CASE-технологии.

Современные методы и средства проектирования информационных систем

А.М. Вендров

Аннотация

Целью данного обзора является введение в особенности современных методов и средств проектирования информационных систем, основанных на использовании CASE-технологии. Читатель должен получить возможность принятия обоснованного, а не волевого решения относительно использования этих технологий. Приводимые в обзоре рекомендации могут способствовать успешному внедрению CASE-средств и уменьшить риск неправильных инвестиций.

Несмотря на высокие потенциальные возможности CASE-технологии (увеличение производительности труда, улучшение качества программных продуктов, поддержка унифицированного и согласованного стиля работы) далеко не все разработчики информационных систем, использующие CASE-средства, достигают ожидаемых результатов.

Существуют различные причины возможных неудач, но, видимо, основной причиной является неадекватное понимание сути программирования информационных систем и применения CASE-средств. Необходимо понимать, что процесс проектирования и разработки информационной системы на основе CASE-технологии не может быть подобен процессу приготовления пищи по поваренной книге. Всегда следует быть готовым к новым трудностям, связанным с освоением новой технологии, последовательно преодолевать эти трудности и последовательно добиваться нужных результатов.

Обзор предназначен для начинающих и опытных разработчиков информационных систем, для руководителей проектов и системных аналитиков.

СОДЕРЖАНИЕ

Введение

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

- сложность описания (достаточно большое количество функций, процессов, элементов данных и сложные взаимосвязи между ними), требующая тщательного моделирования и анализа данных и процессов;
- наличие совокупности тесно взаимодействующих компонентов (подсистем), имеющих свои локальные задачи и цели функционирования (например, традиционных приложений, связанных с обработкой транзакций и решением регламентных задач, и приложений аналитической обработки (поддержки принятия решений), использующих нерегламентированные запросы к данным большого объема);
- отсутствие прямых аналогов, ограничивающее возможность использования каких-либо типовых проектных решений и прикладных систем;
- необходимость интеграции существующих и вновь разрабатываемых приложений;
- функционирование в неоднородной среде на нескольких аппаратных платформах;
- разобщенность и разнородность отдельных групп разработчиков по уровню квалификации и сложившимся традициям использования тех или иных инструментальных средств;
- существенная временная протяженность проекта, обусловленная, с одной стороны, ограниченными возможностями коллектива разработчиков, и, с другой стороны, масштабами организации-заказчика и различной степенью готовности отдельных ее подразделений к внедрению ИС.

Для успешной реализации проекта объект проектирования (ИС) должен быть прежде всего адекватно описан, должны быть построены полные и непротиворечивые функциональные и информационные модели ИС. Накопленный к настоящему времени опыт проектирования ИС показывает, что это логически сложная, трудоемкая и длительная по времени работа, требующая высокой квалификации участвующих в ней специалистов. Однако до недавнего времени проектирование ИС выполнялось в основном на интуитивном уровне с применением неформализованных методов, основанных на искусстве, практическом опыте, экспертных оценках и дорогостоящих экспериментальных проверках качества функционирования ИС. Кроме того, в процессе создания и функционирования ИС информационные потребности пользователей могут изменяться или уточняться, что еще более усложняет разработку и сопровождение таких систем.

В 70-х и 80-х годах при разработке ИС достаточно широко применялась структурная методология, предоставляющая в распоряжение разработчиков строгие формализованные методы описания ИС и принимаемых технических решений. 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 конкретных ИС встречалось достаточно редко, поскольку при неавтоматизированной (ручной) разработке это практически невозможно. Действительно, вручную очень трудно разработать и графически представить строгие формальные спецификации системы, проверить их на полноту и непротиворечивость, и тем более изменить. Если все же удается создать строгую систему проектных документов, то ее переработка при появлении серьезных изменений практически неосуществима. Ручная разработка обычно порождала следующие проблемы:

- неадекватная спецификация требований;
- неспособность обнаруживать ошибки в проектных решениях;
- низкое качество документации, снижающее эксплуатационные качества;

- затяжной цикл и неудовлетворительные результаты тестирования.

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

Перечисленные факторы способствовали появлению программно-технологических средств специального класса - CASE-средств, реализующих CASE-технологию создания и сопровождения ИС. Термин CASE (Computer Aided Software Engineering) используется в настоящее время в весьма широком смысле. Первоначальное значение термина CASE, ограниченное вопросами автоматизации разработки только лишь программного обеспечения (ПО), в настоящее время приобрело новый смысл, охватывающий процесс разработки сложных ИС в целом. Теперь под термином CASE-средства понимаются программные средства, поддерживающие процессы создания и сопровождения ИС, включая анализ и формулировку требований, проектирование прикладного ПО (приложений) и баз данных, генерацию кода, тестирование, документирование, обеспечение качества, конфигурационное управление и управление проектом, а также другие процессы. CASE-средства вместе с системным ПО и техническими средствами образуют полную среду разработки ИС.

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. Кроме того, появлению CASE-технологии способствовали и такие факторы, как:

- подготовка аналитиков и программистов, восприимчивых к концепциям модульного и структурного программирования;
- широкое внедрение и постоянный рост производительности компьютеров, позволившие использовать эффективные графические средства и автоматизировать большинство этапов проектирования;
- внедрение сетевой технологии, предоставившей возможность объединения усилий отдельных исполнителей в единый процесс проектирования путем использования разделяемой базы данных, содержащей необходимую информацию о проекте.

CASE-технология представляет собой методологию проектирования ИС, а также набор инструментальных средств, позволяющих в наглядной форме моделировать предметную область, анализировать эту модель на всех этапах разработки и сопровождения ИС и разрабатывать приложения в соответствии с информационными потребностями пользователей. 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.

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

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

Ввиду разнообразной природы CASE-средств было бы ошибочно делать какие-либо безоговорочные утверждения относительно реального удовлетворения тех или иных ожиданий от

их внедрения. Можно перечислить следующие факторы, усложняющие определение возможного эффекта от использования CASE-средств:

- широкое разнообразие качества и возможностей CASE-средств;
- относительно небольшое время использования CASE-средств в различных организациях и недостаток опыта их применения;
- широкое разнообразие в практике внедрения различных организаций;
- отсутствие детальных метрик и данных для уже выполненных и текущих проектов;
- широкий диапазон предметных областей проектов;
- различная степень интеграции CASE-средств в различных проектах.

Вследствие этих сложностей доступная информация о реальных внедрениях крайне ограничена и противоречива. Она зависит от типа средств, характеристик проектов, уровня сопровождения и опыта пользователей. Некоторые аналитики полагают, что реальная выгода от использования некоторых типов CASE-средств может быть получена только после одно- или двухлетнего опыта. Другие полагают, что воздействие может реально проявиться в фазе эксплуатации жизненного цикла ИС, когда технологические улучшения могут привести к снижению эксплуатационных затрат.

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

- Технология. Понимание ограниченности существующих возможностей и способность принять новую технологию;
- Культура. Готовность к внедрению новых процессов и взаимоотношений между разработчиками и пользователями;
- Управление. Четкое руководство и организованность по отношению к наиболее важным этапам и процессам внедрения.

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

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

- достоверная оценка отдачи от инвестиций в CASE-средства затруднительна ввиду отсутствия приемлемых метрик и данных по проектам и процессам разработки ПО;
- внедрение CASE-средств может представлять собой достаточно длительный процесс и может не принести немедленной отдачи. Возможно даже краткосрочное снижение продуктивности в результате усилий, затрачиваемых на внедрение. Вследствие этого руководство организации-пользователя может утратить интерес к CASE-средствам и прекратить поддержку их внедрения;
- отсутствие полного соответствия между теми процессами и методами, которые поддерживаются CASE-средствами, и теми, которые используются в данной организации, может привести к дополнительным трудностям;
- CASE-средства зачастую трудно использовать в комплексе с другими подобными средствами. Это объясняется как различными парадигмами, поддерживаемыми различными средствами, так и проблемами передачи данных и управления от одного средства к другому;
- некоторые CASE-средства требуют слишком много усилий для того, чтобы оправдать их использование в небольшом проекте, при этом, тем не менее, можно извлечь выгоду из той дисциплины, к которой обязывает их применение;
- негативное отношение персонала к внедрению новой CASE-технологии может быть главной причиной провала проекта.

Пользователи CASE-средств должны быть готовы к необходимости долгосрочных затрат на эксплуатацию, частому появлению новых версий и возможному быстрому моральному

старению средств, а также постоянным затратам на обучение и повышение квалификации персонала.

Несмотря на все высказанные предостережения и некоторый пессимизм, грамотный и разумный подход к использованию CASE-средств может преодолеть все перечисленные трудности. Успешное внедрение CASE-средств должно обеспечить такие выгоды как:

- высокий уровень технологической поддержки процессов разработки и сопровождения ПО;
- положительное воздействие на некоторые или все из перечисленных факторов: производительность, качество продукции, соблюдение стандартов, документирование;
- приемлемый уровень отдачи от инвестиций в CASE-средства.

1. Основы методологии проектирования ИС

1.1. Жизненный цикл по ИС

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

Основным нормативным документом, регламентирующим ЖЦ ПО, является международный стандарт ISO/IEC 12207 [5] (ISO - International Organization of Standardization - Международная организация по стандартизации, IEC - International Electrotechnical Commission - Международная комиссия по электротехнике). Он определяет структуру ЖЦ, содержащую процессы, действия и задачи, которые должны быть выполнены во время создания ПО.

Структура ЖЦ ПО по стандарту ISO/IEC 12207 базируется на трех группах процессов:

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

Разработка включает в себя все работы по созданию ПО и его компонент в соответствии с заданными требованиями, включая оформление проектной и эксплуатационной документации, подготовку материалов, необходимых для проверки работоспособности и соответствующего качества программных продуктов, материалов, необходимых для организации обучения персонала и т.д. Разработка ПО включает в себя, как правило, анализ, проектирование и реализацию (программирование).

Эксплуатация включает в себя работы по внедрению компонентов ПО в эксплуатацию, в том числе конфигурирование базы данных и рабочих мест пользователей, обеспечение эксплуатационной документацией, проведение обучения персонала и т.д., и непосредственно эксплуатацию, в том числе локализацию проблем и устранение причин их возникновения, модификацию ПО в рамках установленного регламента, подготовку предложений по совершенствованию, развитию и модернизации системы.

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 ПО, обучение персонала и т.п. Обеспечение качества проекта связано с проблемами верификации, проверки и тестирования ПО. Верификация - это процесс определения того, отвечает ли текущее состояние разработки, достигнутое на данном этапе, требованиям этого этапа. Проверка позволяет оценить соответствие параметров разработки с исходными требованиями. Проверка частично совпадает с тестированием, которое связано с идентификацией различий между действительными и ожидаемыми результатами и оценкой соответствия характеристик ПО исходным требованиям. В процессе реализации проекта важное

место занимают вопросы идентификации, описания и контроля конфигурации отдельных компонентов и всей системы в целом.

Управление конфигурацией является одним из вспомогательных процессов, поддерживающих основные процессы жизненного цикла ПО, прежде всего процессы разработки и сопровождения ПО. При создании проектов сложных ИС, состоящих из многих компонентов, каждый из которых может иметь разновидности или версии, возникает проблема учета их связей и функций, создания унифицированной структуры и обеспечения развития всей системы. Управление конфигурацией позволяет организовать, систематически учитывать и контролировать внесение изменений в ПО на всех стадиях ЖЦ. Общие принципы и рекомендации конфигурационного учета, планирования и управления конфигурациями ПО отражены в проекте стандарта ISO 12207-2 [5].

Каждый процесс характеризуется определенными задачами и методами их решения, исходными данными, полученными на предыдущем этапе, и результатами. Результатами анализа, в частности, являются функциональные модели, информационные модели и соответствующие им диаграммы. ЖЦ ПО носит итерационный характер: результаты очередного этапа часто вызывают изменения в проектных решениях, выработанных на более ранних этапах.

1.2. Модели жизненного цикла ПО

Стандарт ISO/IEC 12207 не предлагает конкретную модель ЖЦ и методы разработки ПО (под моделью ЖЦ понимается структура, определяющая последовательность выполнения и взаимосвязи процессов, действий и задач, выполняемых на протяжении ЖЦ. Модель ЖЦ зависит от специфики ИС и специфики условий, в которых последняя создается и функционирует). Его регламенты являются общими для любых моделей ЖЦ, методологий и технологий разработки. Стандарт ISO/IEC 12207 описывает структуру процессов ЖЦ ПО, но не конкретизирует в деталях, как реализовать или выполнить действия и задачи, включенные в эти процессы.

К настоящему времени наибольшее распространение получили следующие две основные модели ЖЦ:

- каскадная модель (70-85 г.г.);
- спиральная модель (86-90 г.г.).

В изначально существовавших однородных ИС каждое приложение представляло собой единое целое. Для разработки такого типа приложений применялся каскадный способ. Его основной характеристикой является разбиение всей разработки на этапы, причем переход с одного этапа на следующий происходит только после того, как будет полностью завершена работа на текущем (рис. 1.1). Каждый этап завершается выпуском полного комплекта документации, достаточной для того, чтобы разработка могла быть продолжена другой командой разработчиков.

Положительные стороны применения каскадного подхода заключаются в следующем [2]:

- на каждом этапе формируется законченный набор проектной документации, отвечающий критериям полноты и согласованности;
- выполняемые в логичной последовательности этапы работ позволяют планировать сроки завершения всех работ и соответствующие затраты.

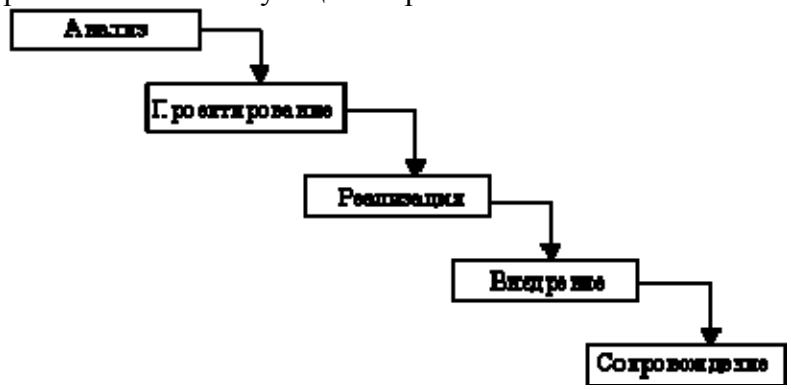


Рис. 1.1. Каскадная схема разработки ПО

Каскадный подход хорошо зарекомендовал себя при построении ИС, для которых в самом начале разработки можно достаточно точно и полно сформулировать все требования, с тем чтобы предоставить разработчикам свободу реализовать их как можно лучше с технической точки зрения. В эту категорию попадают сложные расчетные системы, системы реального времени и другие подобные задачи. Однако, в процессе использования этого подхода обнаружился ряд его недостатков, вызванных прежде всего тем, что реальный процесс создания ПО никогда полностью не укладывался в такую жесткую схему. В процессе создания ПО постоянно возникала потребность в возврате к предыдущим этапам и уточнении или пересмотре ранее принятых решений. В результате реальный процесс создания ПО принимал следующий вид (рис. 1.2):

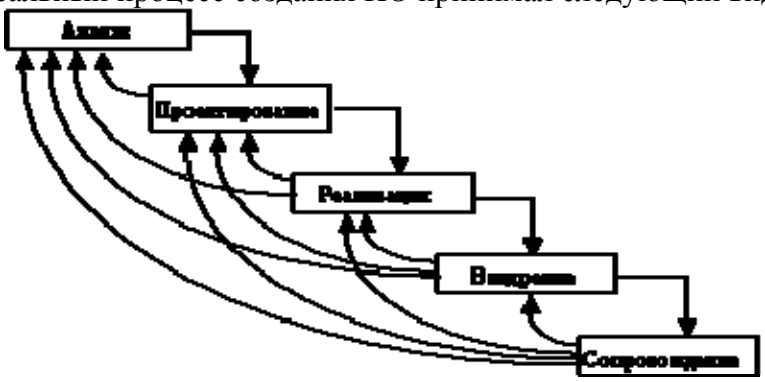


Рис. 1.2. Реальный процесс разработки ПО по каскадной схеме

Основным недостатком каскадного подхода является существенное запаздывание с получением результатов. Согласование результатов с пользователями производится только в точках, планируемых после завершения каждого этапа работ, требования к ИС "заморожены" в виде технического задания на все время ее создания. Таким образом, пользователи могут внести свои замечания только после того, как работа над системой будет полностью завершена. В случае неточного изложения требований или их изменения в течение длительного периода создания ПО, пользователи получают систему, не удовлетворяющую их потребностям. Модели (как функциональные, так и информационные) автоматизируемого объекта могут устареть одновременно с их утверждением.

