальных аспекта*.

Классификация основных методов моделирования (и соответствующих им моделей) представлена на рис. 1.

При исследовании информационных систем (ИС) находят применение все методы моделирования, однако основное внимание будем уделять семиотическим (знаковым) методам.

Напомним, что *семиотикой* (от греч. *semeion* — знак, признак) называют науку об общих свойствах знаковых систем, т.е. систем конкретных или абстрактных объектов (знаков), с каждым из которых сопоставлено некоторое значение. Примерами таких систем являются любые языки (естественные или искусственные, например, языки описания данных или моделирования), системы сигнализации в обществе и животном мире и т.п.

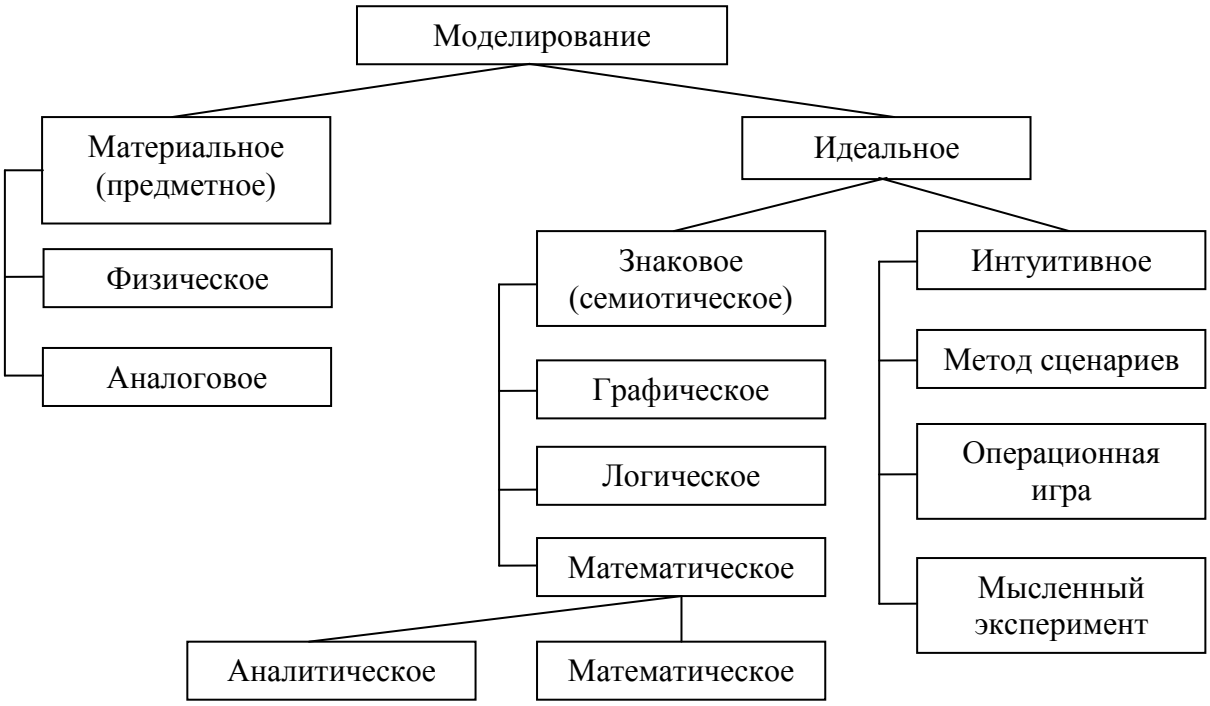

Рис. 1. Классификация методов моделирования.

Семиотика включает *три раздела*:

- *синтактика*;
- *семантика*;
- *прагматика*.

Синтактика исследует синтаксис знаковых систем безотносительно к каким-либо интерпретациям и проблемам, связанным с восприятием знаковых систем как средств общения и сообщения.

Семантика изучает интерпретацию высказываний знаковой системы и с точки зрения моделирования объектов занимает в семиотике главное место.

Прагматика исследует отношение использующего знаковую систему к самой знаковой системе, в частности — восприятие осмысленных выражений знаковой системы.

Из множества семиотических моделей в силу наибольшего распространения, особенно в условиях информатизации современного общества и внедрения формальных методов во все сферы человеческой деятельности, *выделим математические, которые отображают реальные системы с помощью математических символов.* При этом, учитывая то обстоятельство, что мы рассматриваем методы моделирования применительно к исследованию систем в различных операциях, будем использовать общеизвестную методологию системного анализа, теории эффективности и принятия решений.

Математическая модель системы

Задача построения математической модели ИС может быть поставлена следующим образом: для конкретной *цели* моделируемой операции с учетом имеющихся ресурсов построить *операторы моделирования исхода операции и оценки показателя ее эффективности.* Формальная запись этой задачи имеет вид:

$$\langle A_0, \Theta_P, \mathrm{H}, \Psi \rangle,$$

где A_0 – цель; Θ_P – имеющиеся ресурсы; H – оператор моделирования исхода операции; Ψ – оператор оценки показателя эффективности.

Перед рассмотрением каждого из названных операторов приведем два важных определения.

Оператором в математике называют закон (правило), согласно которому каждому элементу x множества X *ставится в соответствие определенный элемент y множества* Y. При этом множества X и Y могут иметь самую различную природу (если они представляют, например, множества действительных или комплексных чисел, понятие оператор совпадает с понятием функции).

Множество Z *упорядоченных пар* $(x; y)$, *где* $x \in \mathrm{X}$, $y \in \mathrm{Y}$, называется *прямым произведением множеств* X *и* Y и обозначается $\mathrm{X} \times \mathrm{Y}$. Аналогично, множество Z упорядоченных конечных последовательностей $(x_1, x_2, ..., x_n)$, где $x_k \in \mathrm{X}_k$, называется прямым произведением множеств $\mathrm{X}_1, \mathrm{X}_2, ..., \mathrm{X}_N$ и обозначается $\mathrm{Z} = \mathrm{X}_1 \times \mathrm{X}_2 \times ... \times \mathrm{X}_N$.

Оператором моделирования исхода операции называется оператор H, устанавливающий соответствие между *множеством* Λ *учитываемых в модели факторов, множеством* U *возможных стратегий управления системой (операцией) и множеством* Y *значений выходных характеристик модели*

$$\mathrm{H}: \Lambda \times \mathrm{U} \xrightarrow{A_0, \Theta_M, R_S} \mathrm{Y},$$

где Θ_M, R_S — ресурсы на этапе моделирования исходов операции и *учитываемые свойства* моделируемой системы соответственно.

Оператором оценки показателя эффективности системы (операции) называется оператор Ψ, ставящий в соответствие *множеству* Y *значений выходных характеристик* модели *множество* W *значений показателя эффективности* системы

$$\Psi: \mathrm{Y} \xrightarrow{A_0, \Theta_Э, R_S} \mathrm{W},$$

где $\Theta_Э$ — *ресурсы* исследователя *на этапе оценивания эффективности системы.*

Особо отметим, что построение приведенных операторов всегда осуществляется с учетом *главного системного принципа – принципа цели.* Кроме того, важным является влияние объема имеющихся в распоряжении исследователя ресурсов на *вид оператора моделирования исхода* H и *состав множества* U *стратегий управления системой*

(операцией). Чем больше выделенные ресурсы, тем детальнее (подробнее) может быть модель и тем большее число стратегий управления может быть рассмотрено (из теории принятия решений известно, что первоначально множество возможных альтернатив должно включать как можно больше стратегий, иначе можно упустить наилучшую).

В самом общем виде *математической моделью системы (операции) называется множество*

$$M=\langle U,\Lambda,H,Y,\Psi,W\rangle,$$

элементами которого являются рассмотренные выше множества и операторы.

Способы задания оператора Ψ и подходы к выбору показателя эффективности W рассматриваются в теории эффективности; методы формирования множества возможных альтернатив – в теории принятия решений.

Для двух классов задач показатель эффективности в явном виде не вычисляется:

- для *задач* так называемой *прямой оценки,* в которых в качестве показателей эффективности используются значения одной или нескольких выходных характеристик модели;
- для *демонстрационных задач,* в ходе решения которых для изучения поведения системы используются лишь значения ее выходных характеристик и внутренних переменных.

В таких случаях используют термин *"математическое описание системы",* представляемое множеством

$$M'=\langle U,\Lambda,H,Y\rangle.$$

Классификация математических моделей

В качестве основного *классификационного признака* для математических моделей целесообразно использовать *свойства операторов моделирования исхода операции и оценки показателя ее эффективности*.

Оператор моделирования исхода H может быть *функциональным (заданным системой аналитических функций)* или *алгоритмическим (содержать математические, логические и логико-лингвистические операции, не приводимые к последовательности аналитических функций).* Кроме того, он может быть детерминированным (когда каждому элементу множества $U\times\Lambda$ соответствует детерминированное подмножество значений выходных характеристик модели $\bar{Y}\subseteq Y$) или стохастическим (когда каждому значению множества $U\times\Lambda$ соответствует случайное подмножество $\tilde{Y}\subseteq Y$).

Оператор оценивания показателя эффективности Ψ может задавать либо *точечно-точечное преобразование (когда каждой точке множества выходных характеристик* Y *ставится в соответствие единственное значение показателя эффективности* W*),* либо

множественно-точечное преобразование (когда показатель эффективности задается на всем множестве полученных в результате моделирования значений выходных характеристик модели).

В зависимости от свойств названных операторов все математические модели подразделяются на *три основных класса*:

- *аналитические*;
- *статистические*;
- *имитационные*.

Для *аналитических моделей характерна детерминированная функциональная связь между элементами множеств* U, Λ, Y, а значение показателя эффективности W определяется с помощью *точечно-точечного отображения.* Аналитические модели имеют весьма широкое распространение. Они хорошо описывают качественный характер (основные тенденции) поведения исследуемых систем. В силу простоты их реализации на ЭВМ и высокой оперативности получения результатов такие модели часто применяются при решении задач *синтеза* систем, а также при *оптимизации вариантов применения* в различных операциях.

К *статистическим* относят математические модели систем, у которых *связь между элементами множеств* U, Λ, Y *задается функциональным оператором* H, а оператор Ψ является *множественно-точечным отображением, содержащим алгоритмы статистической обработки.* Такие модели применяются в тех случаях, когда результат операции является *случайным,* а конечные функциональные зависимости, связывающие статистические характеристики учитываемых в модели случайных факторов с характеристиками исхода операции, отсутствуют. Причинами случайности исхода операции могут быть случайные внешние воздействия; случайные характеристики внутренних процессов; случайный характер реализации стратегий управления. В статистических моделях *сначала формируется представительная выборка значений выходных характеристик модели, а затем производится ее статистическая обработка* с целью получения значения скалярного или векторного показателя эффективности.

Имитационными называются математические модели систем, у которых *оператор моделирования исхода операции задается алгоритмически.* Когда этот оператор является *стохастическим,* а оператор оценивания эффективности задается *множественно-точечным отображением,* имеем *классическую имитационную модель*. Если оператор H является *детерминированным,* а оператор Ψ задает *точечно-точечное отображение,* можно говорить о *вырожденной имитационной модели,*

На рис. 2 представлена классификация наиболее часто встречающихся математических моделей по рассмотренному признаку.

Важно отметить, что при создании аналитических и статистических моделей широко используются их *гомоморфные свойства (способность одних и тех же математических моделей описывать различные по физической природе процессы и явления)*. Для имитационных моделей в наибольшей степени характерен *изоморфизм процессов и структур, т.е. взаимно-однозначное соответствие элементов структур и процессов реальной системы элементам ее математического описания и, соответственно, модели.*

Вид основных операторов		H			
		функциональный		алгоритмический	
		детермин.	стохастих.	детермин.	стохастих.
Ψ	Множественно-точечное отображение	———	Статистические	———	Имитационные
	Точечно-точечное отображение	Аналитические	———	Имитационные	———

Рис. 2. Основная классификация математических моделей.

Изоморфизм — соответствие (отношение) между объектами, выражающее *тождество их структуры (строения).* Именно таким образом организовано большее число классических имитационных моделей. Названное свойство имитационных моделей проиллюстрировано рис. 3. На рисунке обозначены: S_1 — система-оригинал; S_2 — изоморфное отображение оригинала; S_3 — гомоморфное отображение оригинала.

Имитационные модели являются *наиболее общими математическими моделями.* В силу этого *иногда все модели* называют *имитационными*:

- *аналитические модели,* "имитирующие" только физические законы, на которых основано санкционирование реальной системы, можно рассматривать как *имитационные модели I уровня*;
- *статистические модели,* в которых, кроме того, "имитируются" случайные факторы, можно называть *имитационными моделями II уровня*;
- собственно *имитационные модели,* в которых еще имитируется и функционирование системы во времени, называют *имитационными моделями III уровня*.

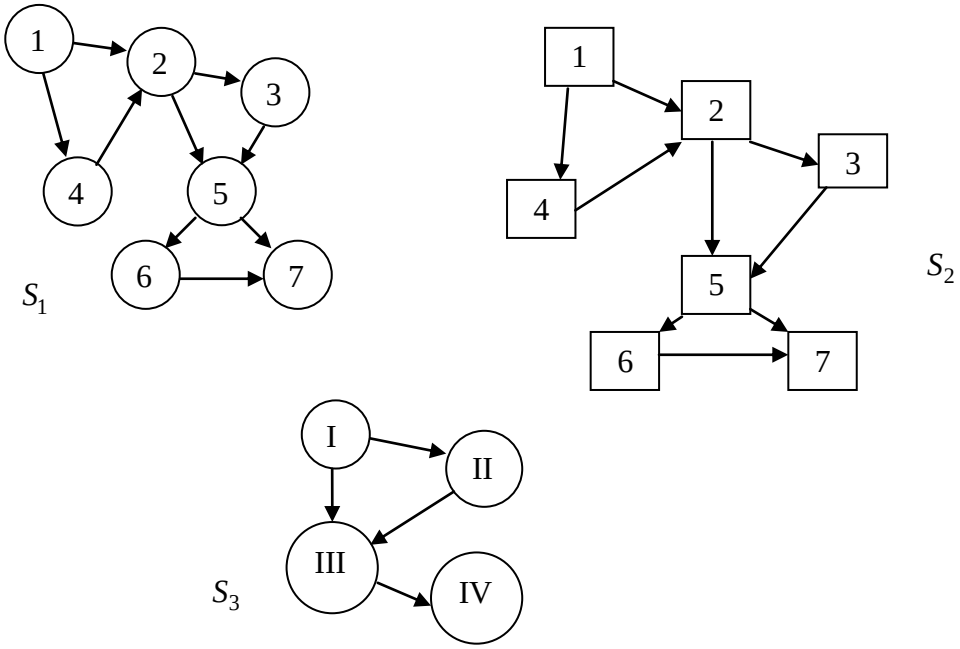

Рис. 3. Пример изоморфного и гомоморфного отображений.