Для преодоления перечисленных проблем была предложена спиральная модель ЖЦ [10] (рис. 1.3), делающая упор на начальные этапы ЖЦ: анализ и проектирование. На этих этапах реализуемость технических решений проверяется путем создания прототипов. Каждый виток спирали соответствует созданию фрагмента или версии ПО, на нем уточняются цели и характеристики проекта, определяется его качество и планируются работы следующего витка спирали. Таким образом углубляются и последовательно конкретизируются детали проекта и в результате выбирается обоснованный вариант, который доводится до реализации.

Разработка итерациями отражает объективно существующий спиральный цикл создания системы. Неполное завершение работ на каждом этапе позволяет переходить на следующий этап, не дожидаясь полного завершения работы на текущем. При итеративном способе разработки недостающую работу можно будет выполнить на следующей итерации. Главная же задача - как можно быстрее показать пользователям системы работоспособный продукт, тем самым активизируя процесс уточнения и дополнения требований.

Основная проблема спирального цикла - определение момента перехода на следующий этап. Для ее решения необходимо ввести временные ограничения на каждый из этапов жизненного цикла. Переход осуществляется в соответствии с планом, даже если не вся запланированная работа закончена. План составляется на основе статистических данных, полученных в предыдущих проектах, и личного опыта разработчиков.

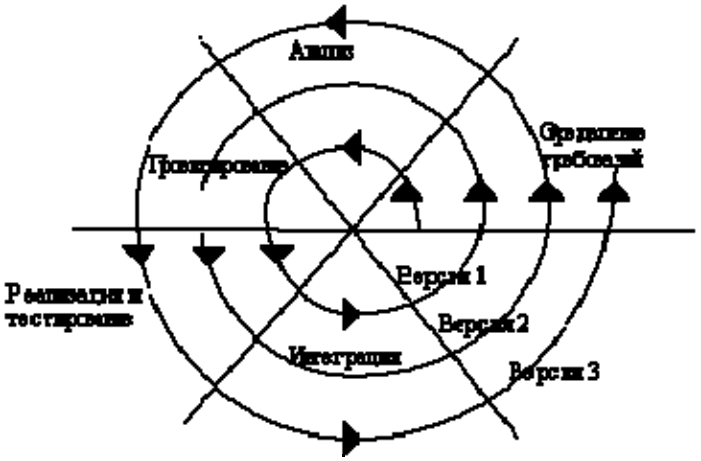


Рис 1.3. Спиральная модель ЖЦ

1.3. Методологии и технологии проектирования ИС

1.3.1. Общие требования к методологии и технологии

Методологии, технологии и инструментальные средства проектирования (CASE-средства) составляют основу проекта любой ИС. Методология реализуется через конкретные технологии и поддерживающие их стандарты, методики и инструментальные средства, которые обеспечивают выполнение процессов ЖЦ.

Технология проектирования определяется как совокупность трех составляющих:

- пошаговой процедуры, определяющей последовательность технологических операций проектирования (рис. 1.4);
- критериев и правил, используемых для оценки результатов выполнения технологических операций;
- нотаций (графических и текстовых средств), используемых для описания проектируемой системы.

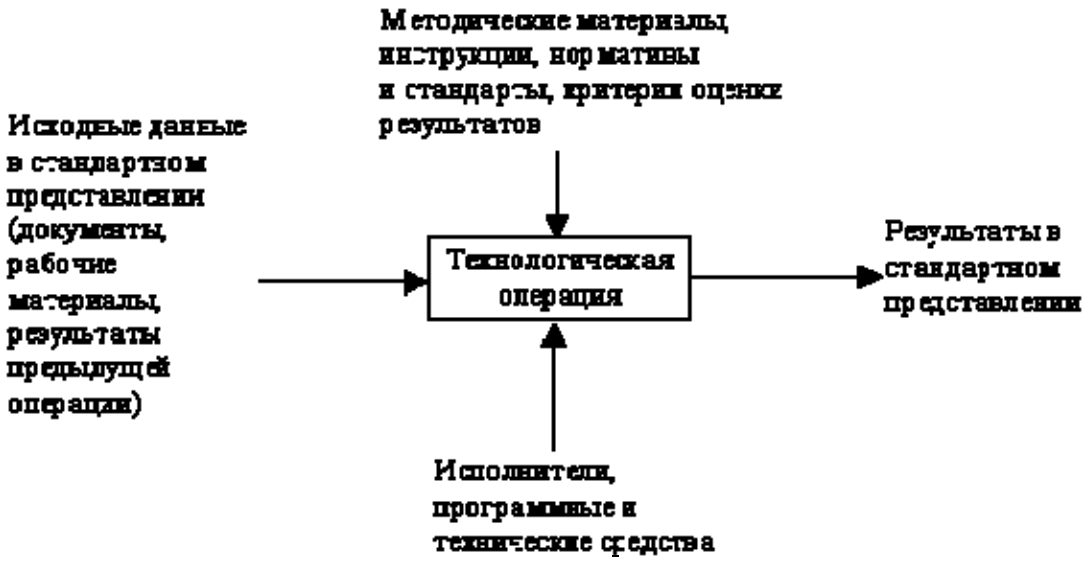


Рис. 1.4. Представление технологической операции проектирования

Технологические инструкции, составляющие основное содержание технологии, должны состоять из описания последовательности технологических операций, условий, в зависимости от которых выполняется та или иная операция, и описаний самих операций.

Технология проектирования, разработки и сопровождения ИС должна удовлетворять следующим общим требованиям:

- технология должна поддерживать полный ЖЦ ПО;
- технология должна обеспечивать гарантированное достижение целей разработки ИС с

заданным качеством и в установленное время;

- технология должна обеспечивать возможность выполнения крупных проектов в виде подсистем (т.е. возможность декомпозиции проекта на составные части, разрабатываемые группами исполнителей ограниченной численности с последующей интеграцией составных частей). Опыт разработки крупных ИС показывает, что для повышения эффективности работ необходимо разбить проект на отдельные слабо связанные по данным и функциям подсистемы. Реализация подсистем должна выполняться отдельными группами специалистов. При этом необходимо обеспечить координацию ведения общего проекта и исключить дублирование результатов работ каждой проектной группы, которое может возникнуть в силу наличия общих данных и функций;
- технология должна обеспечивать возможность ведения работ по проектированию отдельных подсистем небольшими группами (3-7 человек). Это обусловлено принципами управляемости коллектива и повышения производительности за счет минимизации числа внешних связей;
- технология должна обеспечивать минимальное время получения работоспособной ИС. Речь идет не о сроках готовности всей ИС, а о сроках реализации отдельных подсистем. Реализация ИС в целом в короткие сроки может потребовать привлечения большого числа разработчиков, при этом эффект может оказаться ниже, чем при реализации в более короткие сроки отдельных подсистем меньшим числом разработчиков. Практика показывает, что даже при наличии полностью завершенного проекта, внедрение идет последовательно по отдельным подсистемам;
- технология должна предусматривать возможность управления конфигурацией проекта, ведения версий проекта и его составляющих, возможность автоматического выпуска проектной документации и синхронизацию ее версий с версиями проекта;
- технология должна обеспечивать независимость выполняемых проектных решений от средств реализации ИС (систем управления базами данных (СУБД), операционных систем, языков и систем программирования);
- технология должна быть поддержана комплексом согласованных CASE-средств, обеспечивающих автоматизацию процессов, выполняемых на всех стадиях ЖЦ. Общий подход к оценке и выбору CASE-средств описан в разделе 4, примеры комплексов CASE-средств - в подразделе 5.7.

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

- стандарт проектирования;
- стандарт оформления проектной документации;
- стандарт пользовательского интерфейса.
- Стандарт проектирования должен устанавливать:
- набор необходимых моделей (диаграмм) на каждой стадии проектирования и степень их детализации;
- правила фиксации проектных решений на диаграммах, в том числе: правила именования объектов (включая соглашения по терминологии), набор атрибутов для всех объектов и правила их заполнения на каждой стадии, правила оформления диаграмм, включая требования к форме и размерам объектов, и т. д.;
- требования к конфигурации рабочих мест разработчиков, включая настройки операционной системы, настройки CASE-средств, общие настройки проекта и т. д.;
- механизм обеспечения совместной работы над проектом, в том числе: правила интеграции подсистем проекта, правила поддержания проекта в одинаковом для всех разработчиков состоянии (регламент обмена проектной информацией, механизм фиксации общих объектов и т.д.), правила проверки проектных решений на непротиворечивость и т. д.

Стандарт оформления проектной документации должен устанавливать:

- комплектность, состав и структуру документации на каждой стадии проектирования;
- требования к ее оформлению (включая требования к содержанию разделов, подразделов,

пунктов, таблиц и т.д.),

- правила подготовки, рассмотрения, согласования и утверждения документации с указанием предельных сроков для каждой стадии;
- требования к настройке издательской системы, используемой в качестве встроенного средства подготовки документации;
- требования к настройке CASE-средств для обеспечения подготовки документации в соответствии с установленными требованиями.

Стандарт интерфейса пользователя должен устанавливать:

- правила оформления экранов (шрифты и цветовая палитра), состав и расположение окон и элементов управления;
- правила использования клавиатуры и мыши;
- правила оформления текстов помощи;
- перечень стандартных сообщений;
- правила обработки реакции пользователя.

1.3.2. Методология RAD

Одним из возможных подходов к разработке ПО в рамках спиральной модели ЖЦ является получившая в последнее время широкое распространение методология быстрой разработки приложений RAD (Rapid Application Development). Под этим термином обычно понимается процесс разработки ПО, содержащий 3 элемента:

- небольшую команду программистов (от 2 до 10 человек);
- короткий, но тщательно проработанный производственный график (от 2 до 6 мес.);
- повторяющийся цикл, при котором разработчики, по мере того, как приложение начинает обретать форму, запрашивают и реализуют в продукте требования, полученные через взаимодействие с заказчиком.

Команда разработчиков должна представлять из себя группу профессионалов, имеющих опыт в анализе, проектировании, генерации кода и тестировании ПО с использованием CASE-средств. Члены коллектива должны также уметь трансформировать в рабочие прототипы предложения конечных пользователей.

Жизненный цикл ПО по методологии RAD состоит из четырех фаз:

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

На фазе анализа и планирования требований пользователи системы определяют функции, которые она должна выполнять, выделяют наиболее приоритетные из них, требующие проработки в первую очередь, описывают информационные потребности. Определение требований выполняется в основном силами пользователей под руководством специалистов-разработчиков. Ограничивается масштаб проекта, определяются временные рамки для каждой из последующих фаз. Кроме того, определяется сама возможность реализации данного проекта в установленных рамках финансирования, на данных аппаратных средствах и т.п. Результатом данной фазы должны быть список и приоритетность функций будущей ИС, предварительные функциональные и информационные модели ИС.

На фазе проектирования часть пользователей принимает участие в техническом проектировании системы под руководством специалистов-разработчиков. CASE-средства используются для быстрого получения работающих прототипов приложений. Пользователи, непосредственно взаимодействуя с ними, уточняют и дополняют требования к системе, которые не были выявлены на предыдущей фазе. Более подробно рассматриваются процессы системы. Анализируется и, при необходимости, корректируется функциональная модель. Каждый процесс рассматривается детально. При необходимости для каждого элементарного процесса создается частичный прототип: экран, диалог, отчет, устраняющий неясности или неоднозначности. Определяются требования разграничения доступа к данным. На этой же фазе происходит определение набора необходимой документации.

После детального определения состава процессов оценивается количество функциональных элементов разрабатываемой системы и принимается решение о разделении ИС на подсистемы, поддающиеся реализации одной командой разработчиков за приемлемое для RAD-проектов время - порядка 60 - 90 дней. С использованием CASE-средств проект распределяется между различными командами (делится функциональная модель). Результатом данной фазы должны быть:

- 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, отчетов, диалогов.

Все модели и прототипы должны быть получены с применением тех CASE-средств, которые будут использоваться в дальнейшем при построении системы. Данное требование вызвано тем, что в традиционном подходе при передаче информации о проекте с этапа на этап может произойти фактически неконтролируемое искажение данных. Применение единой среды хранения информации о проекте позволяет избежать этой опасности.

В отличие от традиционного подхода, при котором использовались специфические средства прототипирования, не предназначенные для построения реальных приложений, а прототипы выбрасывались после того, как выполняли задачу устранения неясностей в проекте, в подходе RAD каждый прототип развивается в часть будущей системы. Таким образом, на следующую фазу передается более полная и полезная информация.

На фазе построения выполняется непосредственно сама быстрая разработка приложения. На данной фазе разработчики производят итеративное построение реальной системы на основе полученных в предыдущей фазе моделей, а также требований нефункционального характера. Программный код частично формируется при помощи автоматических генераторов, получающих информацию непосредственно из репозитория CASE-средств. Конечные пользователи на этой фазе 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:

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

Результатом фазы является готовая система, удовлетворяющая всем согласованным требованиям.

На фазе внедрения производится обучение пользователей, организационные изменения и параллельно с внедрением новой системы осуществляется работа с существующей системой (до полного внедрения новой). Так как фаза построения достаточно непродолжительна, планирование и подготовка к внедрению должны начинаться заранее, как правило, на этапе проектирования системы. Приведенная схема разработки ИС не является абсолютной. Возможны различные варианты, зависящие, например, от начальных условий, в которых ведется разработка: разрабатывается совершенно новая система; уже было проведено обследование предприятия и существует модель его деятельности; на предприятии уже существует некоторая ИС, которая может быть использована в качестве начального прототипа или должна быть интегрирована с разрабатываемой.

Следует, однако, отметить, что методология RAD, как и любая другая, не может претендовать на универсальность, она хороша в первую очередь для относительно небольших

проектов, разрабатываемых для конкретного заказчика. Если же разрабатывается типовая система, которая не является законченным продуктом, а представляет собой комплекс типовых компонент, централизованно сопровождаемых, адаптируемых к программно-техническим платформам, СУБД, средствам телекоммуникации, организационно-экономическим особенностям объектов внедрения и интегрируемых с существующими разработками, на первый план выступают такие показатели проекта, как управляемость и качество, которые могут войти в противоречие с простотой и скоростью разработки. Для таких проектов необходимы высокий уровень планирования и жесткая дисциплина проектирования, строгое следование заранее разработанным протоколам и интерфейсам, что снижает скорость разработки.

Методология RAD неприменима для построения сложных расчетных программ, операционных систем или программ управления космическими кораблями, т.е. программ, требующих написания большого объема (сотни тысяч строк) уникального кода.