На рис. 4 представлена классификация моделей (прежде всего аналитических и статистических) по зависимости переменных и параметров от времени. Динамические модели, в которых учитывается изменение времени, подразделяются на *стационарные* (в которых от времени зависят только входные и выходные характеристики) и *нестационарные* (в которых от времени могут зависеть либо параметры модели, либо ее структура, либо и то и другое).

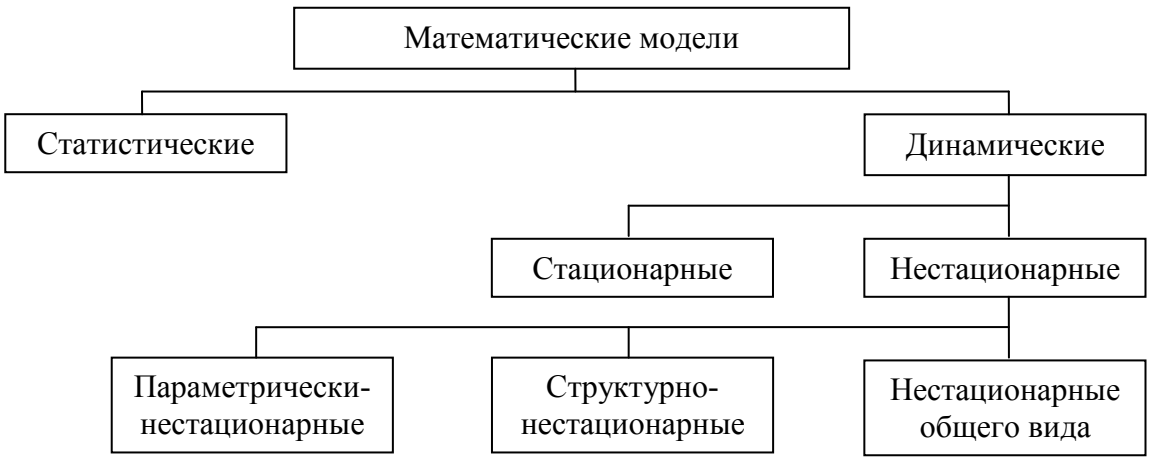

Рис. 4. Классификация математических моделей по зависимости переменных и параметров от времени.

На рис. 5 показана классификация математических моделей еще по трем основаниям: по *характеру изменения переменных*; по *особенностям* используемого *математического аппарата*; по способу учета *проявления случайностей*.

Названия типов (видов) моделей в каждом классе достаточно понятны. Укажем лишь, что в сигнально-стохастических моделях случайными являются только внешние воздействия на систему.

Имитационные модели, как правило, можно отнести к следующим типам:

- по характеру изменения переменных — к *дискретно-непрерывным моделям*;
- по математическому аппарату — к моделям *смешанного* типа;
- по способу учета случайности — к *стохастическим моделям общего* вида.

Рис. 5. Классификация математических моделей.

Лекция № 11

Содержание лекции

Имитационные модели информационных систем

Методологические основы применения метода имитационного моделирования

Приведем классическое вербальное определение имитационного моделирования и проведем его краткий анализ.

По Р. Шеннону (Robert E. Shannon — профессор университета в Хантсвилле, штат Алабама, США) *"имитационное моделирование – есть процесс конструирования на ЭВМ модели сложной реальной системы, функционирующей во времени, и постановки экспериментов на этой модели с целью либо понять поведение системы, либо оценить различные стратегии, обеспечивающие функционирование данной системы"*.

Выделим в этом определении ряд важнейших обстоятельств, учитывая особенности применения метода для исследования информационных систем (ИС).

Во-первых, *имитационное моделирование предполагает два этапа: конструирование модели* на ЭВМ и *проведение экспериментов* с этой моделью. Каждый из этих этапов предусматривает использование собственных методов. Так, на первом этапе весьма важно грамотно провести информационное обследование, разработку всех видов документации и их реализацию. Второй этап должен предполагать использование методов планирования эксперимента с учетом особенностей машинной имитации.

Во-вторых, в полном соответствии с системными принципами четко *выделены две возможные цели имитационных экспериментов*:

- либо *понять поведение исследуемой системы* (о которой по каким-либо причинам было "мало" информации) — потребность в этом часто возникает, например, при создании принципиально новых образцов продукции;
- либо *оценить возможные стратегии управления системой,* что также очень характерно для решения широкого круга экономико-прикладных задач.

В-третьих, с помощью имитационного моделирования *исследуют сложные системы.* Понятие "сложность" является субъективным и по сути выражает отношение исследователя к объекту моделирования. Укажем *пять признаков "сложности" системы,* по которым можно судить о ее принадлежности к такому классу систем:

- наличие *большого* количества взаимосвязанных и взаимодействующих элементов;
- *сложность* функции (функций), выполняемой системой;
- *возможность* разбиения системы на подсистемы (декомпозиции);
- наличие управления (часто имеющего иерархическую структуру), *разветвленной* информационной сети и *интенсивных* потоков информации;
- наличие взаимодействия с внешней средой и функционирование в условиях воздействия случайных (неопределенных) факторов.

Очевидно, что некоторые приведенные признаки сами предполагают субъективные суждения. Вместе с тем становится понятным, почему значительное число ИС относят к сложным системам и, следовательно, применяют метод имитационного моделирования.

В-четвертых, методом имитационного моделирования *исследуют системы, функционирующие во времени,* что определяет необходимость создания и использования специальных *методов (механизмов) управления системным временем*.

Наконец, в-пятых, в определении прямо указывается на *необходимость использования ЭВМ для реализации имитационных моделей,* т.е. проведения машинного эксперимента (машинной имитации), причем в подавляющем большинстве случаев применяются цифровые машины.

Даже столь краткий анализ позволяет сформулировать вывод о целесообразности (а, следовательно, и необходимости) использования метода имитационного моделирования для исследования сложных человекомашинных (эргатических) информационных систем. Особо выделим наиболее характерные обстоятельства применения имитационных моделей:

- если идет *процесс познания объекта моделирования*;
- если аналитические методы исследования имеются, но составляющие их *математические процедуры очень сложны и трудоемки*;
- если *необходимо осуществить наблюдение за поведением компонент системы* в течение определенного времени;
- если *необходимо контролировать протекание процессов в системе путем замедления или ускорения явлений* в ходе имитации;
- если *особое значение имеет последовательность событий в проектируемых системах* и модель используется для предсказания так называемых "узких" мест;
- при *подготовке специалистов* для приобретения необходимых навыков в эксплуатации новой техники;
- и, конечно, если *имитационное моделирование оказывается единственным способом исследований* из-за невозможности проведения реальных экспериментов.

До настоящего момента особое внимание в толковании термина "имитационное моделирование системы" было уделено первому слову. Однако не следует упускать из вида, что создание любой (в том числе и имитационной) модели предполагает, что она будет отражать лишь наиболее существенные с точки зрения конкретной решаемой задачи свойства объекта-оригинала.

Английский аналог этого термина — *systems simulation* — при дословном переводе непосредственно указывает на необходимость воспроизводства (симуляции) лишь основных черт реального явления (сравним с термином "симуляция симптомов болезни" из медицинской практики). Важно отметить еще один аспект: создание любой (в том числе и имитационной модели) есть процесс творческий (не случайно Р. Шеннон назвал свою книгу "Имитационное моделирование систем — искусство и наука"), и, вообще, каждый автор имеет право на собственную версию модели реальной системы. Однако за достаточно длительное время применения метода накоплены определенный опыт и признанные разумными рекомендации, которыми целесообразно руководствоваться при организации имитационных экспериментов.

Укажем ряд *основных достоинств и недостатков* метода имитационного моделирования. Основные *достоинства*:

- имитационная модель позволяет, в принципе, описать моделируемый процесс с *большей адекватностью,* чем другие;
- имитационная модель обладает *гибкостью варьирования* структуры, *алгоритмов и параметров* системы;
- применение ЭВМ *существенно сокращает продолжительность испытаний* по сравнению с натурным экспериментом (если он возможен), а также их стоимость.

Основные *недостатки*:

- *решение,* полученное на имитационной модели, *всегда носит частный характер,* так как оно соответствует фиксированным элементам структуры, алгоритмам поведения и значениям параметров системы;
- *большие трудозатраты* на создание модели и проведение экспериментов, а также обработку их результатов;
- если использование системы предполагает участие людей при проведении машинного эксперимента, на результаты может оказать влияние так называемых *хауторнский эффект* (заключающийся в том, что люди, зная (чувствуя), что за ними наблюдают, могут изменить свое обычное поведение).

Итак, само использование термина "имитационное моделирование" предполагает работу с такими математическими моделями, с помощью которых *результат исследуемой операции нельзя заранее вычислить или предсказать, поэтому необходим эксперимент (имитация) на модели при заданных исходных данных.* В свою очередь, *сущность*

машинной имитации заключается в реализации численного метода проведения на ЭВМ экспериментов с математическими моделями, описывающими поведение сложной системы в течение заданного или формируемого периода времени.

Каждая имитационная модель представляет собой комбинацию шести основных составляющих:

- *компонентов*;
- *переменных*;
- *параметров*;
- *функциональных зависимостей*;
- *ограничений*;
- *целевых функций*.

Под *компонентами* понимают *составные части,* которые при соответствующем объединении образуют систему. Компоненты называют также *элементами системы или* ее *подсистемами.* Например, в модели рынка ценных бумаг компонентами могут выступать отделы коммерческого банка (кредитный, операционный и т.д.), ценные бумаги и их виды, доходы, котировка и т.п.

Параметры — это *величины,* которые исследователь (пользователь модели) *может выбирать произвольно, т.е. управлять ими*.

В отличие от них *переменные* могут принимать только значения, определяемые *видом данной функции.* Так, в выражении для плотности вероятности нормально распределенной случайной величины *X*:

$$f(X) = \frac{1}{\sqrt{2\pi\sigma_X}} \cdot e^{-\frac{(x-m_X)^2}{2\sigma_X^2}},$$

где x – переменная; m_X, σ_X – параметры (математическое ожидание и стандартное отклонение соответственно); π, e — константы.

Различают *экзогенные* (являющиеся для модели *входными и* порождаемые *вне системы) и эндогенные* (возникающие *в системе* в результате воздействия внутренних причин) переменные. Эндогенные переменные иногда называют *переменными состояния.*

Функциональные зависимости описывают *поведение параметров и переменных* в пределах компонента или же выражают *соотношения между компонентами системы.* Эти соотношения могут быть либо *детерминированными,* либо *стохастическими.*

Ограничения — устанавливаемые *пределы изменения значений переменных или ограничивающие условия* их изменения. Они могут вводиться разработчиком (и тогда их называют *искусственными) или* определяться самой системой вследствие присущих ей свойств (так называемые *естественные* ограничения).

Целевая функция предназначена для измерения степени достижения системой желаемой (требуемой) цели и вынесения оценочного суждения по результатам моделирования. Эту функцию также называют функцией критерия. По сути, весь машинный эксперимент с имитационной моделью заключается в поиске таких стратегий управления системой, которые удовлетворяли бы одной из трех концепций ее рационального поведения: *оптимизации, пригодности или адаптивизации*. Если показатель эффективности системы является скалярным, проблем с формированием критерия не возникает и, как правило, решается *оптимизационная задача* — по иска стратегии, соответствующей максимуму или минимуму показателя. Сложнее дело обстоит, если приходится использовать векторный показатель. В этом случае для вынесения оценочного суждения используются методы принятия решений по векторному показателю в условиях определенности (когда в модели учитываются только детерминированные факторы) или неопределенности (в противном случае).

При реализации имитационной модели, как правило, рассматриваются *не все* реально осуществляемые функциональные действия (ФД) системы, а только те из них, которые являются *наиболее существенными для исследуемой операции.* Кроме того, *реальные ФД аппроксимируются упрощенными действиями* ФД′ причем степень этих упрощений определяется уровнем детализации учитываемых в модели факторов. Названные обстоятельства порождают *ошибки имитации* процесса функционирования реальной системы, что, в свою очередь, обусловливает *адекватность модели* объекту-оригиналу и *достоверность* получаемых в ходе моделирования *результатов.*

На рис. 1 схематично представлен пример выполнения некоторых ФД в i-м компоненте реальной системы и ФД′ в i-м компоненте ее модели.

В i-м компоненте реальной системы последовательно выполняются $ФД_{i_1}$, $ФД_{i_2}$, $ФД_{i_3}$,… за времена τ_{i_1}, τ_{i_2}, τ_{i_3},…, соответственно. На рисунке эти действия условно изображены пунктирными ("непрямыми") стрелками. В результате ФД наступают соответствующие события: C_{i_1}, C_{i_2}, C_{i_3},… В модели последовательность имитации иная: выполняется $ФД'_{i_1}$ при *неизменном времени,* наступает модельное событие *a, после чего время сдвигается на величину* τ_{i_1}, инициируя наступление события C_{i_1} и т.д. Иными словами, модельной реализации упрощенных ФД (ФД′) соответствует ломаная $\left(0, a, C_{i_1}, b, C_{i_2}, c, C_{i_3}, ...\right)$. Отметим, что в принципе возможен и другой порядок моделирования: сначала сдвигать время, а затем инициировать наступление соответствующего события.

Рис. 1. Схема моделирования функциональных действий

в i-м компоненте системы.