Не подходят для разработки по методологии RAD приложения, в которых отсутствует ярко выраженная интерфейсная часть, наглядно определяющая логику работы системы (например, приложения реального времени) и приложения, от которых зависит безопасность людей (например, управление самолетом или атомной электростанцией), так как итеративный подход предполагает, что первые несколько версий наверняка не будут полностью работоспособны, что в данном случае исключается.

Оценка размера приложений производится на основе так называемых функциональных элементов (экраны, сообщения, отчеты, файлы и т.п.) Подобная метрика не зависит от языка программирования, на котором ведется разработка. Размер приложения, которое может быть выполнено по методологии RAD, для хорошо отлаженной среды разработки ИС с максимальным повторным использованием программных компонентов, определяется следующим образом:

< 1000 функциональных элементов	один человек
1000-4000 функциональных элементов	одна команда разработчиков
> 4000 функциональных элементов	4000 функциональных элементов на одну команду разработчиков

В качестве итога перечислим основные принципы методологии RAD:

- разработка приложений итерациями;
- необязательность полного завершения работ на каждом из этапов жизненного цикла;
- обязательное вовлечение пользователей в процесс разработки ИС;
- необходимое применение CASE-средств, обеспечивающих целостность проекта;
- применение средств управления конфигурацией, облегчающих внесение изменений в проект и сопровождение готовой системы;
- необходимое использование генераторов кода;
- использование прототипирования, позволяющее полнее выяснить и удовлетворить потребности конечного пользователя;
- тестирование и развитие проекта, осуществляемые одновременно с разработкой;
- ведение разработки немногочисленной хорошо управляемой командой профессионалов;
- грамотное руководство разработкой системы, четкое планирование и контроль выполнения работ

2. Структурный подход к проектированию ИС

2.1. Сущность структурного подхода

Сущность структурного подхода к разработке ИС заключается в ее декомпозиции (разбиении) на автоматизируемые функции: система разбивается на функциональные подсистемы, которые в свою очередь делятся на подфункции, подразделяемые на задачи и так далее. Процесс разбиения продолжается вплоть до конкретных процедур. При этом автоматизируемая система сохраняет целостное представление, в котором все составляющие компоненты взаимоувязаны. При разработке системы "снизу-вверх" от отдельных задач ко всей системе целостность теряется, возникают проблемы при информационной стыковке отдельных компонентов.

Все наиболее распространенные методологии структурного подхода [9,11,12,13] базируются на ряде общих принципов [3]. В качестве двух базовых принципов используются следующие:

- принцип "разделяй и властвуй" - принцип решения сложных проблем путем их разбиения на множество меньших независимых задач, легких для понимания и решения;
- принцип иерархического упорядочивания - принцип организации составных частей проблемы в иерархические древовидные структуры с добавлением новых деталей на каждом уровне.

Выделение двух базовых принципов не означает, что остальные принципы являются второстепенными, поскольку игнорирование любого из них может привести к непредсказуемым последствиям (в том числе и к провалу всего проекта). Основными из этих принципов являются следующие:

- принцип абстрагирования - заключается в выделении существенных аспектов системы и отвлечения от несущественных;
- принцип формализации - заключается в необходимости строгого методического подхода к решению проблемы;
- принцип непротиворечивости - заключается в обоснованности и согласованности элементов;
- принцип структурирования данных - заключается в том, что данные должны быть структурированы и иерархически организованы.

В структурном анализе используются в основном две группы средств, иллюстрирующих функции, выполняемые системой и отношения между данными. Каждой группе средств соответствуют определенные виды моделей (диаграмм), наиболее распространенными среди которых являются следующие:

- SADT (Structured Analysis and Design Technique) модели и соответствующие функциональные диаграммы (подраздел 2.2);
- DFD (Data Flow Diagrams) диаграммы потоков данных (подраздел 2.3);
- ERD (Entity-Relationship Diagrams) диаграммы "сущность-связь" (подраздел 2.4).

На стадии проектирования ИС модели расширяются, уточняются и дополняются диаграммами, отражающими структуру программного обеспечения: архитектуру ПО, структурные схемы программ и диаграммы экранных форм.

Перечисленные модели в совокупности дают полное описание ИС независимо от того, является ли она существующей или вновь разрабатываемой. Состав диаграмм в каждом конкретном случае зависит от необходимой полноты описания системы.

2.2. Методология функционального моделирования SADT

Методология SADT разработана Дугласом Россом и получила дальнейшее развитие в работе [4]. На ее основе разработана, в частности, известная методология IDEF0 (Icam DEFinition), которая является основной частью программы ICAM (Интеграция компьютерных и промышленных технологий), проводимой по инициативе ВВС США.

Методология SADT представляет собой совокупность методов, правил и процедур,

предназначенных для построения функциональной модели объекта какой-либо предметной области. Функциональная модель SADT отображает функциональную структуру объекта, т.е. производимые им действия и связи между этими действиями. Основные элементы этой методологии основываются на следующих концепциях:

- графическое представление блочного моделирования. Графика блоков и дуг SADT-диаграммы отображает функцию в виде блока, а интерфейсы входа/выхода представляются дугами, соответственно входящими в блок и выходящими из него. Взаимодействие блоков друг с другом описываются посредством интерфейсных дуг, выражающих "ограничения", которые в свою очередь определяют, когда и каким образом функции выполняются и управляются;
- строгость и точность. Выполнение правил SADT требует достаточной строгости и точности, не накладывая в то же время чрезмерных ограничений на действия аналитика. Правила SADT включают:
- ограничение количества блоков на каждом уровне декомпозиции (правило 3-6 блоков);
- связность диаграмм (номера блоков);
- уникальность меток и наименований (отсутствие повторяющихся имен);
- синтаксические правила для графики (блоков и дуг);
- разделение входов и управлений (правило определения роли данных).
- отделение организации от функции, т.е. исключение влияния организационной структуры на функциональную модель.

Методология SADT может использоваться для моделирования широкого круга систем и определения требований и функций, а затем для разработки системы, которая удовлетворяет этим требованиям и реализует эти функции. Для уже существующих систем SADT может быть использована для анализа функций, выполняемых системой, а также для указания механизмов, посредством которых они осуществляются.

2.2.1. Состав функциональной модели

Результатом применения методологии SADT является модель, которая состоит из диаграмм, фрагментов текстов и глоссария, имеющих ссылки друг на друга. Диаграммы - главные компоненты модели, все функции ИС и интерфейсы на них представлены как блоки и дуги. Место соединения дуги с блоком определяет тип интерфейса. Управляющая информация входит в блок сверху, в то время как информация, которая подвергается обработке, показана с левой стороны блока, а результаты выхода показаны с правой стороны. Механизм (человек или автоматизированная система), который осуществляет операцию, представляется дугой, входящей в блок снизу (рисунок 2.1).

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

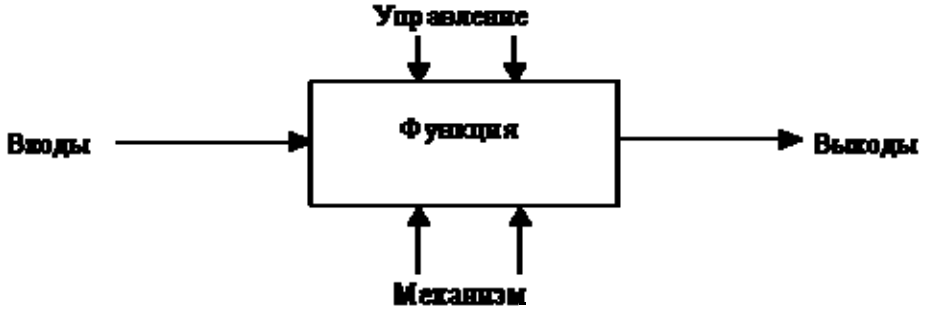


Рис. 2.1. Функциональный блок и интерфейсные дуги

На рисунке 2.2, где приведены четыре диаграммы и их взаимосвязи, показана структура SADT-модели. Каждый компонент модели может быть декомпозирован на другой диаграмме. Каждая диаграмма иллюстрирует "внутреннее строение" блока на родительской диаграмме.

2.2.2. Иерархия диаграмм

Построение SADT-модели начинается с представления всей системы в виде простейшей компоненты - одного блока и дуг, изображающих интерфейсы с функциями вне системы.

Поскольку единственный блок представляет всю систему как единое целое, имя, указанное в блоке, является общим. Это верно и для интерфейсных дуг - они также представляют полный набор внешних интерфейсов системы в целом.

Затем блок, который представляет систему в качестве единого модуля, детализируется на другой диаграмме с помощью нескольких блоков, соединенных интерфейсными дугами. Эти блоки представляют основные подфункции исходной функции. Данная декомпозиция выявляет полный набор подфункций, каждая из которых представлена как блок, границы которого определены интерфейсными дугами. Каждая из этих подфункций может быть декомпозирована подобным образом для более детального представления.

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

Модель SADT представляет собой серию диаграмм с сопроводительной документацией, разбивающих сложный объект на составные части, которые представлены в виде блоков. Детали каждого из основных блоков показаны в виде блоков на других диаграммах. Каждая детальная диаграмма является декомпозицией блока из более общей диаграммы. На каждом шаге декомпозиции более общая диаграмма называется родительской для более детальной диаграммы.

Дуги, входящие в блок и выходящие из него на диаграмме верхнего уровня, являются точно теми же самыми, что и дуги, входящие в диаграмму нижнего уровня и выходящие из нее, потому что блок и диаграмма представляют одну и ту же часть системы.

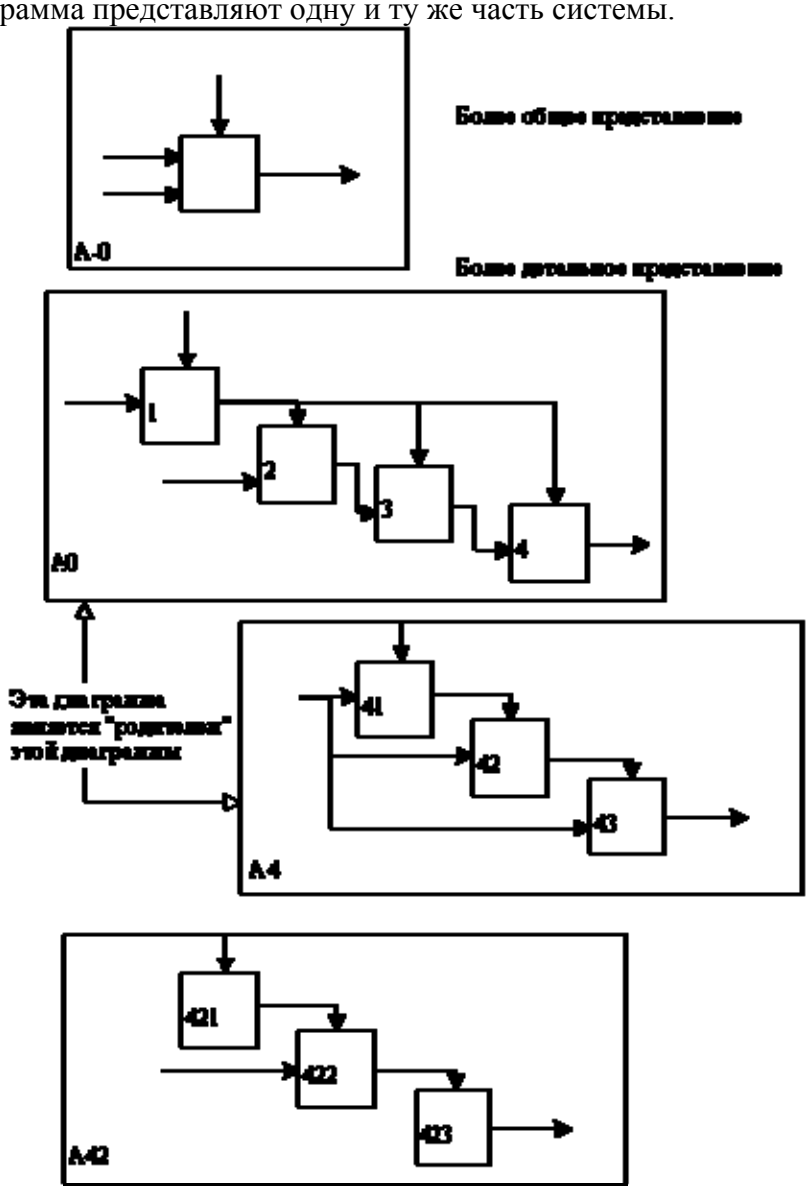


Рис. 2.2. Структура SADT-модели. Декомпозиция диаграмм

На рисунках 2.3 - 2.5 представлены различные варианты выполнения функций и соединения дуг с блоками.

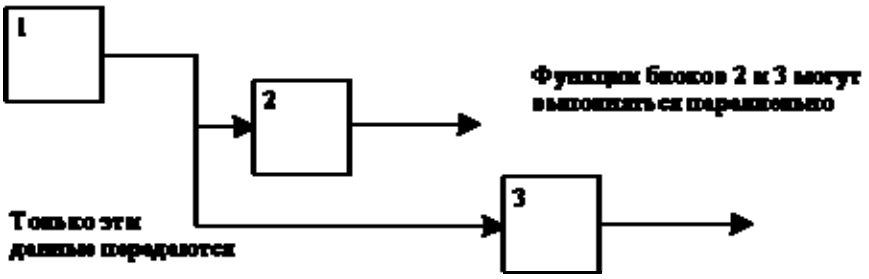


Рис. 2.3. Одновременное выполнение

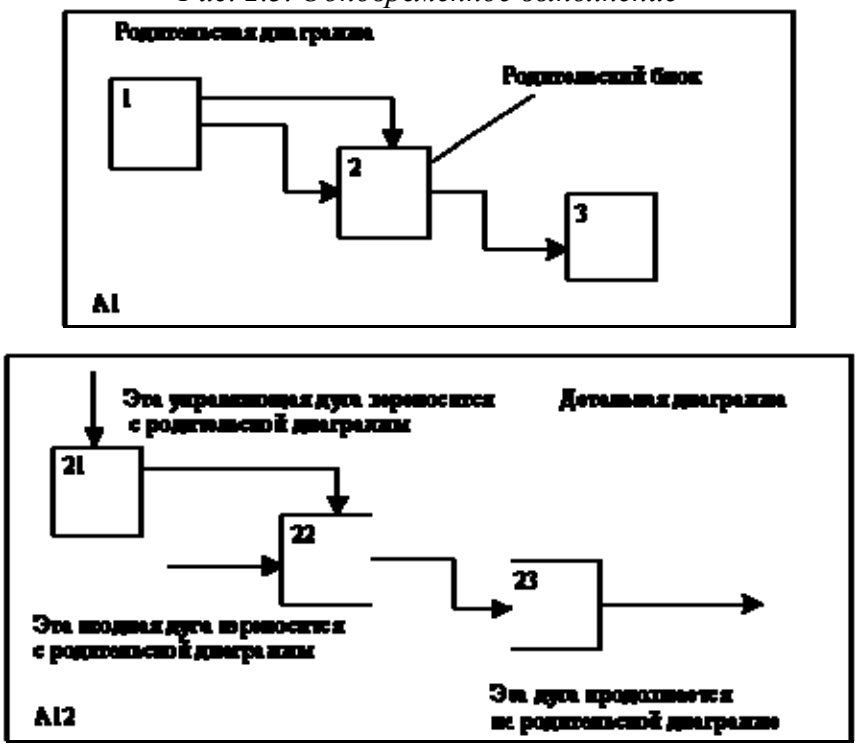


Рис. 2.4. Соответствие должно быть полным и непротиворечивым

Некоторые дуги присоединены к блокам диаграммы обоими концами, у других же один конец остается неприсоединенным. Неприсоединенные дуги соответствуют входам, управлениям и выходам родительского блока. Источник или получатель этих пограничных дуг может быть обнаружен только на родительской диаграмме. Неприсоединенные концы должны соответствовать дугам на исходной диаграмме. Все граничные дуги должны продолжаться на родительской диаграмме, чтобы она была полной и непротиворечивой.