Очевидно, что в реальной системе в различных ее компонентах могут *одновременно (параллельно) производиться функциональные действия* и, соответственно, наступать события. В большинстве же современных ЭВМ в каждый из моментов времени можно отрабатывать *лишь один алгоритм какого-либо ФД.* Возникает вопрос: каким образом учесть параллельность протекания процессов в реальной системе без потери существенной информации о ней?

Для обеспечения имитации наступления параллельных событий в реальной системе вводят специальную глобальную переменную t_0, *которую называют модельным (системным) временем.* Именно с *помощью этой переменной организуется синхронизация наступления всех событий в модели ИС и выполнение алгоритмов функционирования ее компонент. Принцип такой организации моделирования называется принципом квазипараллелизма*.

Таким образом, при реализации имитационных моделей используют три представления времени:

- t_P — *реальное* время системы;
- t_0 — *модельное (системное)* время;
- t_M — *машинное* время имитации.

Лекция № 12

Содержание лекции

Имитационные модели информационных систем

Классификация имитационных моделей

Имитационные модели принято классифицировать по четырем наиболее распространенным признакам:

- *типу* используемой ЭВМ;
- *способу взаимодействия* с пользователем;
- *способу управления системным временем (механизму* системного времени);
- *способу организации квазипараллелизма (схеме формализации* моделируемой системы).

Первые два признака позволяют разделить имитационные модели на совершенно понятные (очевидные) классы.

По *типу* используемой ЭВМ различают *аналоговые, цифровые* и *гибридные имитационные модели.* В дальнейшем будем рассматривать *только цифровые модели*.

По *способу взаимодействия с* пользователем имитационные модели могут быть *автоматическими* (не требующими вмешательства исследователя после определения режима моделирования и задания исходных данных) и *интерактивными* (предусматривающими диалог с пользователем в том или ином режиме в соответствии со сценарием моделирования). Отметим, что моделирование сложных систем, относящихся, как уже отмечалось, к классу *эргатических* систем, как правило, требует применения диалоговых моделей.

Различают *два механизма системного времени*:

- задание времени с помощью *постоянных временных интервалов* (шагов);
- задание времени с помощью *переменных временных интервалов* (моделирование по особым состояниям).

При реализации первого механизма системное время сдвигается на один и тот же интервал (шаг моделирования) независимо от того, какие события должны наступать в системе. При этом наступление всех событий, имевших место на очередном шаге, относят к его окончанию. На рис. 1, а) показана схема реализации механизма системного времени

с постоянным шагом. Так, для этого механизма считают, что событие A_1 наступило в момент окончания первого шага; событие A_2 — в момент окончания второго шага; события A_3, A_4, A_5 — в момент окончания четвертого шага (эти моменты показаны стрелками) и т.д.

При моделировании *по особым состояниям* системное время каждый раз изменяется на величину, соответствующую интервалу времени до планируемого момента наступления следующего события, т.е. события обрабатываются поочередно – каждое "в свое время". Если в реальной системе какие-либо события наступают одновременно, это фиксируется в модели. Для реализации этого механизма требуется специальная процедура, в которой отслеживаются времена наступления всех событий и из них выделяется ближайшее по времени. Такую процедуру называют *календарем событий*. На рис. 1, б) стрелками обозначены моменты изменения системного времени.

Рис. 1. Схемы реализации механизмов системного времени:
а) с постоянным шагом; б) с переменным шагом.

Существует не столь распространенная разновидность механизма моделирования по особым состояниям, предусматривающая возможность изменения порядка обработки событий, так называемый *механизм моделирования с реверсированием (обращением) шага* по времени. Согласно этому механизму все события в системе разбиваются на два класса: *фазовые* и *простые.* К первым относят события, порядок моделирования которых нельзя изменять во избежание нарушения причинно-следственных связей в моделируемой системе. Остальные события относят к простым. Таким образом, сначала моделируют очередное фазовое событие, а затем — все простые события до этого фазового, причем в произвольном порядке.

На рис. 2 приведены перечисленные способы управления системным временем.

Очевидно, что механизм системного времени с постоянным шагом легко реализуем: достаточно менять временную координату на фиксированный шаг и проверять, какие события уже наступили.

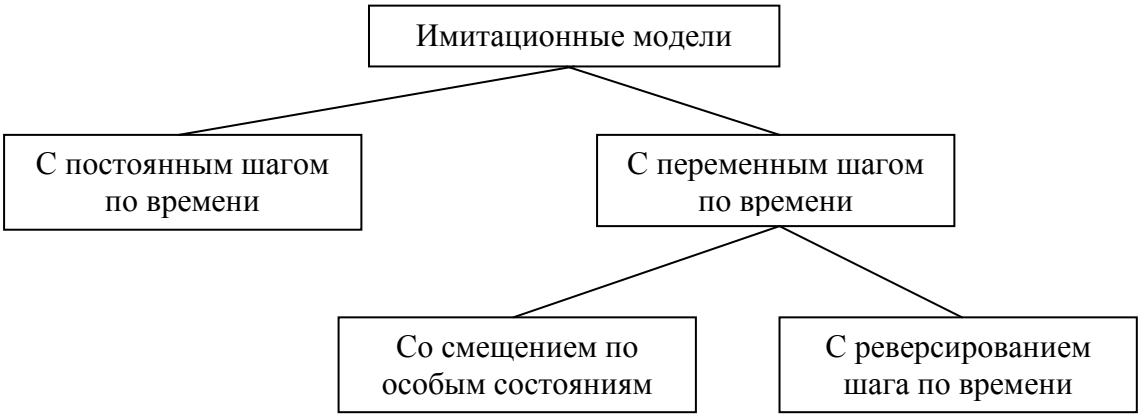

Рис. 2. Механизмы управления системным временем.

Метод фиксированного шага *целесообразно* применять в следующих случаях:

- события в системе появляются *регулярно*;
- число событий *велико*;
- все события являются для исследователя *существенными* (или заранее неизвестно, какие из них существенны).

Вопрос о том, каким же механизмом системного времени воспользоваться, решается путем анализа достоинств и недостатков каждого применительно к конкретной модели и требует от разработчика высокой квалификации. В некоторых моделях используют комбинированные механизмы системного времени в целях исключения недостатков.

Важнейшим классификационным признаком имитационных моделей является схема *формализации моделируемой системы* (способ организации квазипараллелизма).

Наибольшее распространение получили пять способов:

- просмотр *активностей*;
- *составление расписания событий*;
- *управление обслуживанием транзактов*;
- *управление агрегатами*;
- *синхронизация процессов*.

Характеристика этих способов требует введения ряда понятий.

Основными составными частями модели ИС являются *объекты,* которые представляют компоненты реальной системы. Для задания свойств объектов используются *атрибуты (параметры).* Совокупность объектов с одним и тем же набором атрибутов называют *классом объектов.* Все объекты подразделяют на *активные* (представляющие в модели те объекты реальной системы, которые способны функционировать самостоятельно и выполнять некоторые действия над другими

объектами) и *пассивные* (представляющие реальные объекты, самостоятельно в рамках данной модели не функционирующие).

Работа *(активность)* представляется в модели *набором операций, выполняемых в течение некоторого времени и приводящих к изменению состояний объектов системы. В рамках конкретной модели любая работа рассматривается как единый дискретный шаг (возможно, состоящий из других работ). Каждая работа характеризуется временем выполнения и потребляемыми ресурсами.*

Событие представляет собой *мгновенное изменение состояния некоторого объекта системы (т.е. изменение значений его атрибутов).* Окончание любой активности в системе является событием, так как приводит к изменению состояния объекта (объектов), а также может служить инициатором другой работы в системе.

Под процессом понимают логически связанный набор активностей, относящихся к одному объекту. Выполнение таких активностей называют *фазой процесса.* Различие между понятиями "активность" и "процесс" полностью определяется степенью детализации модели. Например, смена позиций мобильным объектом в одних моделях может рассматриваться как сложный процесс, а в других — как работа по изменению за некоторое время номера позиции. *Процессы, включающие одни и те же типы работ и событий, относят к одному классу.* Таким образом, моделируемую систему можно представить соответствующим числом классов процессов. Между двумя последовательными фазами (работами) некоторого процесса может иметь место любое число фаз других процессов, а их чередование в модели, собственно, и выражает суть квазипараллелизма.

В ряде случаев функциональные действия (ФД) *компонент (объектов) реальной системы одинаковы, а общее их число ограничено.* Каждое ФД можно описать *простейшими работами,* которые приводят *лишь к изменению значений временных координат* компонент системы. Взаимодействие такого рода активностей аналогично функционированию системы массового обслуживания. Однотипные активности объединяются и называются *приборами массового обслуживания.* Инициаторами появления событий в такой модели становятся *заявки (транзакты) на обслуживание* этими приборами.

В некоторых реальных системах ФД *отдельных компонент тесно взаимодействуют друг с другом. Компоненты обмениваются между собой сигналами,* причем выходной сигнал одной компоненты может поступать на вход другой, а сами ФД можно *в явном виде описать математическими зависимостями.* Если появление выходного сигнала таким образом определяется соответствующим набором "входов", можно реализовать так называемый *модульный принцип построения модели.* Каждый из

модулей строится по *стандартной (унифицированной, типовой) структуре* и называется *агрегатом.*

Рассмотрим характеристики *способов организации квазипараллелизма.*

Способ просмотра активностей применяется при следующих условиях:

- все ФД компонент реальной системы различны, причем для выполнения каждого из них требуется выполнение некоторых (своих) условий;
- условия выполнимости известны исследователю заранее и могут быть заданы алгоритмически;
- в результате ФД в системе наступают различные события;
- связи между ФД отсутствуют и они осуществляются независимо друг от друга.

В данном контексте имитационная модель состоит из двух частей:

- множества активностей (работ);
- набора процедур проверки выполнимости условий инициализации активностей, т.е. возможности передачи управления на реализацию алгоритма этой активности.

Проверка *выполнимости условия инициализации работы* основана либо *на анализе значений параметров и/или переменных модели,* либо *вычислении моментов времени, когда должно осуществляться данное ФД.*

После *выполнения каждой активности производится модификация системного времени для данного компонента и управление передается в специальный управляющий модуль,* что и составляет суть имитации для этого способа организации квазипараллелизма.

Составление расписания событий применяется в тех случаях, когда реальные процессы характеризуются рядом достаточно строгих ограничений:

- различные компоненты выполняют *одни и те же ФД*;
- начало выполнения этих ФД *определяется одними и теми же условиями,* причем они *известны* исследователю и заданы *алгоритмически*;
- в результате ФД происходят *одинаковые события независимо друг от друга*;
- *связи* между ФД *отсутствуют,* а каждое ФД выполняется *независимо*.

В таких условиях имитационная модель по сути состоит из *двух процедур*:

- *проверки выполнимости* событий;
- обслуживания *(обработки)* событий.

Выполнение этих процедур *синхронизируется в модельном времени* так называемым *списковым механизмом планирования.* Процедура проверки выполнимости событий схожа с ранее рассмотренными для просмотра активностей (напомним, что окончание любой работы является событием и может инициализировать другую

активность) с учетом того, что при выполнении условия происходит *не инициализация работы, а обслуживание (розыгрыш) события с последующим изменением системного времени* для данного компонента. Корректировка системного времени осуществляется календарем событий.

Условия применимости *транзактного способа организации квазипараллелизма* были приведены при определении понятия "транзакт". Связь между приборами массового обслуживания устанавливается с помощью *системы очередей, выбранных способов генерации, обслуживания и извлечения транзактов.* Так организуется *появление* транзактов, *управление их движением, нахождение в очереди, задержки в обслуживании, уход транзакта из системы и т.п. Событием в* такой имитационной модели является *момент инициализации любого транзакта.* Типовыми структурными элементами модели являются *источники* транзактов; их поглотители; блоки, имитирующие *обслуживание* заявок; *управляющий модуль.* Имитация функционирования реальной системы производится путем *выявления очередной (ближайшей по времени) заявки,* ее *обслуживания, обработки итогов обслуживания* (появления нового транзакта; поглощения заявки; изменения возможного времени поступления следующего транзакта и т.п.), *изменения системного времени до момента наступления следующего события.*

В случае построения имитационной модели с *агрегатным способом организации квазипараллелизма* особое внимание следует уделять *оператору перехода системы из одного состояния в другое. Имитация* производится *за счет передачи управления от агрегата к агрегату при выполнении определенных условий, формирования различных сигналов и их доставки адресату, отработки внешних сигналов, изменения состояния агрегата* и т.п. При этом в *управляющем модуле осуществляется временная синхронизация состояний всех агрегатов.* Отметим, что выделение такого способа реализации квазипараллелизма является достаточно условным, так как квазипараллельная работа агрегатов системы может быть организована другими способами — активностями, планированием событий, взаимодействием транзактов, процессами. Иными словами, агрегатный способ прежде всего ориентирован на использование *типовых математических схем (типовых агрегатов) для описания компонент системы и организации их взаимодействия одним из перечисленных способов.*

Процессный способ организации квазипараллелизма применяется в случаях:

- *все ФД* компонент реальной системы *различны*;
- *условия инициализации ФД* также *различны*;
- в любой момент времени в компоненте может выполняться *только одно ФД*;
- *последовательность ФД* в каждом компоненте *определена*.

Принято считать, что процессный подход объединяет лучшие черты других способов: *краткость описания активностей и эффективность событийного*

представления имитации. Процессным способом можно организовать имитацию ИС любой сложности, но такой способ особенно эффективен в тех случаях, когда требуется высокий уровень детализации выполнения ФД, а сама имитационная модель используется для поиска "узких" мест в работе системы. *При таком подходе особо важно соблюдение сходства структуры модели и объекта исследования.* Имитационная модель представляет собой набор описаний процессов, каждое из которых посвящено одному классу процессов, а также информационных и управляющих связей между компонентами модели. *Каждой компоненте объекта моделирования соответствует свой процесс. Переход от выполнения одной активности к другой активности того же процесса* считают изменением его состояния и называют *активизацией процесса. Проверка выполнимости условий активизации* процесса и появление событий *осуществляется самим процессом.* Процессный способ широко применяется в задачах моделирования проектируемых систем. Он позволяет реализовать *многоуровневое модульное моделирование,* предусматривающее внесение в модель частичных изменений по результатам исследований.

На рис. 3 представлена классификация способов организации квазипараллелизма.