На SADT-диаграммах не указаны явно ни последовательность, ни время. Обратные связи, итерации, продолжающиеся процессы и перекрывающиеся (по времени) функции могут быть изображены с помощью дуг. Обратные связи могут выступать в виде комментариев, замечаний, исправлений и т.д. (рисунок 2.5).

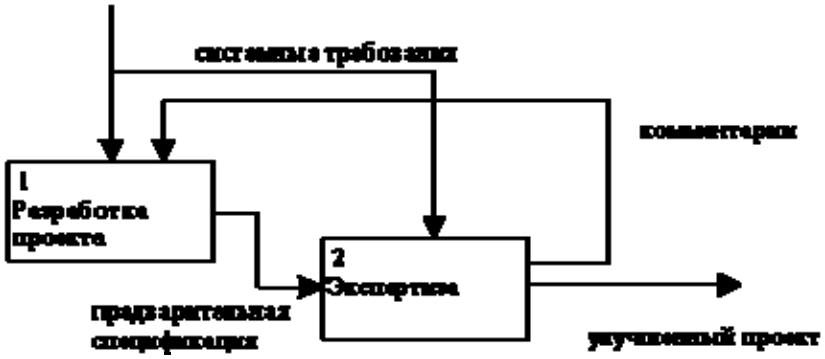


Рис. 2.5. Пример обратной связи

Как было отмечено, механизмы (дуги с нижней стороны) показывают средства, с помощью которых осуществляется выполнение функций. Механизм может быть человеком, компьютером или любым другим устройством, которое помогает выполнять данную функцию (рисунок 2.6).

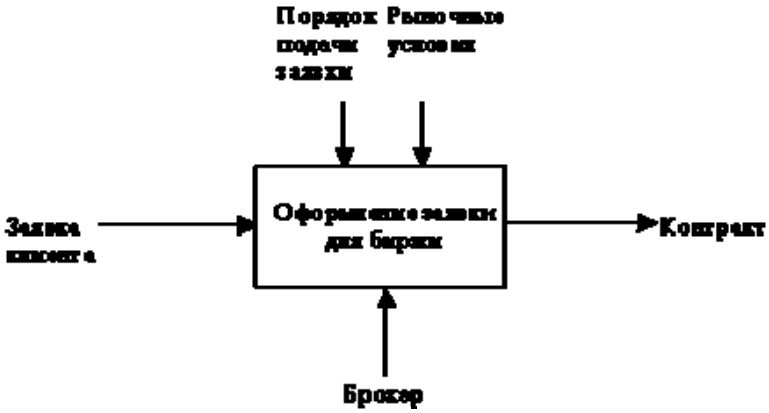


Рис. 2.6. Пример механизма

Каждый блок на диаграмме имеет свой номер. Блок любой диаграммы может быть далее описан диаграммой нижнего уровня, которая, в свою очередь, может быть далее детализирована с помощью необходимого числа диаграмм. Таким образом, формируется иерархия диаграмм.

Для того, чтобы указать положение любой диаграммы или блока в иерархии, используются номера диаграмм. Например, A21 является диаграммой, которая детализирует блок 1 на диаграмме A2. Аналогично, A2 детализирует блок 2 на диаграмме A0, которая является самой верхней диаграммой модели. На рисунке 2.7 показано типичное дерево диаграмм.

Рис. 2.7. Иерархия диаграмм

2.2.3. Типы связей между функциями

Одним из важных моментов при проектировании ИС с помощью методологии SADT является точная согласованность типов связей между функциями. Различают по крайней мере семь типов связывания:

Тип связи	Относительная значимость
Случайная	0
Логическая	1
Временная	2
Процедурная	3

Коммуникационная	4
Последовательная	5
Функциональная	6

Ниже каждый тип связи кратко определен и проиллюстрирован с помощью типичного примера из SADT.

(0) Тип случайной связности: наименее желательный.

Случайная связность возникает, когда конкретная связь между функциями мала или полностью отсутствует. Это относится к ситуации, когда имена данных на SADT-дугах в одной диаграмме имеют малую связь друг с другом. Крайний вариант этого случая показан на рисунке 2.8.

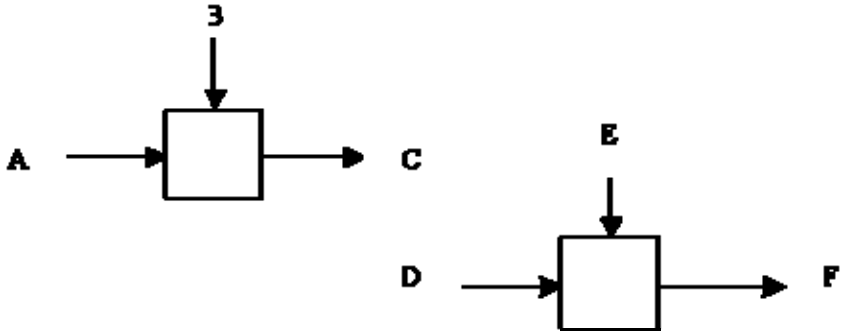


Рис. 2.8. Случайная связность

(1) Тип логической связности. Логическое связывание происходит тогда, когда данные и функции собираются вместе вследствие того, что они попадают в общий класс или набор элементов, но необходимых функциональных отношений между ними не обнаруживается.

(2) Тип временной связности. Связанные по времени элементы возникают вследствие того, что они представляют функции, связанные во времени, когда данные используются одновременно или функции включаются параллельно, а не последовательно.

(3) Тип процедурной связности. Процедурно-связанные элементы появляются сгруппированными вместе вследствие того, что они выполняются в течение одной и той же части цикла или процесса. Пример процедурно-связанной диаграммы приведен на рисунке 2.9.

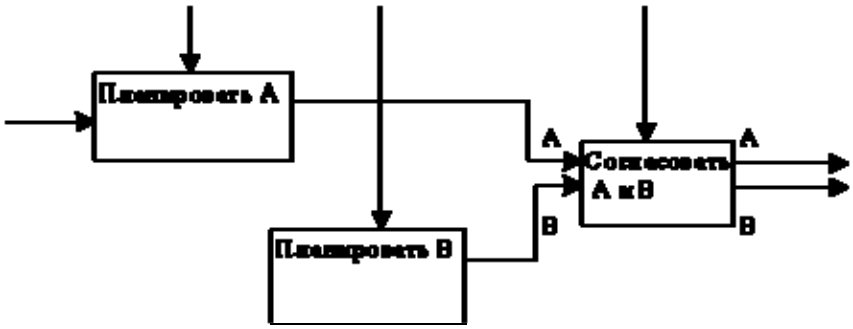


Рис. 2.9. Процедурная связность

(4) Тип коммуникационной связности. Диаграммы демонстрируют коммуникационные связи, когда блоки группируются вследствие того, что они используют одни и те же входные данные и/или производят одни и те же выходные данные (рисунок 2.10).

(5) Тип последовательной связности. На диаграммах, имеющих последовательные связи, выход одной функции служит входными данными для следующей функции. Связь между элементами на диаграмме является более тесной, чем на рассмотренных выше уровнях связок, поскольку моделируются причинно-следственные зависимости (рисунок 2.11).

(6) Тип функциональной связности. Диаграмма отражает полную функциональную связность, при наличии полной зависимости одной функции от другой. Диаграмма, которая является чисто функциональной, не содержит чужеродных элементов, относящихся к последовательному или более слабому типу связности. Одним из способов определения

функционально-связанных диаграмм является рассмотрение двух блоков, связанных через управляющие дуги, как показано на рисунке 2.12.

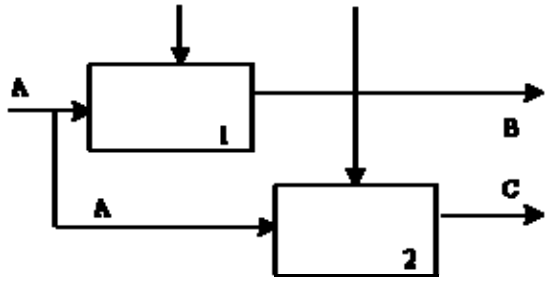


Рис. 2.10. Коммуникационная связность

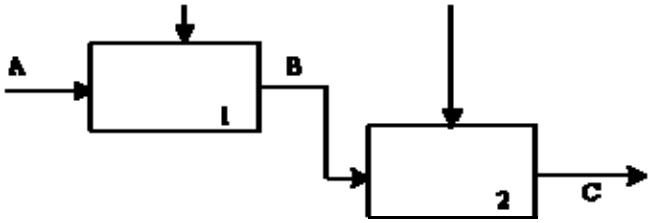


Рис. 2.11. Последовательная связность

В математических терминах необходимое условие для простейшего типа функциональной связности, показанной на рисунке 2.12, имеет следующий вид:

```
C = g(B) = g(f(A))
```

Ниже в таблице представлены все типы связей, рассмотренные выше. Важно отметить, что уровни 4-6 устанавливают типы связностей, которые разработчики считают важнейшими для получения диаграмм хорошего качества.

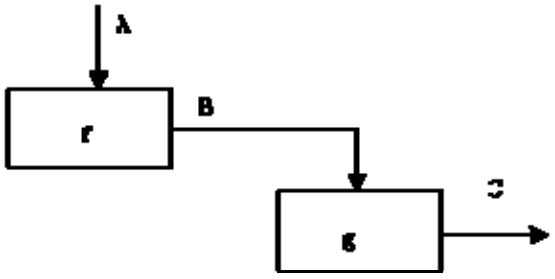


Рис. 2.12. Функциональная связность

Значимость	Тип связности	Для функций	Для данных
0	Случайная	Случайная	Случайная
1	Логическая	Функции одного и того же множества или типа (например, "редактировать все входы")	Данные одного и того же множества или типа
2	Временная	Функции одного и того же периода времени (например, "операции инициализации")	Данные, используемые в каком-либо временном интервале
3	Процедурная	Функции, работающие в одной и той же фазе или итерации (например, "первый проход компилятора")	Данные, используемые во время одной и той же фазы или итерации
4	Коммуникационнная	Функции, использующие одни и те же данные	Данные, на которые воздействует одна и та же деятельность
5	Последовательная	Функции, выполняющие последовательные	Данные, преобразуемые последовательными

		преобразования одних и тех же данных	функциями
6	Функциональная	Функции, объединяемые для выполнения одной функции	Данные, связанные с одной функцией

2.3. Моделирование потоков данных (процессов)

В основе данной методологии (методологии Gane/Sarson [11]) лежит построение модели анализируемой ИС - проектируемой или реально существующей. В соответствии с методологией модель системы определяется как иерархия диаграмм потоков данных (ДПД или DFD), описывающих асинхронный процесс преобразования информации от ее ввода в систему до выдачи пользователю. Диаграммы верхних уровней иерархии (контекстные диаграммы) определяют основные процессы или подсистемы ИС с внешними входами и выходами. Они детализируются при помощи диаграмм нижнего уровня. Такая декомпозиция продолжается, создавая многоуровневую иерархию диаграмм, до тех пор, пока не будет достигнут такой уровень декомпозиции, на котором процесс становятся элементарными и детализировать их далее невозможно.

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

- внешние сущности;
- системы/подсистемы;
- процессы;
- накопители данных;
- потоки данных.

2.3.1. Внешние сущности

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

Внешняя сущность обозначается квадратом (рисунок 2.13), расположенным как бы "над" диаграммой и бросающим на нее тень, для того, чтобы можно было выделить этот символ среди других обозначений:

Рис. 2.13. Внешняя сущность

2.3.2. Системы и подсистемы

При построении модели сложной ИС она может быть представлена в самом общем виде на так называемой контекстной диаграмме в виде одной системы как единого целого, либо может быть декомпозирована на ряд подсистем.

Подсистема (или система) на контекстной диаграмме изображается следующим образом (рисунок 2.14).

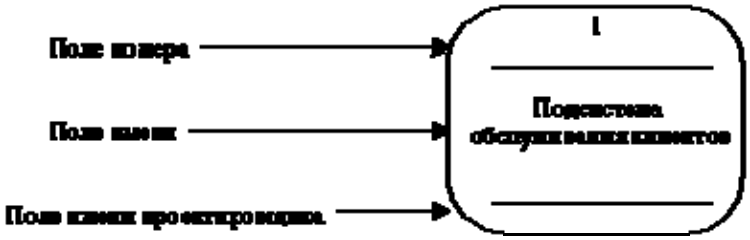


Рис. 2.14. Подсистема

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

2.3.3. Процессы

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

Процесс на диаграмме потоков данных изображается, как показано на рисунке 2.15.

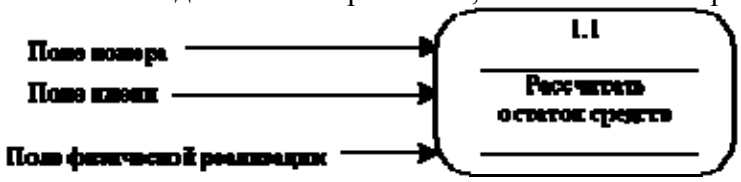


Рис. 2.15. Процесс

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

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

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

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

2.3.4. Накопители данных

Накопитель данных представляет собой абстрактное устройство для хранения информации, которую можно в любой момент поместить в накопитель и через некоторое время извлечь, причем способы помещения и извлечения могут быть любыми.

Накопитель данных может быть реализован физически в виде микрофиши, ящика в картотеке, таблицы в оперативной памяти, файла на магнитном носителе и т.д. Накопитель данных на диаграмме потоков данных изображается, как показано на рисунке 2.16.

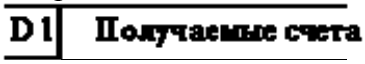


Рис. 2.16. Накопитель данных

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

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

2.3.5. Потоки данных

Поток данных определяет информацию, передаваемую через некоторое соединение от источника к приемнику. Реальный поток данных может быть информацией, передаваемой по кабелю между двумя устройствами, пересылаемыми по почте письмами, магнитными лентами или дискетами, переносимыми с одного компьютера на другой и т.д.

Поток данных на диаграмме изображается линией, оканчивающейся стрелкой, которая показывает направление потока (рисунок 2.17). Каждый поток данных имеет имя, отражающее его содержание.

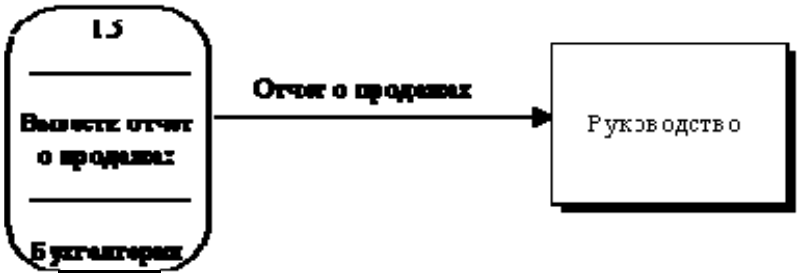


Рис. 2.17. Поток данных

2.3.6. Построение иерархии диаграмм потоков данных

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

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

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

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

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

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

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

Для каждой подсистемы, присутствующей на контекстных диаграммах, выполняется ее детализация при помощи ДПД. Каждый процесс на ДПД, в свою очередь, может быть детализирован при помощи ДПД или миниспецификации. При детализации должны выполняться следующие правила:

- правило балансировки - означает, что при детализации подсистемы или процесса детализирующая диаграмма в качестве внешних источников/приемников данных может

иметь только те компоненты (подсистемы, процессы, внешние сущности, накопители данных), с которыми имеет информационную связь детализируемая подсистема или процесс на родительской диаграмме;

- правило нумерации - означает, что при детализации процессов должна поддерживаться их иерархическая нумерация. Например, процессы, детализирующие процесс с номером 12, получают номера 12.1, 12.2, 12.3 и т.д.