Отметим, что в настоящее время для реализации всех перечисленных схем формализации моделируемой системы созданы *специализированные программные средства, ориентированные на данный способ организации квазипараллелизма,* что, с одной стороны, *облегчает программную реализацию модели,* но, с другой стороны, *повышает ответственность исследователя* за правильность выбора соответствующей схемы.

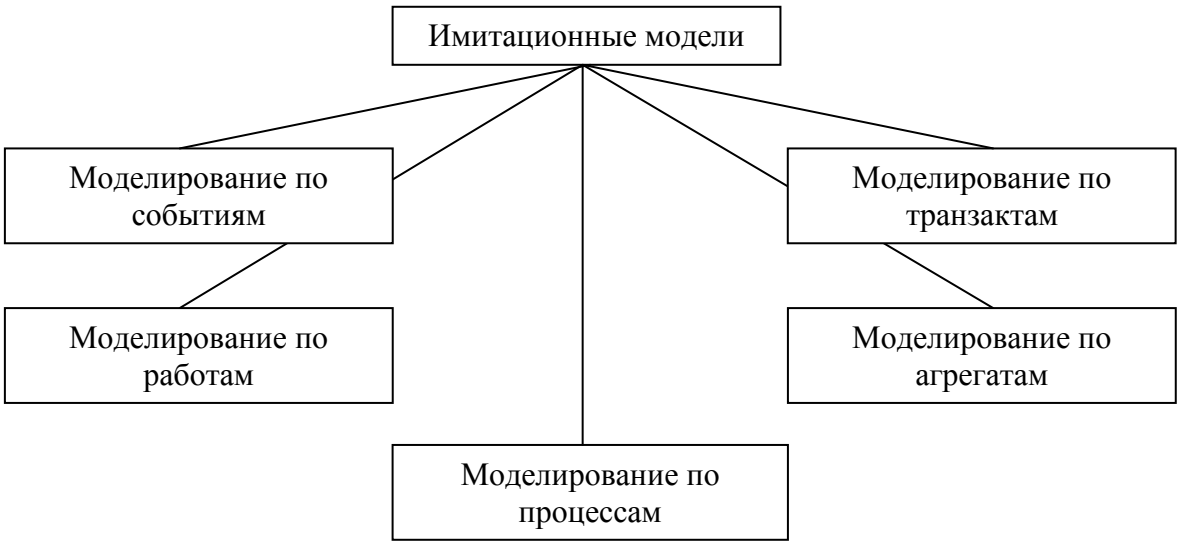

Рис. 3. Классификация имитационных моделей по способу организации квазипараллелизма.

Структура типовой имитационной модели с календарем событий

Составление расписания событий как способ организации квазипараллелизма получило широкое распространение, прежде всего в силу простоты и наглядности реализации.

На рис. 4 представлена структура типовой имитационной модели с календарем событий.

Такая имитационная модель состоит из трех частей:

- *управляющей*;
- *функциональной,* состоящей из функциональных модулей (ФМ);
- информационной, включающей базу (базы) данных (БД).

В свою очередь, *управляющая* часть содержит:

- *блок управления (БУ) моделированием*;
- *блок диалога*;
- *блок обработки результатов моделирования*;
- *календарь событий*.

Рис. 4. Структура типовой имитационной модели с календарем событий.

Блок управления предназначен для *реализации принятого плана имитационного эксперимента.* В соответствии с назначением в его состав обычно включают *управляющий модуль (УМ),* определяющий основные временные установки – моменты начала, остановки, продолжения, окончания моделирования, а также моменты изменения режимов моделирования, и *модуль реализации плана эксперимента,* устанавливающий для каждого прогона модели необходимые значения (уровни) управляемых факторов.

Блок диалога предназначен *для обеспечения комфортной работы пользователя с интерактивной моделью* (в автоматических моделях этого блока нет). Отметим, что кроме понятных процедур ввода вывода информации в требуемых форматах различным потребителям, во многих ("больших") имитационных моделях блок диалога включает *систему интерактивной многоуровневой помощи пользователю.*

В блоке обработки результатов моделирования осуществляется *обмен информацией с базой данных и реализуются процедуры расчета показателя эффективности* (прежде всего — за счет статистической обработки результатов моделируемой операции). Если отсутствует блок диалога, на блок обработки возлагаются функции выдачи результатов моделирования на внешние устройства.

Календарь событий является важнейшим элементом имитационной модели, предназначенным для управления процессом появления событий в системе с целью обеспечения необходимой причинно-следственной связи между ними.

Календарем событий решаются следующие *основные задачи*:

- *ранжирование по времени плановых событий,* т.е. составление упорядоченной временной последовательности плановых событий с учетом вида возможного события и модуля, в котором оно может наступить (для отработки этой задачи в календаре содержится важнейший элемент — *каталог плановых событий,* (рис. 5);
- *вызов необходимых функциональных модулей* в моменты наступления соответствующих событий;
- *получение информационных выходных сигналов* от всех функциональных модулей, *их хранение и передача* в нужные моменты времени адресатам в соответствии с *оператором сопряжения* модели (эта задача решается с помощью специального программного средства — *цепи сигналов* и ее основного элемента — *таблицы сигналов* (рис. 6).

Перед началом моделирования в первую строку каталога плановых событий (см. рис. 5) заносится *время инициализации первого прогона модели,* а в последнюю — *время его окончания,* после чего управление передается на тот ФМ, в котором может наступить ближайшее к начальному по времени событие (если на каждом шаге моделирования проводить *ранжирование событий по времени,* соответствующая этому событию строка каталога будет *первой,* поскольку для всех уже наступивших (отработанных, обслуженных) событий устанавливается и записывается в третий столбец каталога фиктивное время, *заведомо превышающее время окончания моделирования)*.

Наименование модуля	Вид событий	Планируемое время наступления
УМ		
ФМ1		
ФМ2		
. . .		
ФМN		

Рис. 5. Каталог плановых событий.

№ п/п	Адресат	Содержание сигнала	Признак передачи

1			
2			
. . .			
M			

Рис. 6. Таблица сигналов.

Таким образом, если в результате работы очередного ФМ через таблицу сигналов появляется информация о возможном времени наступления в этом или любом другом модуле какого-либо события, это время, а также вид события и модуль, в котором оно может произойти, *заносятся в каталог плановых событий,* после чего осуществляется *новое ранжирование* событий по времени. Затем управление передается ФМ (или УМ), информация о котором находится в первой строке каталога и т.д. до тех пор, пока в первой строке не окажется событие, соответствующее *окончанию моделирования.*

Подобным же образом организуется работа и *таблицы сигналов с* учетом того, что в ней содержится информация не о событиях как таковых, а о *сигналах различных типов.* Так, если в результате работы очередного ФМ возникла необходимость передать какую-либо информацию, соответствующий сигнал (сигналы) помещается в очередную строку таблицы сигналов, после чего осуществляется их передача адресатам. После получения адресатом сигнала в четвертый столбец таблицы заносится *установленный признак,* и данный сигнал считается отработанным. Передача сигналов продолжается до тех пор, пока четвертый столбец таблицы не будет заполнен этим признаком для всех сигналов. Затем управление передается календарю событий, от него – очередному ФМ и т.д.

Функциональная часть имитационной модели состоит из функциональных модулей, являющихся *основными ее элементами.* Именно в них описываются и реализуются все процессы в моделируемой системе. Обычно *один ФМ описывает либо отдельный процесс в системе, либо ее отдельный элемент (подсистему)* — в зависимости от выбранной схемы моделирования. Обобщенная структурная схема работы ФМ представлена на рис. 7.

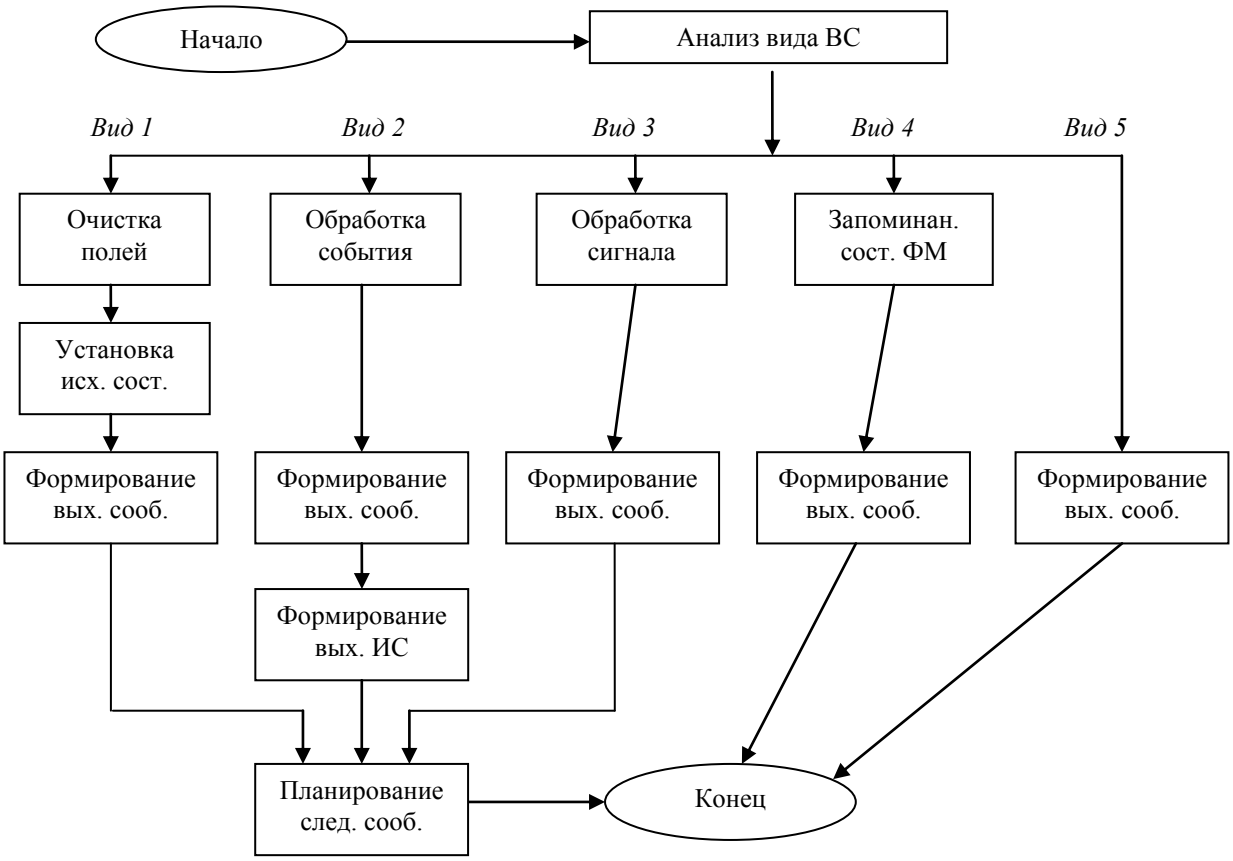

Рис. 7. Обобщенная структурная схема работы типового ФМ.

На рисунке обозначены: ВС — входной сигнал; ИС — информационный сигнал.

В ФМ могут поступать *пять видов входных сигналов*:

- *стартовый* сигнал (сигнал о начале моделирования);
- сигнал о *наступлении планового события*;
- *информационный* сигнал;
- сигнал о *прерывании моделирования*;
- сигнал *об окончании моделирования*.

Отметим, что, какой бы сигнал ни поступил на вход ФМ, *обязательно формируется выходное сообщение о* том, что в ФМ данный сигнал отработан, т.е. проведены соответствующие виду входного сигнала действия: подготовка к моделированию (по ВС вида 1); обработка события (по ВС вида 2); обработка информационного сигнала (по ВС вида 3); запоминание состояния ФМ с целью дальнейшего продолжения моделирования с данного "шага" (по ВС вида 4); завершение моделирования в случае выполнения плана имитационного эксперимента (по ВС вида 5).

Важнейшей задачей любого ФМ является планирование следующих событий, т.е. определение их видов и ожидаемых моментов наступления. Для выполнения этой функции в ФМ реализуется специальный *оператор планирования.* Для "больших" моделей остро стоит вопрос о "глубине планирования", т.е. о длительности интервала времени, на

который прогнозируется наступление событий, поскольку для больших интервалов почти наверняка придется осуществлять повторное планирование после прихода очередного информационного сигнала и соответствующего изменения состояния ФМ.

База (базы) данных представляет собой совокупность специальным образом *организованных (структурированных) данных о моделируемой системе (операции),* а также *программных средств* работы с этими данными. Как правило, информация из БД выдается в другие части имитационной модели в *автоматическом режиме* (в этом смысле можно говорить, что потребителями информации из БД являются пользователи-задачи). Наличие БД в имитационной модели *не является обязательным* и полностью определяется *масштабами модели, объемами* необходимой *информации* и *требованиями* по оперативности получения результатов моделирования и их *достоверности.* Если принято решение о включении БД в состав имитационной модели, проектирование БД не имеет каких-либо специфических особенностей и проводится по стандартной методике.

Лекция №13

Содержание лекции

Имитационные модели информационных систем

Технология моделирования случайных факторов

Генерация псевдослучайных чисел (ПСЧ)

Имитационное моделирование ИС, как правило, предполагает *необходимость учета различных случайных факторов* — событий, величин, векторов (систем случайных величин), процессов.

В основе *всех методов и приемов* моделирования названных случайных факторов лежит *использование случайных чисел, равномерно распределенных на интервале* [0;1].

До появления ЭВМ в качестве генераторов случайных чисел применяли *механические устройства* — колесо рулетки, специальные игральные кости и устройства, которые перемешивали фишки с номерами, вытаскиваемые вручную по одной.

По мере роста объемов применения случайных чисел для ускорения их моделирования стали обращаться к помощи *электронных устройств.* Самым известным из таких устройств был *электронный импульсный генератор, управляемый источником шума,* разработанный широко известной фирмой RAND Corporation. Фирмой в 1955 г. была выпущена книга, содержащая *миллион случайных чисел,* сформированных этим генератором, а также *случайные числа в записи на магнитной ленте.* Использовались и другие подобные генераторы — например, основанные на преобразовании естественного случайного шума при *радиоактивном распаде.* Все эти генераторы обладают *двумя недостатками*:

- *невозможно повторно получить одну и ту же последовательность* случайных чисел, что бывает необходимо при экспериментах с имитационной моделью;

- *технически сложно реализовать физические генераторы,* способные длительное время выдавать случайные числа "требуемого качества".