Миниспецификация (описание логики процесса) должна формулировать его основные функции таким образом, чтобы в дальнейшем специалист, выполняющий реализацию проекта, смог выполнить их или разработать соответствующую программу.

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

- наличия у процесса относительно небольшого количества входных и выходных потоков данных (2-3 потока);
- возможности описания преобразования данных процессом в виде последовательного алгоритма;
- выполнения процессом единственной логической функции преобразования входной информации в выходную;
- возможности описания логики процесса при помощи миниспецификации небольшого объема (не более 20-30 строк).

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

После построения законченной модели системы ее необходимо верифицировать (проверить на полноту и согласованность). В полной модели все ее объекты (подсистемы, процессы, потоки данных) должны быть подробно описаны и детализированы. Выявленные недетализированные объекты следует детализировать, вернувшись на предыдущие шаги разработки. В согласованной модели для всех потоков данных и накопителей данных должно выполняться правило сохранения информации: все поступающие куда-либо данные должны быть считаны, а все считываемые данные должны быть записаны

2.4. Моделирование данных

2.4.1. Case-метод Баркера

Цель моделирования данных состоит в обеспечении разработчика ИС концептуальной схемой базы данных в форме одной модели или нескольких локальных моделей, которые относительно легко могут быть отображены в любую систему баз данных.

Наиболее распространенным средством моделирования данных являются диаграммы "сущность-связь" (ERD). С их помощью определяются важные для предметной области объекты (сущности), их свойства (атрибуты) и отношения друг с другом (связи). ERD непосредственно используются для проектирования реляционных баз данных.

Нотация ERD была впервые введена П. Ченом (Chen) и получила дальнейшее развитие в работах Баркера [8]. Метод Баркера будет излагаться на примере моделирования деятельности компании по торговле автомобилями. Ниже приведены выдержки из интервью, проведенного с персоналом компании.

Главный менеджер: одна из основных обязанностей - содержание автомобильного имущества. Он должен знать, сколько заплачено за машины и каковы накладные расходы. Обладая этой информацией, он может установить нижнюю цену, за которую мог бы продать данный экземпляр. Кроме того, он несет ответственность за продавцов и ему нужно знать, кто что продает и сколько машин продал каждый из них.

Продавец: ему нужно знать, какую цену запрашивать и какова нижняя цена, за которую можно совершить сделку. Кроме того, ему нужна основная информация о машинах: год выпуска, марка, модель и т.п.

Администратор: его задача сводится к составлению контрактов, для чего нужна информация о покупателе, автомашине и продавце, поскольку именно контракты приносят продавцам вознаграждения за продажи.

Первый шаг моделирования - извлечение информации из интервью и выделение сущностей.

Сущность (Entity) - реальный либо воображаемый объект, имеющий существенное значение для рассматриваемой предметной области, информация о котором подлежит хранению (рисунок 2.18).

Рис. 2.18. Графическое изображение сущности

Каждая сущность должна обладать уникальным идентификатором. Каждый экземпляр сущности должен однозначно идентифицироваться и отличаться от всех других экземпляров данного типа сущности. Каждая сущность должна обладать некоторыми свойствами:

- каждая сущность должна иметь уникальное имя, и к одному и тому же имени должна всегда применяться одна и та же интерпретация. Одна и та же интерпретация не может применяться к различным именам, если только они не являются псевдонимами;
- сущность обладает одним или несколькими атрибутами, которые либо принадлежат сущности, либо наследуются через связь;
- сущность обладает одним или несколькими атрибутами, которые однозначно идентифицируют каждый экземпляр сущности;
- каждая сущность может обладать любым количеством связей с другими сущностями модели.

Обращаясь к приведенным выше выдержкам из интервью, видно, что сущности, которые могут быть идентифицированы с главным менеджером - это автомашины и продавцы. Продавцу важны автомашины и связанные с их продажей данные. Для администратора важны покупатели, автомашины, продавцы и контракты. Исходя из этого, выделяются 4 сущности (автомашина, продавец, покупатель, контракт), которые изображаются на диаграмме следующим образом (рисунок 2.19).

Рис. 2.19.

Следующим шагом моделирования является идентификация связей.

Связь (Relationship) - поименованная ассоциация между двумя сущностями, значимая для рассматриваемой предметной области. Связь - это ассоциация между сущностями, при которой, как правило, каждый экземпляр одной сущности, называемой родительской сущностью, ассоциирован с произвольным (в том числе нулевым) количеством экземпляров второй сущности, называемой сущностью-потомком, а каждый экземпляр сущности-потомка ассоциирован в точности с одним экземпляром сущности-родителя. Таким образом, экземпляр сущности-потомка может существовать только при существовании сущности родителя.

Связи может даваться имя, выражаемое грамматическим оборотом глагола и помещаемое возле линии связи. Имя каждой связи между двумя данными сущностями должно быть уникальным, но имена связей в модели не обязаны быть уникальными. Имя связи всегда

формируется с точки зрения родителя, так что предложение может быть образовано соединением имени сущности-родителя, имени связи, выражения степени и имени сущности-потомка.

Например, связь продавца с контрактом может быть выражена следующим образом:

- продавец может получить вознаграждение за 1 или более контрактов;
- контракт должен быть инициирован ровно одним продавцом.

Степень связи и обязательность графически изображаются следующим образом (рисунок 2.20).

Рис. 2.20.

Таким образом, 2 предложения, описывающие связь продавца с контрактом, графически будут выражены следующим образом (рисунок 2.21).

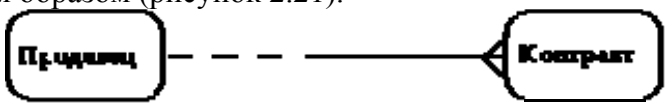


Рис. 2.21.

Описав также связи остальных сущностей, получим следующую схему (рисунок 2.22).

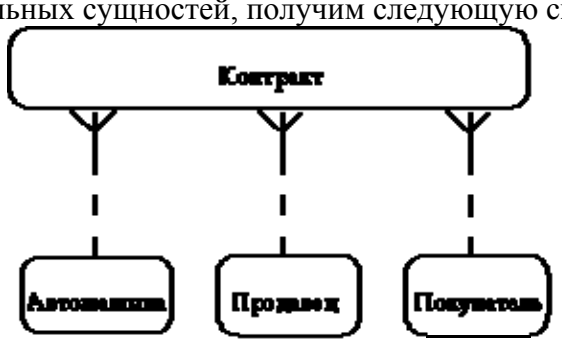


Рис. 2.22.

Последним шагом моделирования является идентификация атрибутов.

Атрибут - любая характеристика сущности, значимая для рассматриваемой предметной области и предназначенная для квалификации, идентификации, классификации, количественной характеристики или выражения состояния сущности. Атрибут представляет тип характеристик или свойств, ассоциированных со множеством реальных или абстрактных объектов (людей, мест, событий, состояний, идей, пар предметов и т.д.). Экземпляр атрибута - это определенная характеристика отдельного элемента множества. Экземпляр атрибута определяется типом характеристики и ее значением, называемым значением атрибута. В ER-модели атрибуты ассоциируются с конкретными сущностями. Таким образом, экземпляр сущности должен обладать единственным определенным значением для ассоциированного атрибута.

Атрибут может быть либо обязательным, либо необязательным (рисунок 2.23). Обязательность означает, что атрибут не может принимать неопределенных значений (null values). Атрибут может быть либо описательным (т.е. обычным дескриптором сущности), либо входить в состав уникального идентификатора (первичного ключа).

Уникальный идентификатор - это атрибут или совокупность атрибутов и/или связей, предназначенная для уникальной идентификации каждого экземпляра данного типа сущности. В случае полной идентификации каждый экземпляр данного типа сущности полностью идентифицируется своими собственными ключевыми атрибутами, в противном случае в его идентификации участвуют также атрибуты другой сущности-родителя (рисунок 2.24).

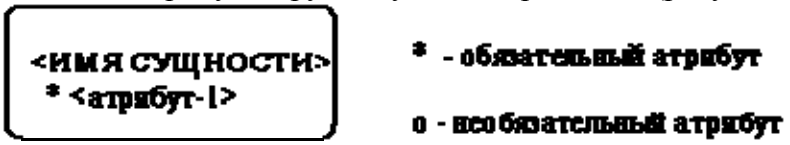


Рис. 2.23.

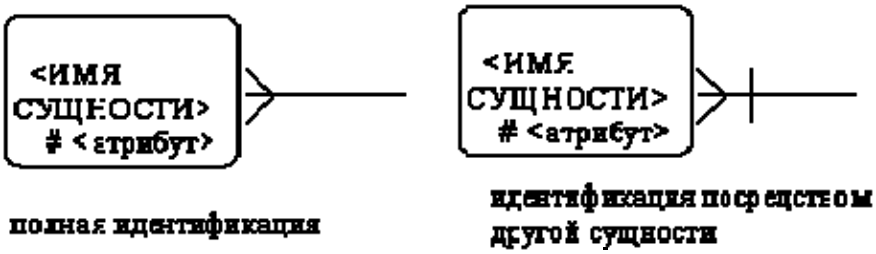


Рис. 2.24.

Каждый атрибут идентифицируется уникальным именем, выражаемым грамматическим оборотом существительного, описывающим представляемую атрибутом характеристику. Атрибуты изображаются в виде списка имен внутри блока ассоциированной сущности, причем каждый атрибут занимает отдельную строку. Атрибуты, определяющие первичный ключ, размещаются наверху списка и выделяются знаком "#".

Каждая сущность должна обладать хотя бы одним возможным ключом. Возможный ключ сущности - это один или несколько атрибутов, чьи значения однозначно определяют каждый экземпляр сущности. При существовании нескольких возможных ключей один из них обозначается в качестве первичного ключа, а остальные - как альтернативные ключи.

С учетом имеющейся информации дополним построенную ранее диаграмму (рисунок 2.25).

Помимо перечисленных основных конструкций модель данных может содержать ряд дополнительных.

Подтипы и супертипы: одна сущность является обобщающим понятием для группы подобных сущностей (рисунок 2.26).

Взаимно исключающие связи: каждый экземпляр сущности участвует только в одной связи из группы взаимно исключающих связей (рисунок 2.27).

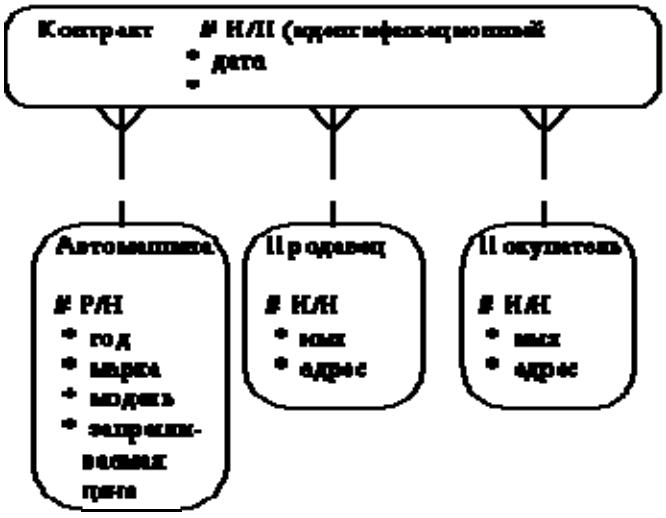


Рис. 2.25.

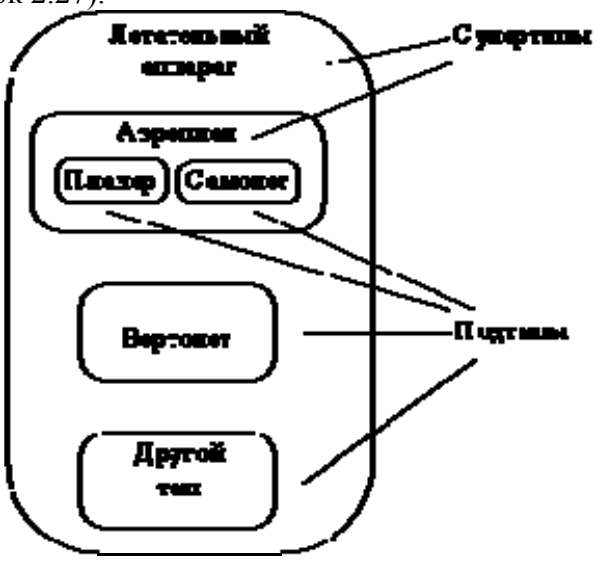


Рис. 2.26. Подтипы и супертипы

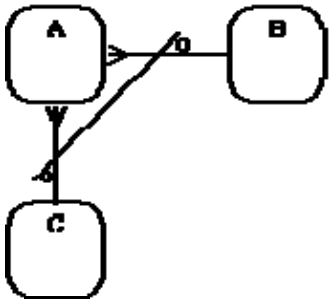


Рис. 2.27. Взаимно исключающие связи

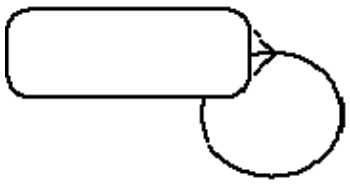


Рис. 2.28. Рекурсивная связь

Рекурсивная связь: сущность может быть связана сама с собой (рисунок 2.28).

Неперемещаемые (non-transferrable) связи: экземпляр сущности не может быть перенесен из одного экземпляра связи в другой (рисунок 2.29).

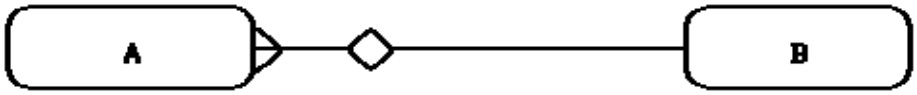


Рис. 2.29. Неперемещаемая связь

2.4.2. Методология IDEF1

Метод IDEF1, разработанный Т.Рэмей (T.Ramey), также основан на подходе П.Чена и позволяет построить модель данных, эквивалентную реляционной модели в третьей нормальной форме. В настоящее время на основе совершенствования методологии IDEF1 создана ее новая версия - методология IDEF1X. IDEF1X разработана с учетом таких требований, как простота изучения и возможность автоматизации. IDEF1X-диаграммы используются рядом распространенных CASE-средств (в частности, ERwin, Design/IDEF).

Сущность в методологии IDEF1X является независимой от идентификаторов или просто независимой, если каждый экземпляр сущности может быть однозначно идентифицирован без определения его отношений с другими сущностями. Сущность называется зависимой от идентификаторов или просто зависимой, если однозначная идентификация экземпляра сущности зависит от его отношения к другой сущности (рисунок 2.30).

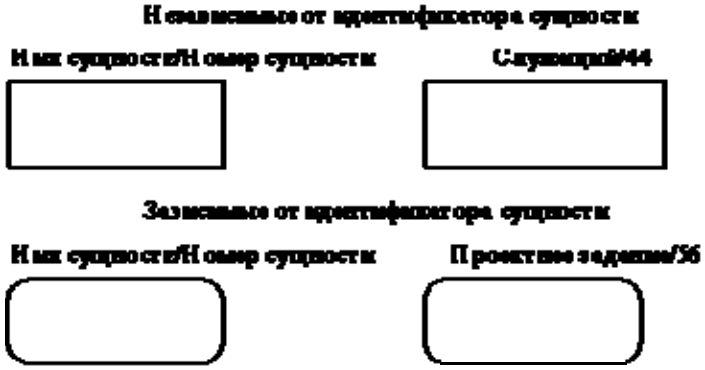


Рис. 2.30. Сущности

Каждой сущности присваивается уникальное имя и номер, разделяемые косой чертой "/" и помещаемые над блоком.