В принципе, можно заранее ввести полученные таким образом случайные числа в память машины и обращаться к ним по мере необходимости, что сопряжено с понятными негативными обстоятельствами — *большим* (причем неоправданным) расходов *ресурсов ЭВМ и затратой времени на обмен данными* между долгосрочной и оперативной памятью.

В силу этого наибольшее распространение получили другие генераторы, позволяющие получать так называемые *псевдослучайные числа* (ПСЧ) с помощью *детерминированных рекуррентных формул.* Псевдослучайными эти числа называют потому, что фактически они, даже пройдя все тесты на случайность и равномерность распределения, остаются полностью детерминированными. Это значит, что если каждый цикл работы генератора начинается с одними и теми же исходными данными, то на выходе получаем одинаковые последовательности чисел. Это свойство генератора обычно называют *воспроизводимостью последовательности* ПСЧ.

Программные генераторы ПСЧ должны удовлетворять *следующим требованиям*:

- ПСЧ должны быть *равномерно распределены* на интервале [0; 1] и *независимы, т.*е. случайные последовательности должны быть *некоррелированы*;
- *цикл генератора* должен иметь *возможно большую длину*;
- последовательность ПСЧ должна быть *воспроизводима*;
- генератор должен быть *быстродействующим*;
- генератор должен занимать *малый объем памяти.*

Первой расчетной процедурой генерации ПСЧ, получившей достаточно широкое распространение, можно считать *метод срединных квадратов,* предложенный фон Нейманом и Метрополисом в 1946 г. Сущность метода заключается в последовательном нахождении квадрата некоторого n-значного числа; выделении из него m средних цифр, образующих новое число, которое и принимается за очередное в последовательности ПСЧ; возведении этого числа в квадрат; выделении из квадрата m средних цифр и т.д. до получения последовательности требуемой длины.

Как следует из описания процедуры метода, он весьма *прост в вычислительном отношении* и, следовательно, *легко реализуем программно.* Однако ему присущ очень *серьезный недостаток — обусловленность статистических свойств генерируемой последовательности выбором ее корня (начального значения),* причем эта обусловленность не является "регулярной", т.е. трудно определить заранее, можно ли использовать полученные данным методом ПСЧ при проведении исследований. Иными словами, *метод срединных квадратов не позволяет по начальному значению оценить качество последовательности ПСЧ, в частности ее период.*

Мультипликативный метод

Основная формула мультипликативного генератора для расчета значения очередного ПСЧ по значению предыдущего имеет вид:

$$X_{i+1} = aX_i \cdot |m|,$$

где *a*, *m* — неотрицательные целые числа (их называют *множитель* и *модуль).*

Как следует из формулы, для генерации последовательности ПСЧ необходимо задать начальное значение (корень) последовательности, множитель и модуль, причем *период (длина) последовательности P зависит от разрядности ЭВМ и выбранного модуля,* а *статистические свойства — от выбранного начального значения и множителя.* Таким образом, следует выбирать перечисленные величины так, чтобы по возможности максимизировать длину последовательности и минимизировать корреляцию между генерируемыми ПСЧ. В специальной литературе приводятся *рекомендации* по выбору значений параметров метода, использование которых обеспечивает (гарантирует) получение определенного количества ПСЧ с требуемыми статистическими свойствами. Так, если для машины с двоичной системой счисления задать $m = 2^b$, $a = 8 \cdot T \pm 3 \cdot V$, где b — число двоичных цифр (бит) в машинном слове; *T* — любое целое положительное число; *V* — любое положительное нечетное число, получим последовательность ПСЧ с периодом, равным $P = 2^{b-2} = \frac{m}{4}$. Заметим, что в принципе возможно за счет другого выбора модуля *m* увеличить длину последовательности до $P = m - 1$, частично пожертвовав скоростью вычислений. Кроме того, важно, что получаемые таким образом ПСЧ оказываются нормированными, т.е. распределенными от 0 до 1. От выбора корня последовательности ее длина не зависит (при равенстве остальных параметров).

Аддитивный метод

Основная формула для генерации ПСЧ по аддитивному методу имеет вид:

$$X_{i+1} = a\left(X_i + X_{i-1}\right) \cdot |m|,$$

где *m* — целое число.

Очевидно, что для инициализации генератора, построенного по этому методу, необходимо помимо модуля *m* задать два исходных члена последовательности. При $X_0 = 0$; $X_1 = 1$ последовательность превращается в ряд Фибоначчи. Рекомендации по выбору модуля совпадают с предыдущим случаем; длину последовательности можно оценить по приближенной формуле

$$P = 2^{b+1} - 2\left(b - 1\right).$$

Смешанный метод

Данный метод несколько расширяет возможности мультипликативного генератора за счет введения так называемого коэффициента сдвига *c*. Формула метода имеет вид:

$$X_{i+1} = (aX_i + c) \cdot |m| .$$

За счет выбора параметров генератора можно обеспечить максимальный период последовательности ПСЧ $P = 2^b$.

Разработано *множество модификаций* перечисленных конгруэнтных методов, обладающих определенными преимуществами при решении конкретных практических задач, а также *рекомендаций* по выбору того или иного метода. Для весьма широкого круга задач *вполне удовлетворительными оказываются типовые генераторы* ПСЧ, разработанные, как правило, на основе смешанного метода и входящие в состав стандартного общего программного обеспечения большинства ЭВМ. Специальным образом генерацию ПСЧ организуют либо для особо *масштабных имитационных исследований,* либо *при повышенных требованиях к точности имитации* реального процесса (объекта).

Подводя итог, подчеркнем, что *разработка конгруэнтных методов зачастую осуществляется на основе эвристического подхода,* основанного на опыте и интуиции исследователя. После модификации известного метода тщательно проверяют, обладают ли генерируемые в соответствии с новой формулой последовательности ПСЧ требуемым статистическими свойствами, и в случае положительного ответа формируют рекомендации по условиям ее применения.

Моделирование случайных событий

В теории вероятностей *реализацию некоторого комплекса условий называют испытанием. Результат испытания,* регистрируемый как факт, называют событием.

Случайным называют событие, которое в результате испытания *может наступить, а может и не наступить* (в отличие от достоверного события, которое при реализации данного комплекса наступает всегда, и *невозможного* события, которое при реализации данного комплекса условий не наступает никогда). *Исчерпывающей характеристикой случайного события является вероятность его наступления.* Примерами случайных событий являются отказы в экономических системах; объемы выпускаемой продукции предприятием каждый день; котировки валют в обменных пунктах; состояние рынка ценных бумаг и биржевого дела и т.п.

Моделирование случайного события заключается в " определении ("розыгрыше") факта его наступления.

Для моделирования случайного события *A*, наступающего в опыте с вероятностью P_A, достаточно *одного случайного (псевдослучайного) числа R, равномерно*

распределенного на интервале [0;1]. В случае попадания ПСЧ R в интервал $[0;P_A]$ событие A считают наступившим в данном опыте; в противном случае — не наступившим в данном опыте. На рис. 1 показаны оба исхода: при ПСЧ R_1 событие следует считать наступившим; при ПСЧ R_2 — событие в данном испытании не наступило. Очевидно, что чем больше вероятность наступления моделируемого события, тем чаще ПСЧ, равномерно распределенные на интервале [0;1], будут попадать в интервал $[0;P_A]$, что и означает факт наступления события в испытании.

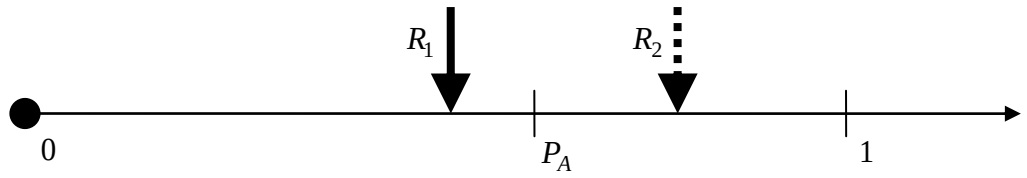

Рис. 1. Моделирование случайных событий.

Для моделирования одного из *полной группы N случайных несовместных событий* $A1, A2, \ldots, AN$, с вероятностями наступления $\{P_{A1}, P_{A2}, ..., P_{AN}\}$ соответственно, также достаточно *одного ПСЧ R.*

Напомним, что для таких случайных событий можно записать:

$$\sum_{i=1}^{N} P_{Ai} = 1.$$

Факт наступления одного из событий группы определяют исходя из условия принадлежности ПСЧ R тому или иному интервалу, на который разбивают интервал [0;1]. Так, на рис. 2 для ПСЧ R_1 считают, что наступило событие $A2$. Если ПСЧ оказалось равным R_2, считают, что наступило событие $A(N-1)$.

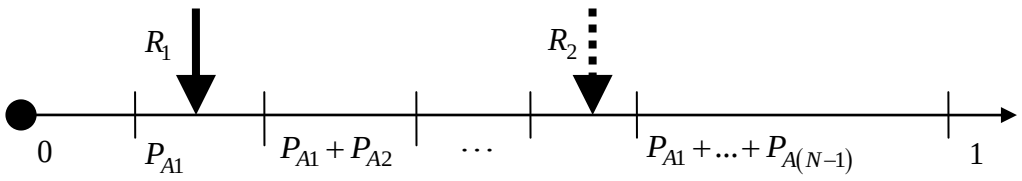

Рис. 2. Моделирование полной группы несовместных событий.

Если группа событий не является полной, вводят *дополнительное (фиктивное)* событие $A(N+1)$, вероятность которого определяют по формуле:

$$P_{A(N+1)} = \sum_{i=1}^{N} P_{Ai} \ .$$

Далее действуют по уже изложенному алгоритму для полной группы событий с одним изменением: если ПСЧ попадает в последний, $(N+1)$-й интервал, считают, что *ни одно из N событий,* составляющих неполную группу, *не наступило.*

В практике имитационных исследований часто возникает необходимость моделирования *зависимых событий,* для которых вероятность наступления одного события оказывается зависящей от того, наступило или не наступило другое событие. В качестве одного из примеров зависимых событий приведем доставку груза потребителю в двух случаях: когда маршрут движения известен и был поставщиком дополнительно уточнен, и когда уточнения движения груза не проводилось. Понятно, что вероятность доставки груза от поставщика к потребителю для приведенных случаев будет различной.

Для того чтобы провести моделирование двух зависимых случайных событий *A* и *B,* необходимо задать следующие полные и условные вероятности:

$$P(A);\ P(B);\ P(B/A);\ P(B\bar{A}).$$

Заметим, что, если вероятность наступления события *B* при условии, что событие *A* не наступило, не задана, ее можно определить по формуле:

$$P(B/\bar{A}) = \frac{P(B) - P(A)\cdot P(B/A)}{1 - P(A)}.$$

Существуют *два алгоритма* моделирования зависимых событий. Один из них условно можно назвать "последовательным моделированием"; другой — "моделированием *после предварительных расчетов".*

Последовательное моделирование

Алгоритм последовательного моделирования представлен на рис. 3.

Несомненными достоинствами данного алгоритма являются его *простота* и *естественность,* поскольку зависимые события "разыгрываются" последовательно — так, как они наступают (или не наступают) в реальной жизни, что и является *характерной особенностью большинства имитационных моделей.* Вместе с тем алгоритм предусматривает троекратное обращение к датчику случайных чисел (ДСЧ), что увеличивает время моделирования.

Рис. 3. "Последовательное моделирование" зависимых событий.

Моделирование после предварительных расчетов

Приведенные на рис. 3 четыре исхода моделирования зависимых событий образуют *полную группу несовместных событий.* На этом основан алгоритм моделирования, предусматривающий *предварительный расчет* вероятностей каждого из исходов и

"розыгрыш" факта наступления одного из них, как для любой группы несовместных событий. Рис. 4 иллюстрирует разбиение интервала [0;1] на четыре отрезка, длины которых соответствуют вероятностям исходов наступления событий.

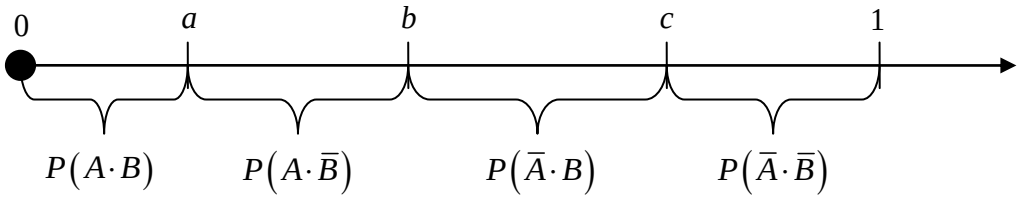

Рис. 4. Разбиение интервала [0;1] для реализации алгоритма моделирования зависимых событий "после предварительных расчетов".

На рис. 5 представлен алгоритм моделирования. Данный алгоритм предусматривает одно *обращение* к датчику случайных чисел, что обеспечивает выигрыш во времени имитации по сравнению с "последовательным моделированием", однако перед началом работы алгоритма исследователь должен рассчитать и ввести вероятности реализации всех возможных исходов (естественно, эту несложную процедуру можно также оформить программно, но это несколько удлинит алгоритм).

Рис. 5. Алгоритм моделирования зависимых случайных событий "после предварительных расчетов".

Лекция №14

Содержание лекции

Имитационные модели информационных систем

Технология моделирования случайных факторов

Моделирование случайных величин

В практике создания и использования имитационных моделей весьма часто приходится сталкиваться с *необходимостью моделирования важнейшего класса факторов — случайных величин* (СВ) различных типов.

Случайной называют переменную величину, которая в результате *испытания* принимает то или иное значение, причем заранее *неизвестно, какое именно.* При этом под *испытанием* понимают *реализацию некоторого* (вполне определенного) *комплекса условий.* В зависимости от множества возможных значений различают три типа СВ:

- *непрерывные*;
- *дискретные*;
- *смешанного типа*.

Исчерпывающей характеристикой любой СВ является ее *закон распределения,* который может быть задан в различных формах: *функции распределения* — для всех типов СВ; *плотности вероятности (распределения)* — для непрерывных СВ; *таблицы или ряда распределения* — для дискретных СВ.

Моделирование непрерывных случайных величин

Моделирование СВ заключается в определении ("розыгрыше") в нужный по ходу имитации момент времени *конкретного значения* СВ в соответствии с *требуемым (заданным) законом распределения.*

Наибольшее распространение получили три метода:

- метод *обратной функции*;
- метод *исключения (Неймана)*;
- метод *композиций*.

Метод обратной функции

Метод позволяет при моделировании СВ учесть все ее статистические свойства и основан на следующей теореме:

Если непрерывная СВ Y имеет плотность вероятности $f(y)$, *то СВ X, определяемая преобразованием*

$$x = \int_{-\infty}^{y} f(y)dy = F(y),$$

имеет равномерный закон распределения на интервале [0;1].

Теорему доказывает следующая цепочка рассуждений, основанная на определении понятия "функция распределения" и условии теоремы:

$$F(x) = P(X < y) = P(Y < y) = F(y) = \int_{-\infty}^{y} f(y)dy = x.$$

Таким образом, получили равенство $F(x) = x$, а это и означает, что СВ X распределена равномерно в интервале [0;1].

Напомним, что в общем виде функция распределения равномерно распределенной на интервале $[a;b]$ СВ X имеет вид:

$$F(x) = \begin{cases} 0, x < a; \\ \dfrac{x-a}{b-a}, a \le x \le b; \\ 1, x > b. \end{cases}$$

Теперь можно найти *обратное преобразование* функции распределения $F^{-1}(x)$.

Если такое преобразование *существует* (условием этого *является наличие первой производной* у функции распределения), алгоритм метода включает всего два шага:

- моделирование ПСЧ, равномерно распределенного на интервале [0;1];
- подстановка этого ПСЧ в обратную функцию и вычисление значения СВ *Y*:

$$x = F(y) \Rightarrow y = F^{-1}(x).$$

При необходимости эти два шага повторяются столько раз, сколько возможных значений СВ *Y* требуется получить.

Простота метода обратной функции позволяет сформулировать такой вывод: *если обратное преобразование функции распределения СВ,* возможные значения которой необходимо получить, *существует, следует использовать именно этот метод.* К сожалению, круг СВ с функциями распределения, допускающими обратное преобразование, не столь широк, что потребовало разработки иных методов.

Метод исключения (Неймана)

Метод Неймана позволяет из совокупности равномерно распределенных ПСЧ R_i, по определенным правилам выбрать совокупность значений y_i с требуемой функцией распределения $f(y)$.

Алгоритм метода

1. Выполняется *усечение исходного распределения* таким образом, чтобы область возможных значений СВ *Y* совпадала с интервалом $[a;b]$.

В результате формируется плотность вероятности $f^*(y)$ такая, что

$$\int_a^b f^*(y)\,dy = 1.$$

Длина интервала $[a;b]$ определяется требуемой точностью моделирования значений СВ в рамках конкретного исследования.

2. Генерируется пара ПСЧ R_1 и R_2, равномерно распределенных на интервале [0;1].

3. Вычисляется пара ПСЧ R_1^* и R_2^* по формулам:

$$R_1^* = a + (b-a)\cdot R_1;$$

$$R_2^* = M \cdot R_2,$$

где $M = \max\limits_{y\in[a;b]} f^*(y)$.

На координатной плоскости пара чисел $\left(R_1^*; R_2^*\right)$ определяет *точку* — например, точку Q_1 на рис. 1. На рисунке обозначены: A — прямоугольник, ограничивающий график плотности распределения моделируемой СВ; D — область прямоугольника A, находящаяся ниже графика $f^*(y)$; B — область прямоугольника A, находящаяся выше графика $f^*(y)$.

4. Если точка (Q_1) принадлежит области D, считают, что *получено первое*

требуемое значение СВ $y_1 = R_1^*$.

5. *Генерируется следующая пара* ПСЧ R_3 и R_4 равномерно распределенных на интервале [0;1], после пересчета по п. 3 задающих на координатной плоскости вторую точку — Q_2.

6. Если точка (Q_2) *принадлежит области* В, переходят к моделированию следующей пары ПСЧ $(R_5; R_6)$ и т.д. *до получения необходимого количества* ПСЧ.

Рис. 1. Моделирование СВ методом Неймана.

Очевидно, что в ряде случаев (при попадании изображающих точек в область В соответствующие ПСЧ с нечетными индексами не могут быть включены в требуемую выборку возможных значений моделируемой СВ, причем это будет происходить тем чаще, чем сильнее график $f^*(y)$ по форме будет "отличаться" от прямоугольника А. Оценить среднее относительное число q "пустых" обращений к генератору ПСЧ можно геометрическим методом, вычислив отношение площадей соответствующих областей (В и А):

$$S_{\text{A}} = M \cdot (b-a) = S_{\text{B}} + S_{\text{D}} = S_{\text{B}} + 1;$$

$$S_{\text{B}} = S_{\text{A}} - 1;$$

$$q = \frac{S_{\text{B}}}{S_{\text{A}}} = 1 - \frac{1}{S_{\text{A}}} = 1 - \frac{1}{M \cdot (b-a)}.$$

Главным достоинством метода Неймана является его *универсальность* — применимость для генерации СВ, имеющих *любую вычислимую или заданную таблично плотность вероятности.*

Метод композиции

Применение метода основано на теоремах теории вероятностей, доказывающих *представимость одной СВ композицией двух или более СВ, имеющих относительно простые, более легко реализуемые законы распределения.* Наиболее часто данным методом пользуются для генерации ПСЧ, имеющих *нормальное распределение. Согласно центральной предельной теореме* распределение СВ *Y*, задаваемой преобразованием

$$y = \frac{1}{\sqrt{\frac{k}{12}}} \cdot \left(\sum_{i=1}^{k} R_i - \frac{k}{2} \right),$$

где R_i — равномерно распределенные на интервале [0;1] ПСЧ, при росте *k неограниченно приближается к нормальному распределению* со стандартными параметрами $\left[m_y = 0; \sigma_y = 1\right]$.

Последнее обстоятельство легко подтверждается следующим образом. Введем СВ *Z* и найдем параметры ее распределения, используя соответствующие теоремы о математическом ожидании и дисперсии суммы СВ:

$$Z = \sum_{i=1}^{k} R_i ;$$

$$M(Z) = m_z = M\left[\sum_{i=1}^{k} m_r\right] = \sum_{i=1}^{k} m_r = \sum_{i=1}^{k} \frac{1}{2} = \frac{k}{2};$$

$$\sigma_z = \sqrt{D_z} = \sqrt{\sum_{i=1}^{k} D_r} = \sqrt{\sum_{i=1}^{k} \frac{1}{12}} = \sqrt{\frac{k}{12}}.$$

Напомним, что при равномерном распределении в интервале [0;1] СВ имеет параметры:

$$m_r = \frac{1}{2};\ D_r = \frac{1}{12}.$$

Очевидно, что

$$Y = \frac{Z - m_z}{\sigma_z},$$

и, как любая центрированно-нормированная СВ, имеет стандартные параметры.

Как правило, берут $k = 12$ и считают, что для подавляющего числа практических задач обеспечивается должная точность вычислений. Если же к точности имитации предъявляются особые требования, можно *улучшить качество* моделирования СВ *за счет введения нелинейной поправки*:

$$y_{\text{мод}} = y(k) = \frac{1}{20 \cdot k} \cdot \left[y^3(k) - 3 \cdot y(k)\right],$$

где $y(k)$ — возможное значение СВ *Y*, полученное в результате сложения, центрирования и нормирования *k* равномерно распределенных ПСЧ R_i.

В целом можно сделать вывод о том, что метод композиции применим и дает хорошие результаты тогда, когда из теории вероятностей известно, композиция каких легко моделируемых СВ позволяет получить СВ с требуемым законом распределения.

Моделирование дискретных случайных величин

Дискретные случайные величины (ДСВ) достаточно часто используются при моделировании систем. *Основными методами* генерации возможных значений ДСВ являются:

- метод *последовательных сравнений*;
- метод *интерпретации*.

Метод последовательных сравнений

Алгоритм метода практически совпадает с ранее рассмотренным алгоритмом моделирования *полной группы несовместных случайных событий,* если считать номер события номером возможного значения ДСВ, а вероятность наступления события — вероятностью принятия ДСВ этого возможного значения. На рис. 2 показана схема определения номера возможного значения ДСВ, полученного на очередном шаге.

Из анализа ситуации, показанной на рис. 2 для ПСЧ R, "попавшего" в интервал $[P_1; P_1 + P_2]$, следует сделать вывод, что ДСВ приняла свое второе возможное значение; а для ПСЧ R' — что ДСВ приняла свое $(N-1)$-е значение и т.д. Алгоритм последовательных сравнений можно улучшить (ускорить) за счет применения методов оптимизации перебора — дихотомии (метода половинного деления); перебора с предварительным ранжированием вероятностей возможных значений по убыванию и т.п.

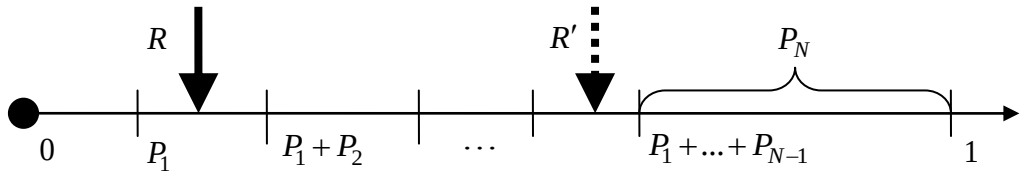

Рис. 2. Моделирование ДСВ методом последовательных сравнений.

Метод интерпретации

Метод основан на использовании *модельных аналогий с сущностью физических явлений,* описываемых моделируемыми законами распределения.

На практике метод чаще всего используют для моделирования *биномиального закона распределения,* описывающего число успехов в *n независимых* опытах с вероятностью *успеха в каждом* испытании *p* и вероятностью неудачи $q = 1 - p$.

Алгоритм метода для этого случая весьма прост:

- моделируют *n* равномерно распределенных на интервале [0;1] ПСЧ;
- подсчитывают число *m* тех из ПСЧ, которые меньше *p*;
- это число *m* и считают возможным значением моделируемой ДСВ, подчиненной биномиальному закону распределения.

Помимо перечисленных, существуют и другие методы моделирования ДСВ, основанные на специальных свойствах моделируемых распределений или на связи между распределениями.

Моделирование случайных векторов

Случайным вектором (системой случайных величин) называют *совокупность случайных величин, совместно характеризующих какое-либо случайное явление*: где X_i – СВ с теми или иными законами распределения.

$$X = (X_1, X_2, ..., X_n).$$

Исчерпывающей характеристикой случайного вектора является *совместная многомерная функция распределения его компонент* $F(x_1, x_2, ..., x_n)$ или соответствующая ему *совместная многомерная плотность вероятности* $f(x_1, x_2, ..., x_n)$.

Проще всего моделировать случайный вектор с *независимыми* компонентами, для которого

$$f(x_1, x_2, ..., x_n) = \prod_{i=1}^{n} f_i(x_i)$$

справедливо, т.е. *каждую* из компонент случайного вектора *можно моделировать независимо* от других в соответствии с ее *"собственной" плотностью вероятности* $f_i(x_i)$.

В случае, когда компоненты случайного вектора статистически зависимы, необходимо использовать специальные методы:

- метод *условных распределений*;
- метод *исключения (Неймана)*;
- метод *линейных преобразований*.

Метод условных распределений

Метод основан на *рекуррентном вычислении условных плотностей вероятностей для каждой из компонент случайного вектора X с многомерной совместной плотностью вероятности* $f(x_1, x_2, ..., x_n)$.

Для плотности распределения случайного вектора *X* можно записать:

$$f(x_1, x_2, ..., x_n) = f_1(x_1) \cdot f_2(x_2/x_1) \cdot f_3(x_3/x_2, x_1) \cdot ... \cdot f_n(x_n/x_{n-1}, x_{n-2}, ..., x_1),$$

где $f_1(x_1)$ — плотность распределения СВ X_1; $f_k(x_k/x_{k-1}, x_{k-2}, ..., x_1)$ — плотность условного распределения СВ X_k при условии: $X_1 = x_1$; $X_2 = x_2$; …; $X_{k-1} = x_{k-1}$.

Для получения указанных плотностей необходимо провести *интегрирование совместной плотности распределения случайного вектора в соответствующих пределах*:

$$\begin{cases} f_1(x_1) = \int \ldots \int\limits_{S_1} f(x_1, x_2, \ldots, x_n)\, dx_2 \ldots dx_n; \\ f_2(x_2/x_1) = \int \ldots \int\limits_{S_2} \dfrac{f(x_1, x_2, \ldots, x_n)}{f_1(x_1)}\, dx_3 \ldots dx_n; \\ \ldots \\ f_n(x_n/x_{n-1}, x_{n-2}, \ldots, x_1) = \int\limits_{S_n} \dfrac{f(x_1, x_2, \ldots, x_n)}{f_1(x_1) \cdot f_2(x_2/x_1) \cdot f_3(x_3/x_2, x_1) \cdot \ldots \cdot f_n(x_{n-1}/x_{n-2}, x_{n-3}, \ldots, x_1)}\, dx_n. \end{cases}$$

Порядок моделирования:

- моделировать значение x_1^* СВ X_1 по закону $f_1(x_1)$;
- моделировать значение x_2^* СВ X_2 по закону $f_2\left(x_2/x_1^*\right)$;
- ...;
- моделировать значение x_n^* СВ X_n по закону $f_n\left(x_n/x_{n-1}^*, x_{n-2}^*, \ldots, x_1^*\right)$.

Тогда вектор $\left(x_1^*, x_2^*, \ldots, x_n^*\right)$ и есть реализация искомого случайного вектора *X*.