Связь может дополнительно определяться с помощью указания степени или мощности (количества экземпляров сущности-потомка, которое может существовать для каждого экземпляра сущности-родителя). В IDEF1X могут быть выражены следующие мощности связей:

- каждый экземпляр сущности-родителя может иметь ноль, один или более связанных с ним экземпляров сущности-потомка;
- каждый экземпляр сущности-родителя должен иметь не менее одного связанного с ним экземпляра сущности-потомка;
- каждый экземпляр сущности-родителя должен иметь не более одного связанного с ним экземпляра сущности-потомка;
- каждый экземпляр сущности-родителя связан с некоторым фиксированным числом экземпляров сущности-потомка.

Если экземпляр сущности-потомка однозначно определяется своей связью с сущностью-родителем, то связь называется идентифицирующей, в противном случае - неидентифицирующей.

Связь изображается линией, проводимой между сущностью-родителем и сущностью-потомком с точкой на конце линии у сущности-потомка. Мощность связи обозначается как показано на рис. 2.31 (мощность по умолчанию - N).

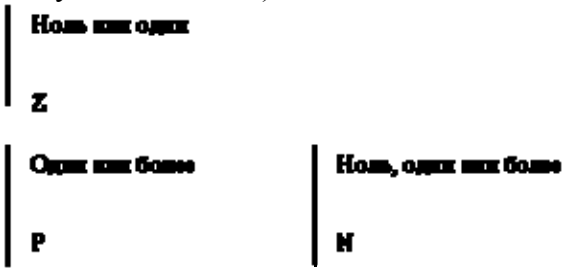


Рис. 2.31. Мощность связи

Идентифицирующая связь между сущностью-родителем и сущностью-потомком изображается сплошной линией (рисунок 2.32). Сущность-потомок в идентифицирующей связи является зависимой от идентификатора сущностью. Сущность-родитель в идентифицирующей связи может быть как независимой, так и зависимой от идентификатора сущностью (это определяется ее связями с другими сущностями).

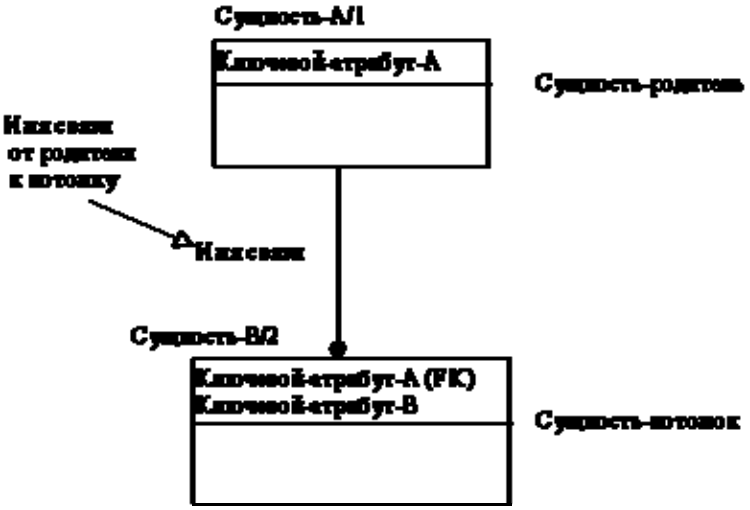


Рис. 2.32. Идентифицирующая связь

Пунктирная линия изображает неидентифицирующую связь (рисунок 2.33). Сущность-потомок в неидентифицирующей связи будет независимой от идентификатора, если она не является также сущностью-потомком в какой-либо идентифицирующей связи.

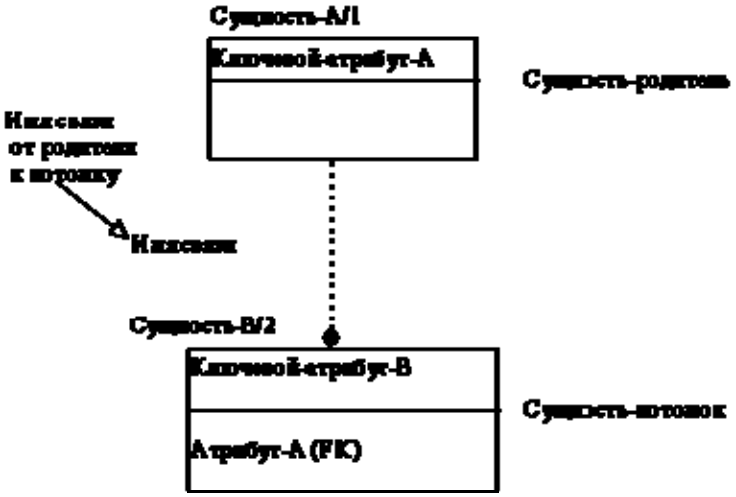


Рис. 2.33. Неидентифицирующая связь

Атрибуты изображаются в виде списка имен внутри блока сущности. Атрибуты, определяющие первичный ключ, размещаются наверху списка и отделяются от других атрибутов горизонтальной чертой (рисунок 2.34).

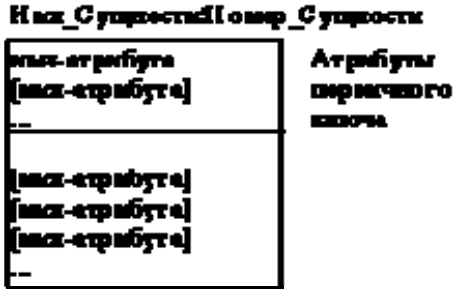


Рис. 2.34. Атрибуты и первичные ключи

Сущности могут иметь также внешние ключи (Foreign Key), которые могут использоваться в качестве части или целого первичного ключа или неключевого атрибута. Внешний ключ изображается с помощью помещения внутрь блока сущности имен атрибутов, после которых следуют буквы FK в скобках (рисунок 2.35).

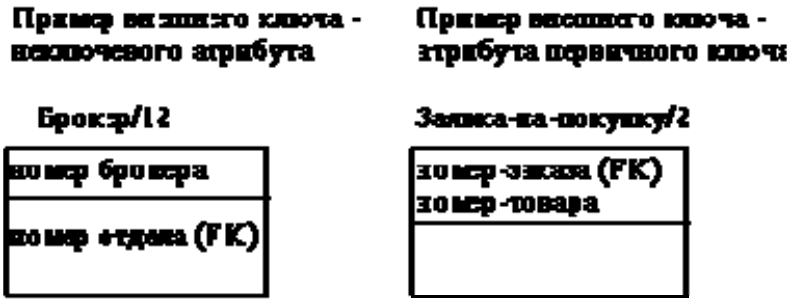


Рис. 2.35. Примеры внешних ключей

2.4.3. Подход, используемый в CASE-средстве Vantage Team Builder

В CASE-средстве Vantage Team Builder (Westmount I-CASE) [14] используется один из вариантов нотации П. Чена. На ER-диаграммах сущность обозначается прямоугольником, содержащим имя сущности (рисунок 2.36), а связь - ромбом, связанным линией с каждой из взаимодействующих сущностей. Числа над линиями означают степень связи.

Рис. 2.36. Обозначение сущностей и связей

Связи являются многонаправленными и могут иметь атрибуты (за исключением ключевых). Выделяют два вида связей:

- необязательная связь (optional);
- слабая связь (weak).

В **необязательной связи** (рисунок 2.37) могут участвовать не все экземпляры сущности.

Рис. 2.37. Необязательная связь

В отличие от необязательной связи в **полной (total)** связи участвуют все экземпляры хотя бы одной из сущностей. Это означает, что экземпляры такой связи существуют только при условии существования экземпляров другой сущности. Полная связь может иметь один из 4-х видов: обязательная связь, слабая связь, связь "супертип-подтип" и ассоциативная связь.

Обязательная (mandatory) связь описывает связь между "независимой" и "зависимой" сущностями. Все экземпляры зависимой ("обязательной") сущности могут существовать только при наличии экземпляров независимой ("необязательной") сущности, т.е. экземпляр "обязательной" сущности может существовать только при условии существования определенного экземпляра "необязательной" сущности.

В примере (рисунок 2.38) подразумевается, что каждый автомобиль имеет по крайней мере одного водителя, но не каждый служащий управляет машиной.

Рис. 2.38. Обязательная связь

В **слабой связи** существование одной из сущностей, принадлежащей некоторому множеству ("слабой") зависит от существования определенной сущности, принадлежащей другому множеству ("сильной"), т.е. экземпляр "слабой" сущности может быть идентифицирован только посредством экземпляра "сильной" сущности. Ключ "сильной" сущности является частью составного ключа "слабой" сущности.

Слабая связь всегда является бинарной и подразумевает обязательную связь для "слабой" сущности. Сущность может быть "слабой" в одной связи и "сильной" в другой, но не может быть "слабой" более, чем в одной связи. Слабая связь может не иметь атрибутов.

Пример на рисунке 2.39: ключ (номер) строки в документе может не быть уникальным и должен быть дополнен ключом документа.

Рис. 2.39. Слабая связь

Связь "супертип-подтип" изображена на рисунке 2.40. Общие характеристики (атрибуты) типа определяются в сущности-супертипе, сущность-подтип наследует все характеристики супертипа. Экземпляр подтипа существует только при условии существования определенного экземпляра супертипа. Подтип не может иметь ключа (он импортирует ключ из супертипа). Сущность, являющаяся супертипом в одной связи, может быть подтипом в другой связи. Связь супертипа не может иметь атрибутов.

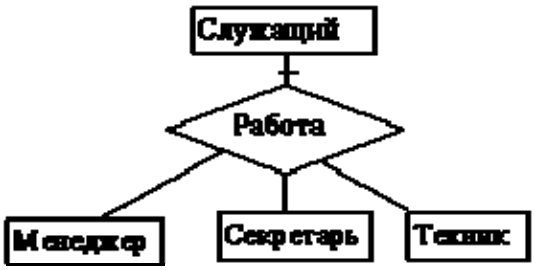


Рис. 2.40. Связь "супертип-подтип"

В **ассоциативной связи** каждый экземпляр связи (ассоциативный объект) может существовать только при условии существования определенных экземпляров каждой из взаимосвязанных сущностей. Ассоциативный объект - объект, являющийся одновременно сущностью и связью. Ассоциативная связь - это связь между несколькими "независимыми" сущностями и одной "зависимой" сущностью. Связь между независимыми сущностями имеет атрибуты, которые определяются в зависимой сущности. Таким образом, зависимая сущность определяется в терминах атрибутов связи между остальными сущностями.

В примере на рисунке 2.41 самолет выполняет посадку на взлетную полосу в заданное время при определенной скорости и направлении ветра. Поскольку эти характеристики применимы только к конкретной посадке, они являются атрибутами посадки, а не самолета или взлетной полосы. Пилот, выполняющий посадку, связан гораздо сильнее с конкретной посадкой, чем с самолетом или взлетной полосой.

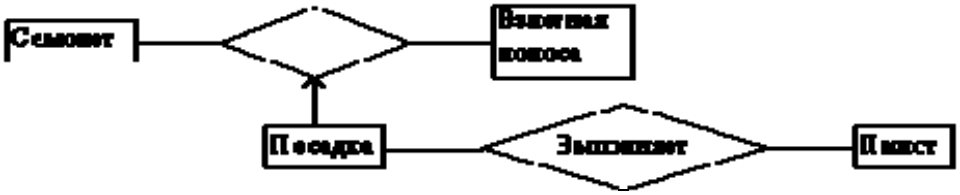


Рис. 2.41. Ассоциативная связь

Первичный ключ каждого типа сущности помечается звездочкой (*).

ER-диаграмма должна подчиняться следующим правилам:

- каждая сущность, каждый атрибут и каждая связь должны иметь имя (связь супертипа или ассоциативная связь может не иметь имени);
- имя сущности должно быть уникально в рамках модели данных;
- имя атрибута должно быть уникально в рамках сущности;
- имя связи должно быть уникально, если для нее генерируется таблица БД;
- каждый атрибут должен иметь определение типа данных;
- сущность в необязательной связи должна иметь ключевой атрибут. То же самое относится к сильной сущности в слабой связи, супертипу в связи "супертип-подтип" и необязательной сущности в обязательной (полной) связи;
- подтип в связи "супертип-подтип" не может иметь ключевой атрибут;
- в ассоциативной или слабой связи может быть только одна ассоциативная (слабая) сущность;
- связь не может быть одновременно обязательной, "супертип-подтип" или ассоциативной.

2.5. Пример использования структурного подхода

2.5.1. Описание предметной области

В данном примере используется методология Yourdon [12], реализованная в CASE-средстве Vantage Team Builder [14].

В качестве предметной области используется описание работы видеобиблиотеки, которая получает запросы на фильмы от клиентов и ленты, возвращаемые клиентами. Запросы рассматриваются администрацией видеобиблиотеки с использованием информации о клиентах, фильмах и лентах. При этом проверяется и обновляется список арендованных лент, а также проверяются записи о членстве в библиотеке. Администрация контролирует также возвраты лент, используя информацию о фильмах, лентах и список арендованных лент, который обновляется. Обработка запросов на фильмы и возвратов лент включает следующие действия: если клиент не является членом библиотеки, он не имеет права на аренду. Если требуемый фильм имеется в наличии, администрация информирует клиента об арендной плате. Однако, если клиент просрочил срок возврата имеющихся у него лент, ему не разрешается брать новые фильмы. Когда лента возвращается, администрация рассчитывает арендную плату плюс пени за несвоевременный возврат.

Видеобиблиотека получает новые ленты от своих поставщиков. Когда новые ленты поступают в библиотеку, необходимая информация о них фиксируется. Информация о членстве в библиотеке содержится отдельно от записей об аренде лент.

Администрация библиотеки регулярно готовит отчеты за определенный период времени о членах библиотеки, поставщиках лент, выдаче определенных лент и лентах, приобретенных библиотекой.

2.5.2. Организация проекта

Весь проект разделяется на 4 фазы: анализ, глобальное проектирование (проектирование архитектуры системы), детальное проектирование и реализация (программирование).

На фазе анализа строится модель среды (Environmental Model). Построение модели среды включает:

- анализ поведения системы (определение назначения ИС, построение начальной контекстной диаграммы потоков данных (DFD) и формирование матрицы списка событий (ELM), построение контекстных диаграмм);
- анализ данных (определение состава потоков данных и построение диаграмм структур данных (DSD), конструирование глобальной модели данных в виде ER-диаграммы).

Назначение ИС определяет соглашение между проектировщиками и заказчиками относительно назначения будущей ИС, общее описание ИС для самих проектировщиков и границы ИС. Назначение фиксируется как текстовый комментарий в "нулевом" процессе контекстной диаграммы.

Например, в данном случае назначение ИС формулируется следующим образом: ведение базы данных о членах библиотеки, фильмах, аренде и поставщиках. При этом руководство библиотеки должно иметь возможность получать различные виды отчетов для выполнения своих задач.

Перед построением контекстной DFD необходимо проанализировать внешние события (внешние объекты), оказывающие влияние на функционирование библиотеки. Эти объекты взаимодействуют с ИС путем информационного обмена с ней.

Из описания предметной области следует, что в процессе работы библиотеки участвуют следующие группы людей: клиенты, поставщики и руководство. Эти группы являются внешними объектами. Они не только взаимодействуют с системой, но также определяют ее границы и изображаются на начальной контекстной DFD как терминаторы (внешние сущности).

Начальная контекстная диаграмма изображена на рисунке 2.42. В отличие от нотации Gane/Sarson внешние сущности обозначаются обычными прямоугольниками, а процессы - окружностями.