Метод условных распределений (как и метод обратной функции для скалярной СВ) позволяет учесть *все статистические свойства случайного вектора.* Поэтому справедлив вывод: *если имеется возможность получить условные плотности распределения* $f_k(x_k/x_{k-1}, x_{k-2}, \ldots, x_1)$, *следует пользоваться именно этим методом.*

Метод исключения (Неймана)

Метод *является обобщением* уже рассмотренного для СВ метода Неймана на случай *n* переменных. Предполагается, что все компоненты случайного вектора распределены *в конечных интервалах*

$$x_i \in [a_i; b_i]. \quad i = 1, 2, \ldots, n.$$

Если это не так, необходимо произвести *усечение* плотности распределения для выполнения данного условия.

Алгоритм метода:

1. Генерируются $(n+1)$ ПСЧ: $R_1, R_2, \ldots, R_n$; R_f, распределенных, соответственно, на интервалах $[a_1; b_1]$, $[a_2; b_2]$,…,$[a_n; b_n]$; $[0; f_{\max}]$;

$$f_{\max} = \max \ldots \max_{x_i \in [a_i; b_i]; i=1,2,\ldots,n} f(x_1, x_2, \ldots, x_n).$$

2. Если выполняется условие: $R_f \le f(R_1, R_2, \ldots, R_n)$, то вектор $(R_1, R_2, \ldots, R_n)$ и есть искомая реализация случайного вектора.

3. Если данное условие не выполняется, переходят к первому пункту и т.д.

Рис. 3 содержит иллюстрацию данного алгоритма для двумерного случая.

Возврат к п. 1 после "неудачного" моделирования *n* ПСЧ происходит тогда, когда точка Q окажется выше поверхности, представляющей двумерную плотность вероятности $f(x_1, x_2)$. Для случая, представленного на рисунке, в качестве (очередной) реализации двумерного случайного вектора следует взять пару ПСЧ $(R_1; R_2)$.

Среднюю относительную частоту "неудач" *можно вычислить геометрическим способом, взяв отношение объемов соответствующих фигур.*

Для одномерного случая, *основным достоинством* метода Неймана является его *универсальность.* Однако для плотностей вероятностей, поверхности которых имеют острые пики, достаточно часто будут встречаться "пустые" прогоны, когда очередные *n* ПСЧ бракуются. Этот *недостаток тем существеннее,* чем больше *размерность* моделируемого вектора (n) и *длиннее требуемая выборка* реализаций случайного вектора. На практике такие ситуации встречаются не слишком часто, поэтому метод исключений и имеет столь широкое распространение.

Рис. 3. Моделирование двумерного случайного вектора методом Неймана.

Метод линейных преобразований

Метод линейных преобразований является одним из наиболее распространенных так называемых *корреляционных методов,* применяемых в случаях, когда при моделировании непрерывного *n*-мерного случайного вектора достаточно обеспечить лишь *требуемые значения элементов корреляционной матрицы* этого вектора (это особенно важно для случая *нормального* распределения, для которого выполнение названного требования означает выполнение *достаточного условия полного статистического соответствия теоретического и моделируемого распределений*).

Идея метода заключается в *линейном преобразовании случайного n-мерного вектора Y с независимыми (чаще всего — нормально распределенными) компонентами в случайный вектор X с требуемыми корреляционной матрицей и вектором математических ожиданий компонент.*

Математическая постановка задачи выглядит следующим образом.

Дано: корреляционная матрица и математическое ожидание вектора *X*

$$Q = \|q_{ij}\| = \left\| M\left[\left(X_i - m_{x_i} \right) \cdot \left(X_j - m_{x_j} \right) \right] \right\|;$$

$$M = \left(m_{x_1}, m_{x_2}, ..., m_{x_n} \right)^T.$$

Требуется: найти такую матрицу B, которая позволяла бы в результате преобразования

$$X = B \cdot Y + M, \tag{1}$$

где Y — n-мерный вектор с *независимыми нормально распределенными компонентами со стандартными параметрами, получить вектор X с требуемыми характеристиками.*

Будем искать матрицу B в виде *нижней треугольной* матрицы, все элементы которой, расположенные выше главной диагонали, равны 0. Перейдем от матричной записи к системе алгебраических уравнений:

$$\begin{Vmatrix} X_1 \\ X_2 \\ . \\ . \\ X_n \end{Vmatrix} = \begin{Vmatrix} b_{11} & 0 & 0 & . & 0 \\ b_{12} & b_{22} & 0 & . & 0 \\ . & . & . & . & 0 \\ . & . & . & . & 0 \\ b_{1n} & b_{2n} & b_{3n} & . & b_{nn} \end{Vmatrix} \cdot \begin{Vmatrix} Y_1 \\ Y_2 \\ . \\ . \\ Y_n \end{Vmatrix} + \begin{Vmatrix} m_{x_1} \\ m_{x_2} \\ . \\ . \\ m_{x_n} \end{Vmatrix} \Rightarrow$$

$$\begin{cases} X_1 - m_{x_1} = b_{11} \cdot Y_1; \\ X_2 - m_{x_2} = b_{21} \cdot Y_1 + b_{22} \cdot Y_2; \\ . \\ . \\ X_n - m_{x_n} = b_{n1} \cdot Y_1 + b_{n2} \cdot Y_2 + \ldots + b_{nn} \cdot Y_n. \end{cases} \tag{2}$$

Поскольку компоненты вектора Y *независимы и имеют стандартные параметры,* справедливо выражение:

$$M\left[Y_i \cdot Y_j\right] = \begin{cases} 0, & i \neq j; \\ 1, & i = j. \end{cases}$$

Почленно перемножив сами на себя и между собой соответственно левые и правые части уравнений системы (2) и взяв от результатов перемножения математическое ожидание, получим систему уравнений вида:

$$\begin{cases} M\left[\left(X_1 - m_{x_1}\right) \cdot \left(X_1 - m_{x_1}\right)\right] = M\left[b_{11} \cdot Y_1 \cdot b_{11} \cdot Y_1\right]; \\ M\left[\left(X_1 - m_{x_1}\right) \cdot \left(X_1 - m_{x_2}\right)\right] = M\left[b_{11} \cdot Y_1 \cdot \left(b_{21} \cdot Y_1 + b_{22} \cdot Y_2\right)\right]; \\ \ldots \end{cases}$$

Как легко увидеть, в левых частях полученной системы уравнений — элементы заданной корреляционной матрицы Q а в правых — элементы искомой матрицы B.

Последовательно решая эту систему, получаем формулы для расчета элементов b_{ij}:

$$b_{11} = \sqrt{q_{11}};\ b_{21} = \frac{q_{12}}{\sqrt{q_{11}}};\ b_{22} = \sqrt{q_{22} - \frac{q_{12}}{q_{11}}};\ldots$$

Формула для расчета любого элемента матрицы преобразования B имеет вид:

$$b_{ij} = \frac{q_{ij} - \sum_{k=1}^{j-1} b_{ik} \cdot b_{jk}}{\sqrt{q_{ij} - \sum_{k=1}^{j-1} b_{jk}^2}};\ 1 \le j \le i \le n\,.$$

Таким образом, алгоритм метода линейных преобразований весьма прост:

- по заданной корреляционной матрице *рассчитывают значения коэффициентов матрицы преобразования B*;
- *генерируют, одну реализацию вектораY*, компоненты которого независимы и распределены нормально со стандартными параметрами;
- полученный вектор *подставляют* в выражение (1) и *определяют очередную реализацию вектора X*, имеющего заданные корреляционную матрицу и вектор математических ожиданий компонент;
- *при необходимости* два предыдущих шага алгоритма *повторяют требуемое число раз* (до получения нужного количества реализаций вектора *X*).

Как правило, все *современные программные средства,* применяемые для реализации тех или иных имитационных моделей, *содержат встроенные генераторы равномерно распределенных ПСЧ, что позволяет исследователю легко моделировать любые случайные факторы.*

Лекция №15

Содержание лекции

Имитационные модели информационных систем

Основы организации имитационного моделирования

Этапы имитационного моделирования

Имитационное моделирование применяют для исследования *сложных информационных систем.* Естественно, что и *имитационные модели оказываются достаточно сложными как с точки зрения заложенного в них математического аппарата, так и в плане машинной реализации.* При этом *сложность любой модели* определяется двумя факторами:

- *сложностью* исследуемого *объекта-оригинала*;
- *точностью,* предъявляемой к *результатам расчетов*.

Использование машинного эксперимента как средства решения сложных прикладных проблем, несмотря на присущую каждой конкретной задаче специфику, имеет *ряд общих черт (этапов).* На рис. 1 представлены этапы применения математической (имитационной) модели (по взглядам академика А. А. Самарского).

Каждому из показанных на рисунке этапов присущи *собственные приемы, методы, технологии.* Отметим, что все эти этапы носят ярко выраженный *творческий характер* и требуют от разработчика модели особой подготовки.

После того как имитационная модель реализована на ЭВМ, исследователь должен выполнить последовательно следующие этапы (их часто называют технологическими):

- *испытание* модели;
- *исследование свойств* модели;
- *планирование имитационного эксперимента*;
- *эксплуатация модели (проведение расчетов)*.

Охарактеризуем первые два этапа.

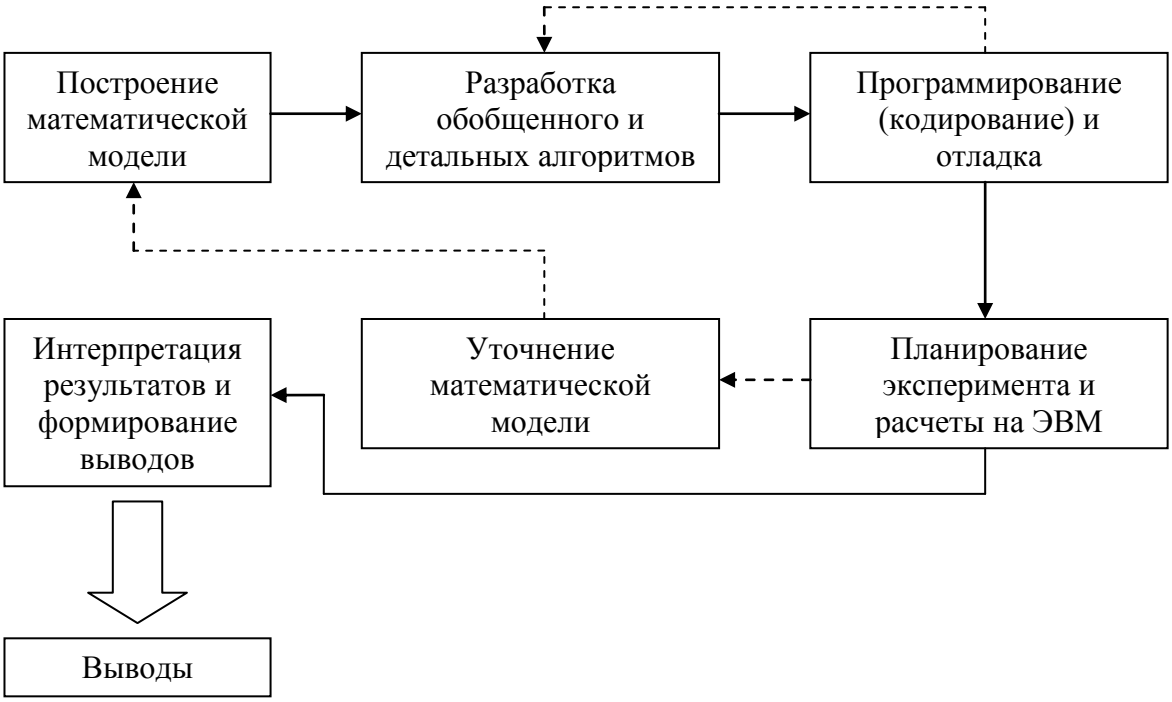

Рис. 1. Этапы машинного эксперимента.

Испытание имитационной модели

Испытание имитационной модели включает работы по четырем направлениям:

- задание *исходной информации*;
- *верификацию* имитационной модели;
- *проверку адекватности* модели;
- *калибровку* имитационной модели.

Задание исходной информации

Процедура задания исходной информации полностью определяется *типом моделируемой системы*:

- если моделируется *функционирующая (существующая)* система, проводят *измерение характеристик ее функционирования* и затем используют эти данные в качестве исходных при моделировании;
- если моделируется *проектируемая* система, проводят *измерения на прототипах*;
- если прототипов нет, используют *экспертные оценки параметров и переменных модели,* формализующих характеристики реальной системы.

Каждому из этих вариантов присущи собственные особенности и сложности. Так, проведение измерений на существующих и проектируемых системах требует применения *качественных измерительных* средств, а проведение экспертного оценивания исходных данных представляет собой *комплекс достаточно сложных процедур* получения, обработки и интерпретации экспертной информации.

Верификация имитационной модели

Верификация имитационной модели состоит в *доказательстве утверждений соответствия алгоритма ее функционирования цели моделирования путем формальных и неформальных исследований реализованной программы модели.*

Неформальные исследования представляют собой *ряд процедур, входящих в автономную и комплексную отладку. Формальные методы* включают:

- использование *специальных процессоров — "читателей" программ*;
- замену *стохастических элементов* модели *детерминированными*;
- *тест* на так называемую *непрерывность моделирования* и др.

Проверка адекватности модели

Количественную оценку адекватности модели объекту исследования проводят для случая, когда *можно определить значения отклика системы в ходе натурных испытаний.*

Наиболее распространены *три способа* проверки:

- по *средним значениям откликов модели и системы*;
- по *дисперсиям отклонений откликов*;
- по *максимальному значению абсолютных отклонений откликов*.

Если *возможность измерения отклика реальной системы отсутствует,* оценку адекватности модели проводят на *основе субъективного суждения* соответствующего *должностного лица о возможности использования результатов,* полученных с использованием этой модели, *при выполнении им служебных обязанностей* (в частности — при обосновании решений).

Калибровка имитационной модели

К калибровке имитационной модели приступают в случае, когда *модель оказывается неадекватной реальной системе.* За счет калибровки иногда удается *уменьшить неточности описания отдельных подсистем (элементов) реальной системы* и тем самым *повысить достоверность получаемых модельных результатов.*

В модели при калибровке возможны *изменения трех типов*:

- *глобальные структурные* изменения;
- *локальные структурные* изменения;
- изменение так называемых *калибровочных параметров* в результате реализации

достаточно сложной итерационной процедуры, включающей многократное построение регрессионных зависимостей и статистическую оценку значимости улучшения модели на очередном шаге.