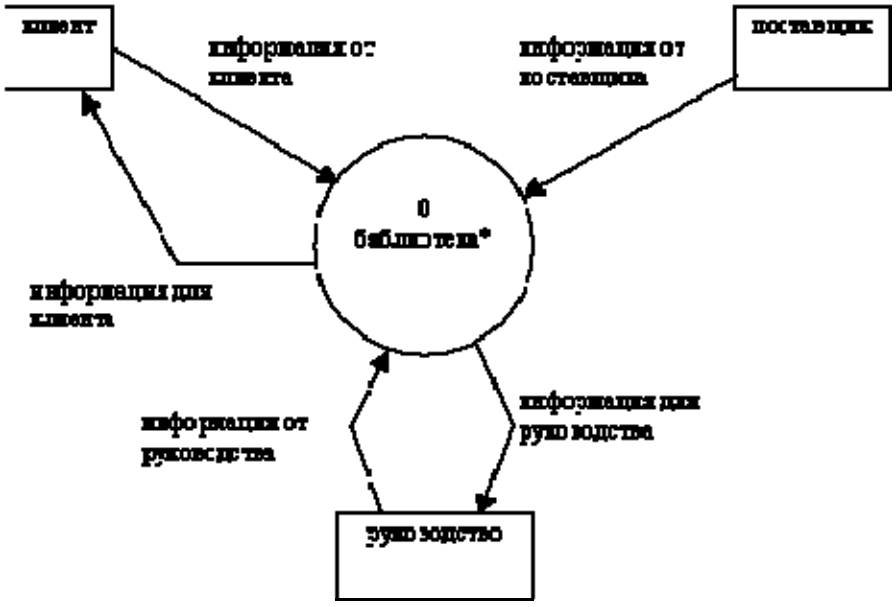


Рис. 2.42. Начальная контекстная диаграмма

Список событий строится в виде матрицы (ELM) и описывает различные действия внешних сущностей и реакцию ИС на них. Эти действия представляют собой внешние события, воздействующие на библиотеку. Различают следующие типы событий:

Аббревиатура	Тип
NC	Нормальное управление
ND	Нормальные данные
NCD	Нормальное управление/данные
TC	Временное управление
TD	Временные данные
TCD	Временное управление/данные

Все действия помечаются как нормальные данные. Эти данные являются событиями, которые ИС воспринимает непосредственно, например, изменение адреса клиента, которое должно быть сразу зарегистрировано. Они появляются в DFD в качестве содержимого потоков данных.

Матрица списка событий имеет следующий вид:

№	Описание	Тип	Реакция
1	Клиент желает стать членом библиотеки	ND	Регистрация клиента в качестве члена библиотеки
2	Клиент сообщает об изменении адреса	ND	Регистрация измененного адреса клиента
3	Клиент запрашивает аренду фильма	ND	Рассмотрение запроса
4	Клиент возвращает фильм	ND	Регистрация возврата
5	Руководство предоставляет полномочия новому поставщику	ND	Регистрация поставщика
6	Поставщик сообщает об изменении адреса	ND	Регистрация измененного адреса поставщика
7	Поставщик направляет фильм в библиотеку	ND	Получение нового фильма
8	Руководство запрашивает новый отчет	ND	Формирование требуемого отчета для руководства

Для завершения анализа функционального аспекта поведения системы строится полная контекстная диаграмма, включающая диаграмму нулевого уровня. При этом процесс

"библиотека" декомпозируется на 4 процесса, отражающие основные виды административной деятельности библиотеки. Существующие "абстрактные" потоки данных между терминаторами и процессами трансформируются в потоки, представляющие обмен данными на более конкретном уровне. Список событий показывает, какие потоки существуют на этом уровне: каждое событие из списка должно формировать некоторый поток (событие формирует входной поток, реакция - выходной поток). Один "абстрактный" поток может быть разделен на более чем один "конкретный" поток.

Потоки на диаграмме верхнего уровня	**Потоки на диаграмме нулевого уровня**
Информация от клиента	Данные о клиенте, Запрос об аренде
Информация для клиента	Членская карточка, Ответ на запрос об аренде
Информация от руководства	Запрос отчета о новых членах, Новый поставщик, Запрос отчета о поставщиках, Запрос отчета об аренде, Запрос отчета о фильмах
Информация для руководства	Отчет о новых членах, Отчет о поставщиках, Отчет об аренде, Отчет о фильмах
Информация от поставщика	Данные о поставщике, Новые фильмы

На приведенной DFD (рисунок 2.43) накопитель данных "библиотека" является глобальным или абстрактным представлением хранилища данных.

Анализ функционального аспекта поведения системы дает представление об обмене и преобразовании данных в системе. Взаимосвязь между "абстрактными" потоками данных и "конкретными" потоками данных на диаграмме нулевого уровня выражается в диаграммах структур данных (рисунок 2.44).

На фазе анализа строится глобальная модель данных, представляемая в виде диаграммы "сущность-связь" (рисунок 2.45).

Между различными типами диаграмм существуют следующие взаимосвязи:

- ELM-DFD: события - входные потоки, реакции - выходные потоки
- DFD-DSD: потоки данных - структуры данных верхнего уровня
- DFD-ERD: накопители данных - ER-диаграммы
- DSD-ERD: структуры данных нижнего уровня - атрибуты сущностей

На фазе проектирования архитектуры строится предметная модель. Процесс построения предметной модели включает в себя:

- детальное описание функционирования системы;
- дальнейший анализ используемых данных и построение логической модели данных для последующего проектирования базы данных;
- определение структуры пользовательского интерфейса, спецификации форм и порядка их появления;
- уточнение диаграмм потоков данных и списка событий, выделение среди процессов нижнего уровня интерактивных и неинтерактивных, определение для них миниспецификаций.

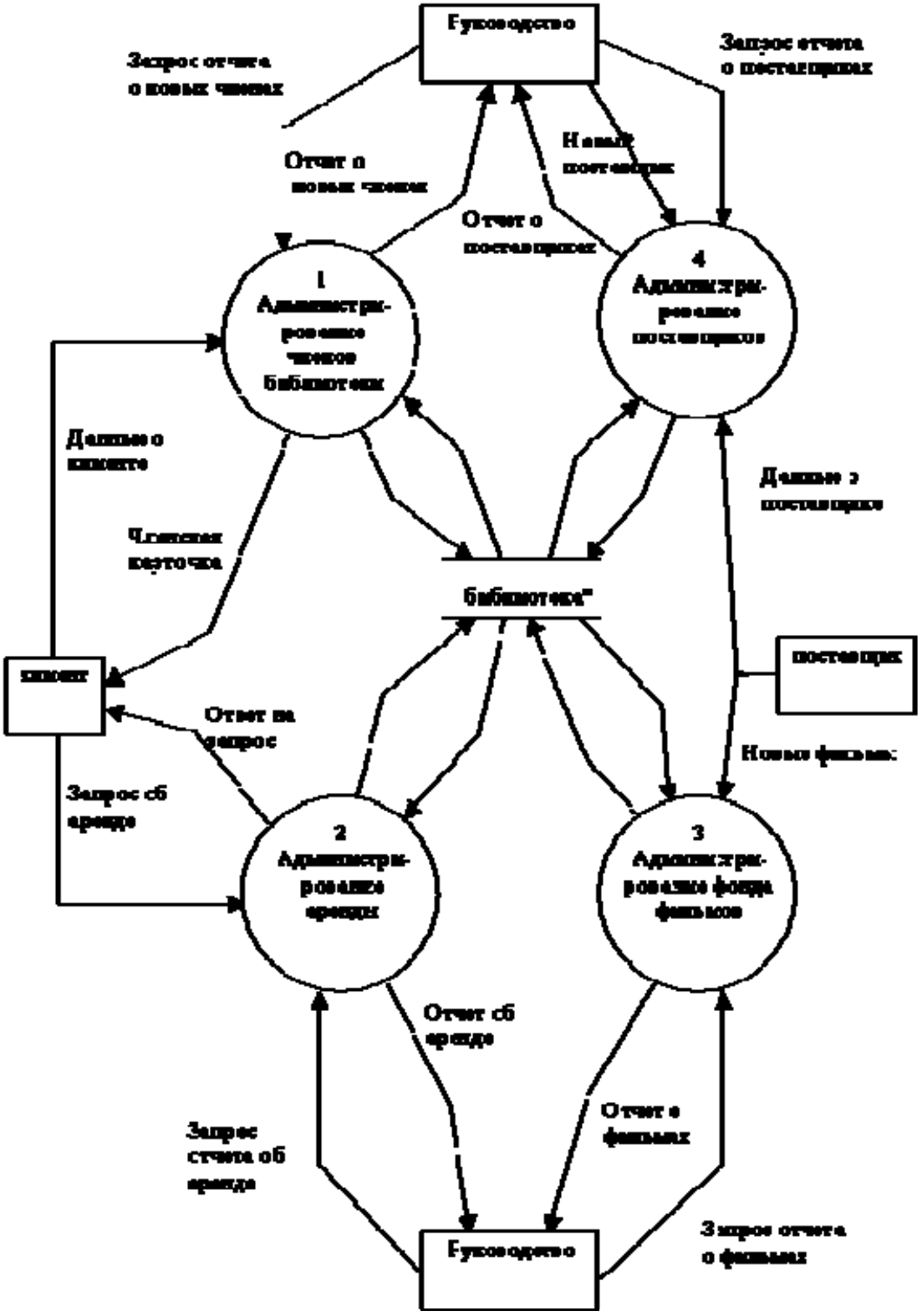


Рис. 2.43. Контекстная диаграмма

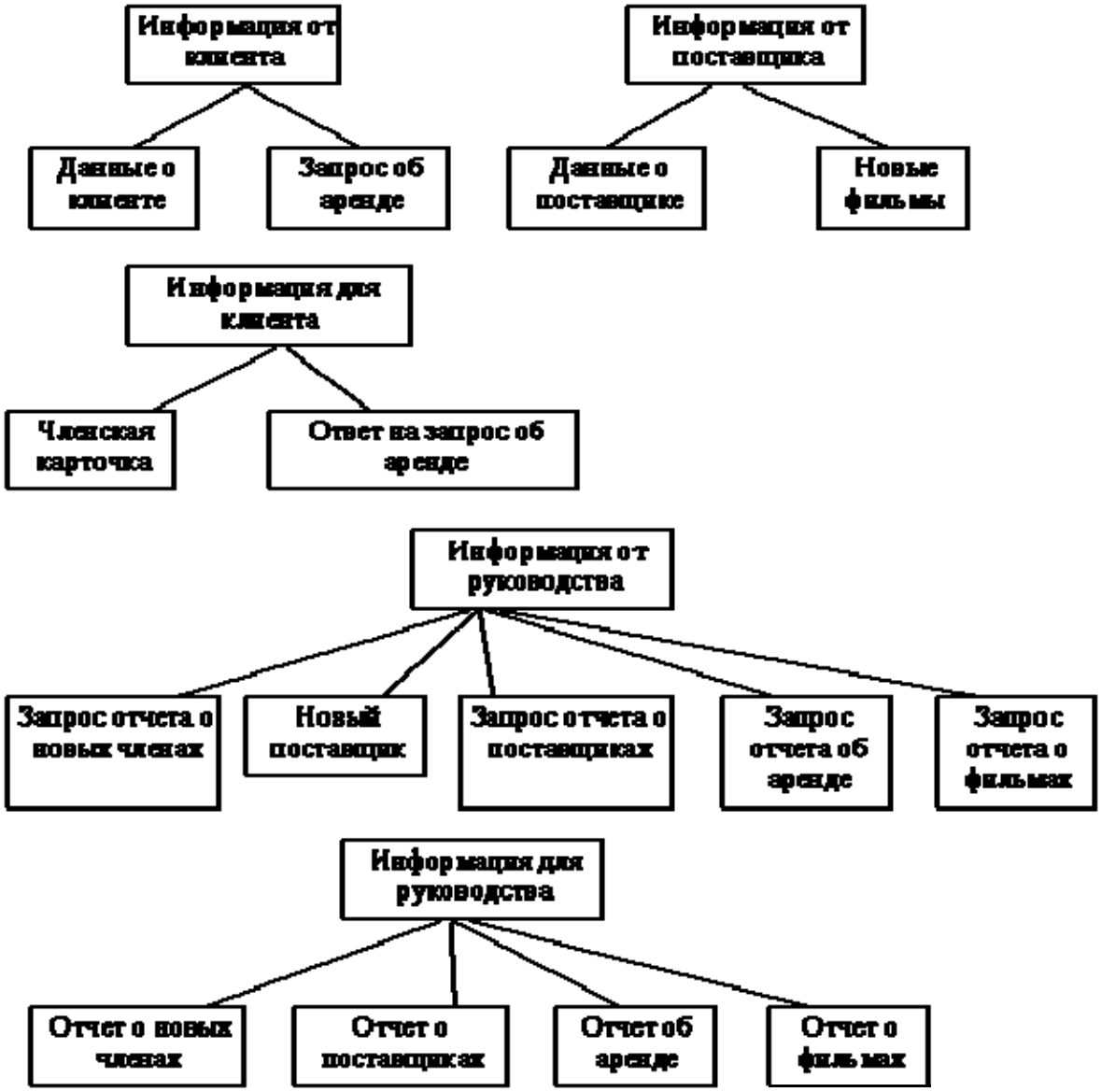


Рис. 2.44. Диаграмма структур данных

Результатами проектирования архитектуры являются:

- модель процессов (диаграммы архитектуры системы (SAD) и миниспецификации на структурированном языке);
- модель данных (ERD и подсхемы ERD);
- модель пользовательского интерфейса (классификация процессов на интерактивные и неинтерактивные функции, диаграмма последовательности форм (FSD - Form Sequence Diagram), показывающая, какие формы появляются в приложении и в каком порядке. На FSD фиксируется набор и структура вызовов экранных форм. Диаграммы FSD образуют иерархию, на вершине которой находится главная форма приложения, реализующего подсистему. На втором уровне находятся формы, реализующие процессы нижнего уровня функциональной структуры, зафиксированной на диаграммах SAD.

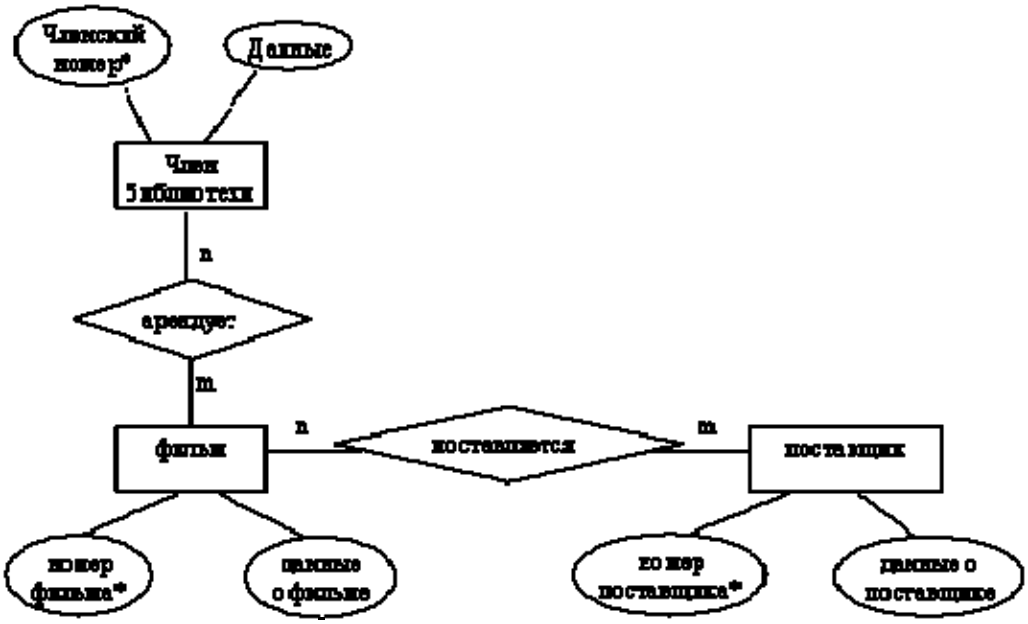


Рис. 2.45. Диаграмма "сущность-связь"

На фазе детального проектирования строится модульная модель. Под модульной моделью понимается реальная модель проектируемой прикладной системы. Процесс ее построения включает в себя:

- уточнение модели базы данных для последующей генерации SQL-предложений;
- уточнение структуры пользовательского интерфейса;
- построение структурных схем, отражающих логику работы пользовательского интерфейса и модель бизнес-логики (Structure Charts Diagram - SCD) и привязка их к формам.
- Результатами детального проектирования являются:
- модель процессов (структурные схемы интерактивных и неинтерактивных функций);
- модель данных (определение в ERD всех необходимых параметров для приложений);
- модель пользовательского интерфейса (диаграмма последовательности форм (FSD), показывающая, какие формы появляются в приложении и в каком порядке, взаимосвязь между каждой формой и определенной структурной схемой, взаимосвязь между каждой формой и одной или более сущностями в ERD).