При необходимости проведения некоторых локальных и особенно глобальных структурных изменений приходится *возвращаться к содержательному описанию* моделируемой системы и искать *дополнительную информацию* о ней.

Исследование свойств имитационной модели

После испытаний имитационной модели переходят к изучению ее свойств. При этом наиболее важны *четыре процедуры*:

- оценка *погрешности имитации*;
- определение *длительности переходного режима* в имитационной модели;
- оценка *устойчивости результатов* имитации;
- исследование *чувствительности* имитационной модели.

Оценка погрешности имитации, связанной с использованием в модели генераторов псевдослучайных чисел (ПСЧ)

Исследование качества генераторов ПСЧ проводится известными методами теории вероятностей и математической статистики. *Важнейшим* показателем качества любого генератора ПСЧ является *период последовательности* ПСЧ (при требуемых статистических свойствах). В большинстве случаев о качестве генератора ПСЧ судят по *оценкам математических ожиданий и дисперсий отклонений компонент функции отклика.* Как уже отмечалось, для подавляющего числа практических задач стандартные (встроенные) генераторы дают вполне пригодные последовательности ПСЧ.

Определение длительности переходного режима

Обычно имитационные модели применяются для изучения системы в *типичных* для нее и *повторяющихся* условиях. В большинстве стохастических моделей требуется некоторое время T_0 для достижения моделью *установившегося состояния.*

Под *статистическим равновесием или установившимся состоянием* модели понимают такое состояние, в котором *противодействующие влияния сбалансированы и компенсируют друг друга.* Иными словами: модель находится в равновесии, если ее отклик *не выходит за предельные значения.*

Существуют *три способа* уменьшения влияния начального периода на динамику моделирования сложной системы:

- использование *"длинных прогонов",* позволяющих получать результаты после заведомого выхода модели на установившийся режим;
- *исключение* из рассмотрения *начального периода* прогона;
- *выбор* таких *начальных условий,* которые ближе всего к типичным.

Каждый из этих способов не свободен от недостатков: "длинные прогоны" приводят к *большим затратам* машинного времени; при исключении из рассмотрения начального периода *теряется часть информации*; выбор типичных начальных условий, обеспечивающих быструю сходимость, как правило, *затруднен отсутствием достаточного объема исходных данных* (особенно для принципиально новых систем).

Для отделения переходного режима от стационарного у исследователя должна быть *возможность наблюдения за моментом входа контролируемого параметра в стационарный режим.* Часто используют такой метод: строят графики изменения контролируемого параметра в модельном времени и на нем выявляют переходный режим.

На рис. 2 представлен график изменения k-го контролируемого параметра модели g_k в зависимости от модельного времени t_0. На рисунке видно, что, начиная со времени $t_{\text{перех}}$, этот параметр "вошел" в установившийся режим со средним значением $\overline{g}_k$.

Если построить подобные графики *для всех (или большинства существенных)* контролируемых параметров модели, определить *для каждого* из них длительность переходного режима и выбрать из них *наибольшую,* в большинстве случаев можно считать, что после этого времени *все интересующие исследователя параметры находятся в установившемся режиме.*

Рис. 2. Определение длительности переходного периода для k-го контролируемого параметра модели.

На практике встречаются случаи, *когда переходные режимы исследуются специально.* Понятно, что при этом используют *"короткие прогоны",* исключают из рассмотрения установившиеся режимы и стремятся найти начальные условия моделирования, приводящие к *наибольшей длительности переходных процессов.* Иногда для увеличения точности результатов проводят *замедление изменения системного времени.*

Оценка устойчивости результатов имитации

Под *устойчивостью результатов имитации понимают степень их нечувствительности к изменению входных условий. Универсальной процедуры* оценки устойчивости нет. Практически часто находят дисперсию отклика модели *Y* по нескольким компонентам и проверяют, увеличивается ли она с ростом интервала моделирования. Если *увеличения дисперсии отклика не наблюдается,* результаты имитации считают *устойчивыми.*

Важная практическая рекомендация: чем *ближе структура модели к структуре реальной системы* и чем *выше степень детализации* учитываемых в модели факторов, тем *шире область устойчивости* (пригодности) результатов имитации.

Исследование чувствительности модели

Работы на этом этапе имеют *два направления*:

- установление *диапазона изменения отклика* модели при варьировании каждого параметра;
- проверка *зависимости отклика модели от изменения параметров внешней среды*.

В зависимости от диапазона изменения откликов *Y* при изменении каждой компоненты вектора параметров *X* определяется *стратегия планирования экспериментов* на модели. Если при *значительной амплитуде изменения* некоторой компоненты вектора параметров модели *отклик меняется незначительно,* то *точность* представлении ее в модели *не играет существенной роли.*

Проверка зависимости отклика модели *Y* от изменений параметров внешней среды основана на *расчете соответствующих частных производных и их анализе.*

Языки моделирования

Чтобы реализовать на ЭВМ модель сложной системы, нужен *аппарат моделирования, который в принципе должен быть специализированным.* Он должен предоставлять исследователю:

- удобные способы *организации данных,* обеспечивающие простое и эффективное моделирование;
- удобные *средства формализации и воспроизведения динамических свойств* моделируемой системы;
- возможность *имитации стохастических систем,* т. е. процедур генерации ПСЧ и вероятностного (статистического) анализа результатов моделирования;
- простые и удобные *процедуры отладки и контроля* программы;
- доступные *процедуры восприятия и использования языка* и др.

Вместе с тем существующие языки программирования общего назначения для достаточно широкого круга задач *позволяют без значительных затрат ресурсов* создавать весьма *совершенные имитационные модели.* Можно сказать, что они способны составить *конкуренцию специализированным языкам* моделирования. Для систематизации представлений о средствах реализации имитационных моделей приведем основные определения и краткие сведения о подходах к выбору соответствующего языка.

Языком программирования называют *набор (систему) символов, распознаваемых ЭВМ и обозначающих операции, которые можно реализовать на ЭВМ.* Выделяют машинно-ориентированные, проблемно (процедурно)-ориентированные и объектно-ориентированные языки.

Классические языки моделирования являются *процедурно-ориентированными* и обладают рядом специфических черт. Можно сказать, что основные языки моделирования разработаны как *средство программного обеспечения имитационного подхода к изучению сложных систем*.

Языки моделирования позволяют описывать моделируемые системы в терминах, *разработанных на базе основных понятий имитации. С* их помощью можно *организовать процесс общения Заказчика и Разработчика* модели. Различают языки моделирования *непрерывных и дискретных* процессов.

В настоящее время сложилась ситуация, когда *не следует противопоставлять языки общего назначения* (ЯОН) *и языки имитационного моделирования* (ЯИМ). На рис. 3 представлена классификация языков программирования по различным основаниям, которая может служить основой для формирования рационального подхода к выбору конкретного языка реализации имитационной модели исследуемой ИС.

Легко заметить из названий, что некоторые ЯИМ базируются на конструкциях ЯОН: например, FORSIM — на языке FORTRAN; ПЛИС — на языке PL и т.д.

В силу своего целевого назначения *при правильном выборе* и использовании языки моделирования обладают *рядом понятных* достоинств.

Вместе с тем им присущи и *определенные недостатки,* главными из которых являются *сугубо индивидуальный характер соответствующих трансляторов,* затрудняющий их реализацию на различных ЭВМ; *низкая эффективность рабочих программ*; *сложность процесса отладки программ*; *нехватка документации* (литературы) для пользователей и *специалистов-консультантов* и др. В ряде случаев эти недостатки способны перечеркнуть любые достоинства.

Существует несколько подходов к выбору языка, на котором будет реализовываться разрабатываемая имитационная модель. Предлагается классическая *двухэтапная схема выбора,* имеющая широкое практическое применение.

На *первом этапе* следует найти ответы на следующие вопросы:

1. Имеются ли *руководства* и *инструкции* для пользователей?

2. *Совместим ли язык транслятора с имеющимися вычислительными системами*?

3. Можно ли данный язык *использовать на других вычислительных системах,* способных решать задачи пользователя?

4. *Обеспечивает ли транслятор* языка *выдачу информации об ошибках и глубокую их диагностику*?

5. Насколько *эффективен данный язык с учетом общего времени подготовки, программирования, отладки программы, компиляции и прогона ее на ЭВМ*?

6. Какова *стоимость внедрения, эксплуатации и обновления программного обеспечения для данного языка*?

7. *Знаком ли язык* и, если нет, *легко ли его изучить*?

8. *Оправдает ли частота* использования языка в различных будущих моделях *затраты на его изучение и освоение*?

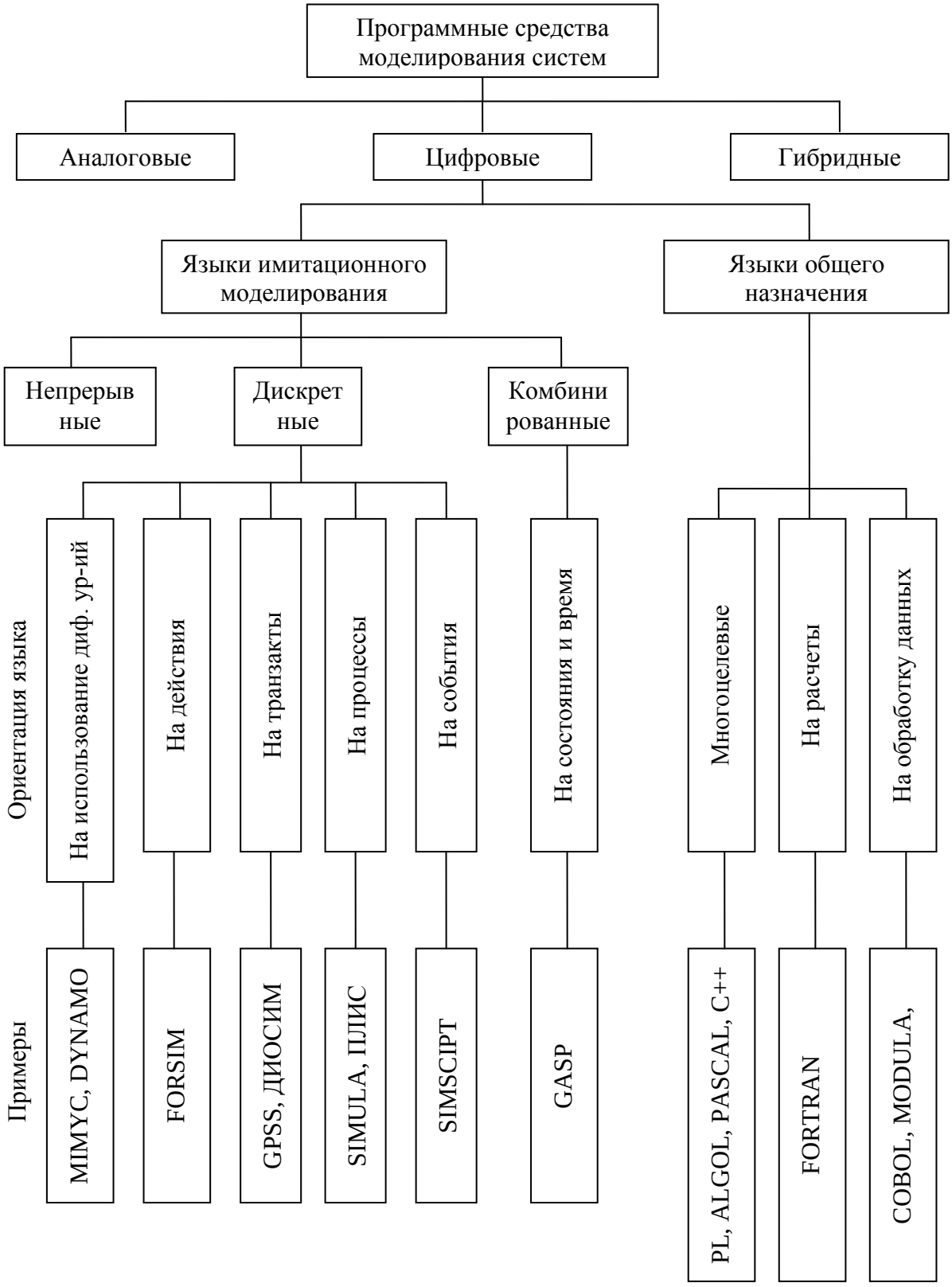

Рис. 3. Классификация программных средств моделирования систем.

По результатам ответов на данные вопросы, как правило, *отбираются несколько языков. Окончательный выбор* основывается на учете характеристик конкретной задачи при ее решении на *определенной* машине.

Второй этап выбора предусматривает поиск ответов на такие вопросы:

1. *Какова область применения языка и его пригодность* для описания явлений реального мира (методы прогнозирования; ориентация; способность генерировать

случайные факторы)?

2. *Насколько легко осуществляется хранение и извлечение данных,* характеризующих состояния системы и работу отдельных ее частей?

3. *Обеспечивается ли необходимая гибкость и каковы возможности* языка в отношении *модифицирования состояний* системы?

4. *Насколько легко данный язык может описывать динамическое поведение*?

5. *Каковы выходные формы документов,* чем они полезны и *какой статистический анализ возможен на основе этих данных*?

6. *Насколько просто вставлять в модель стандартные подпрограммы,* написанные пользователями?

Приведенные вопросы можно *конкретизировать или расширять* с учетом современного уровня и перспектив развития технических, программных средств и информационных технологий, но *изложенный подход* к выбору языка является неизменно *актуальным* и *конструктивным.*

Если попытаться обобщить направленность данных вопросов, то можно заметить, что важнейшими проблемами применения языков моделирования являются их *эффективность, совместимость* с другими программными средствами и *возможность установки на имеющиеся технические средства,* а также *затраты* различных ресурсов. Иными словами, при выборе программного средства моделирования следует руководствоваться известным *критерием "эффективность–время–стоимость",* причем зачастую важность каждого из этих частных показателей *меняется* в зависимости от существа задачи; объема располагаемых ресурсов; резерва (дефицита) времени; сложившихся условий и т.п.