На фазе реализации строится реализационная модель. Процесс ее построения включает в себя:

- генерацию SQL-предложений, определяющих структуру целевой БД (таблицы, индексы, ограничения целостности);
- уточнение структурных схем (SCD) и диаграмм последовательности форм (FSD) с последующей генерацией кода приложений.

На основе анализа потоков данных и взаимодействия процессов с хранилищами данных осуществляется окончательное выделение подсистем (предварительное должно было быть сделано и зафиксировано на этапе формулировки требований в техническом задании). При выделении подсистем необходимо руководствоваться принципом функциональной связанности и принципом минимизации информационной зависимости. Необходимо учитывать, что на основании таких элементов подсистемы как процессы и данные на этапе разработки должно быть создано приложение, способное функционировать самостоятельно. С другой стороны при группировке процессов и данных в подсистемы необходимо учитывать требования к конфигурированию продукта, если они были сформулированы на этапе анализа.

3. Программные средства поддержки жизненного цикла ПО

3.1. Методологии проектирования ПО как программные продукты. Методология DATARUN и инструментальное средство SE Companion

Современные методологии и реализующие их технологии поставляются в электронном виде вместе с CASE-средствами и включают библиотеки процессов, шаблонов, методов, моделей и других компонент, предназначенных для построения ПО того класса систем, на который ориентирована методология. Электронные методологии включают также средства, которые должны обеспечивать их адаптацию для конкретных пользователей и развитие методологии по результатам выполнения конкретных проектов.

Процесс адаптации заключается в удалении ненужных процессов, действий ЖЦ и других компонентов методологии, в изменении неподходящих или в добавлении собственных процессов и действий, а также методов, моделей, стандартов и руководств. Настройка методологии может осуществляться также по следующим аспектам: этапы и операции ЖЦ, участники проекта, используемые модели ЖЦ, поддерживаемые концепции и др.

Электронные методологии и технологии (и поддерживающие их CASE-средства) составляют ядро комплекса согласованных инструментальных средств среды разработки ИС.

3.1.1. Методология DATARUN

Одной из наиболее распространенных в мире электронных методологий является методология DATARUN [6,26]. В соответствии с методологией DATARUN ЖЦ ПО разбивается на стадии, которые связываются с результатами выполнения основных процессов, определяемых стандартом ISO 12207. Каждую стадию кроме ее результатов должен завершать план работ на следующую стадию.

Стадия формирования требований и планирования включает в себя действия по определению начальных оценок объема и стоимости проекта. Должны быть сформулированы требования и экономическое обоснование для разработки ИС, функциональные модели (модели бизнес-процессов организации) и исходная концептуальная модель данных, которые дают основу для оценки технической реализуемости проекта. Основными результатами этой стадии должны быть модели деятельности организации (исходные модели процессов и данных организации), требования к системе, включая требования по сопряжению с существующими ИС, исходный бизнес-план.

Стадия концептуального проектирования начинается с детального анализа первичных данных и уточнения концептуальной модели данных, после чего проектируется архитектура системы. Архитектура включает в себя разделение концептуальной модели на обозримые подмодели. Оценивается возможность использования существующих ИС и выбирается соответствующий метод их преобразования. После построения проекта уточняется исходный бизнес-план. Выходными компонентами этой стадии являются концептуальная модель данных, модель архитектуры системы и уточненный бизнес-план.

На стадии спецификации приложений продолжается процесс создания и детализации проекта. Концептуальная модель данных преобразуется в реляционную модель данных. Определяется структура приложения, необходимые интерфейсы приложения в виде экранов, отчетов и пакетных процессов вместе с логикой их вызова. Модель данных уточняется бизнес-правилами и методами для каждой таблицы. В конце этой стадии принимается окончательное решение о способе реализации приложений. По результатам стадии должен быть построен проект ИС, включающий модели архитектуры ИС, данных, функций, интерфейсов (с внешними системами и с пользователями), требований к разрабатываемым приложениям (модели данных, интерфейсов и функций), требований к доработкам существующих ИС, требований к интеграции приложений, а также сформирован окончательный план создания ИС.

На стадии разработки, интеграции и тестирования должна быть создана тестовая база данных, частные и комплексные тесты. Проводится разработка, прототипирование и тестирование баз данных и приложений в соответствии с проектом. Отлаживаются интерфейсы с

существующими системами. Описывается конфигурация текущей версии ПО. На основе результатов тестирования проводится оптимизация базы данных и приложений. Приложения интегрируются в систему, проводится тестирование приложений в составе системы и испытания системы. Основными результатами стадии являются готовые приложения, проверенные в составе системы на комплексных тестах, текущее описание конфигурации ПО, скорректированная по результатам испытаний версия системы и эксплуатационная документация на систему.

Стадия внедрения включает в себя действия по установке и внедрению баз данных и приложений. Основными результатами стадии должны быть готовая к эксплуатации и перенесенная на программно-аппаратную платформу заказчика версия системы, документация сопровождения и акт приемочных испытаний по результатам опытной эксплуатации.

Стадии сопровождения и развития включают процессы и операции, связанные с регистрацией, диагностикой и локализацией ошибок, внесением изменений и тестированием, проведением доработок, тиражированием и распространением новых версий ПО в места его эксплуатации, переносом приложений на новую платформу и масштабированием системы. Стадия развития фактически является повторной итерацией стадии разработки.

Методология DATARUN опирается на две модели или на два представления:

- модель организации;
- модель ИС.

Методология DATARUN базируется на системном подходе к описанию деятельности организации. Построение моделей начинается с описания процессов, из которых затем извлекаются первичные данные (стабильное подмножество данных, которые организация должна использовать для своей деятельности). Первичные данные описывают продукты или услуги организации, выполняемые операции (транзакции) и потребляемые ресурсы. К первичным относятся данные, которые описывают внешние и внутренние сущности, такие как служащие, клиенты или агентства, а также данные, полученные в результате принятия решений, как например, графики работ, цены на продукты.

Основной принцип DATARUN заключается в том, что первичные данные, если они должным образом организованы в модель данных, становятся основой для проектирования архитектуры ИС. Архитектура ИС будет более стабильной, если она основана на первичных данных, тесно связанных с основными деловыми операциями, определяющими природу бизнеса, а не на традиционной функциональной модели.

Любая ИС (рисунок 3.1) представляет собой набор модулей, исполняемых процессорами и взаимодействующих с базами данных. Базы данных и процессоры могут располагаться централизованно или быть распределенными. События в системе могут инициироваться внешними сущностями, такими как клиенты у банкоматов или временные события (конец месяца или квартала). Все транзакции осуществляются через объекты или модули интерфейса , которые взаимодействуют с одной или более базами данных.

Подход DATARUN преследует две цели:

- определить стабильную структуру, на основе которой будет строиться ИС. Такой структурой является модель данных, полученная из первичных данных, представляющих фундаментальные процессы организации;
- спроектировать ИС на основании модели данных.

Объекты, формируемые на основании модели данных, являются объектами базы данных, обычно размещаемыми на серверах в среде клиент/сервер. Объекты интерфейса, определенные в архитектуре компьютерной системы, обычно размещаются на клиентской части. Модель данных, являющаяся основой для спецификации совместно используемых объектов базы данных и различных объектов интерфейса, обеспечивает сопровождаемость ИС. На рисунке 3.2 представлена последовательность шагов проектирования ИС.

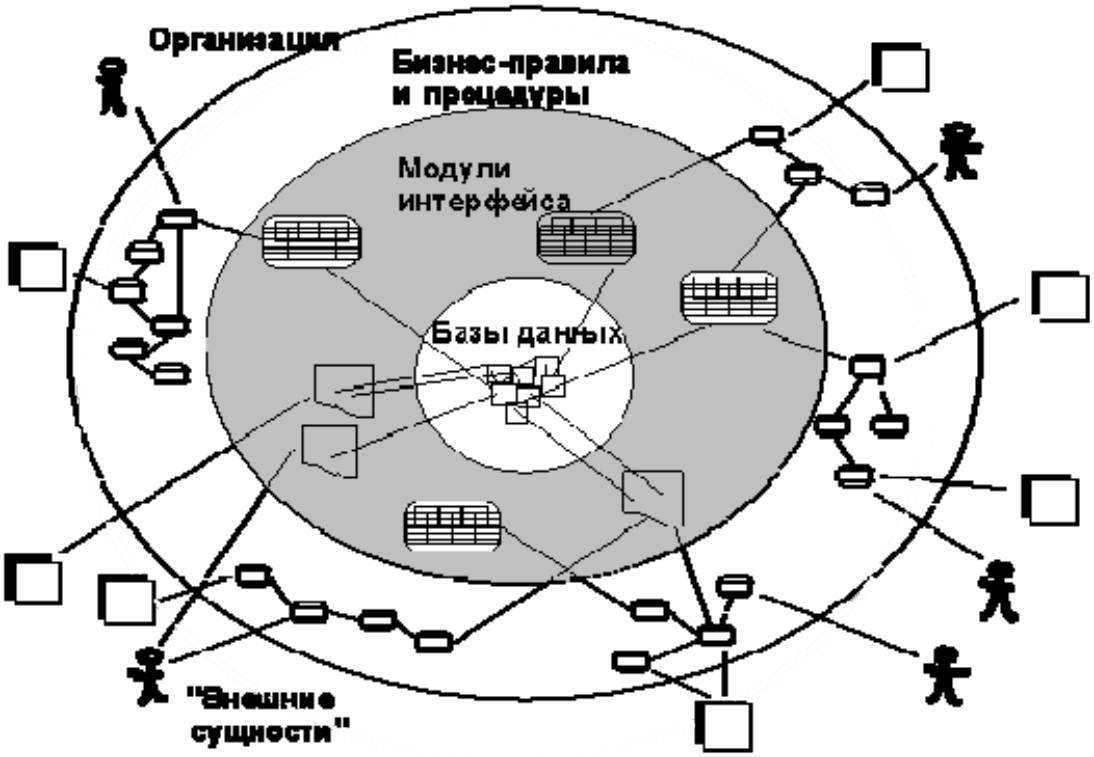


Рис. 3.1. Модель ИС

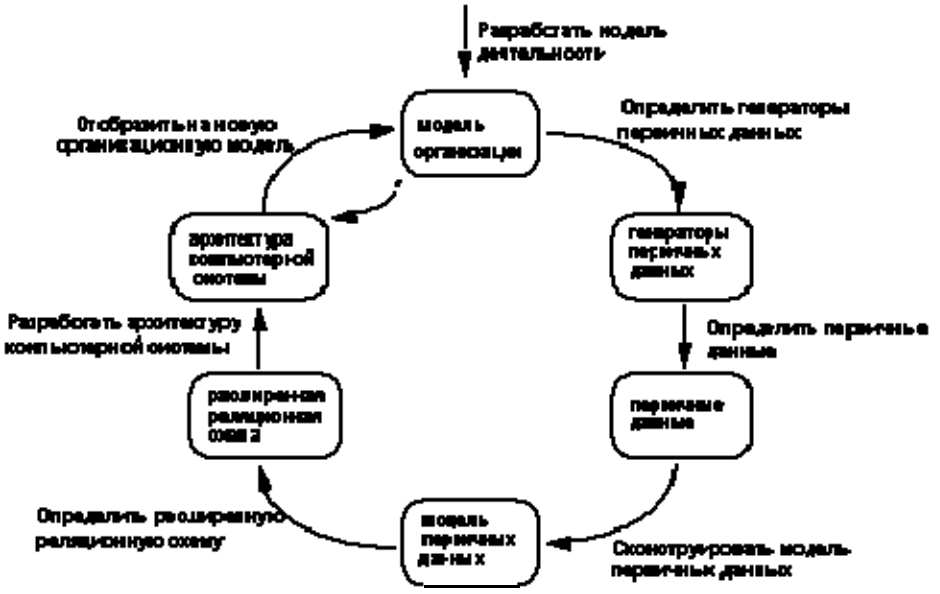
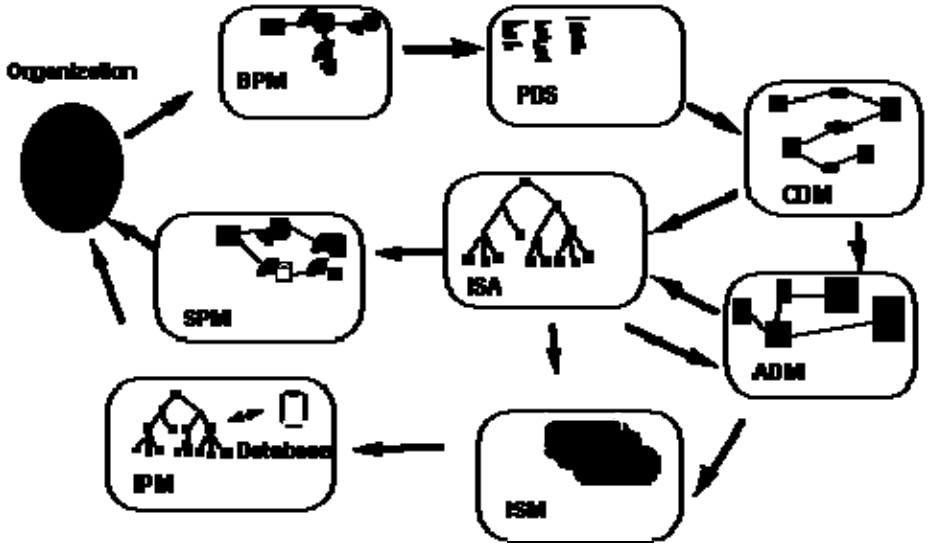


Рис. 3.2. Последовательность шагов проектирования системы

На рисунке 3.3 определены модели, создаваемые в процессе разработки ИС. Для их создания используется CASE-средство Silverrun, описанное в подразделе 5.1. Silverrun обеспечивает автоматизацию проведения проектных работ в соответствии с методологией DATARUN. Предоставляемая этими средствами среда проектирования дает возможность руководителю проекта контролировать проведение работ, отслеживать выполнение работ, вовремя замечать отклонения от графика. Каждый участник проекта, подключившись к этой среде, может выяснить содержание и сроки выполнения порученной ему работы, детально изучить технику ее выполнения в гипертексте по технологиям, и вызвать инструмент (модуль Silverrun) для реального выполнения работы.

Информационная система создается последовательным построением ряда моделей, начиная с модели бизнес-процессов и заканчивая моделью программы, автоматизирующей эти процессы.



BPM (**B**usiness **P**rocess **M**odel) - модель бизнес-процессов.
PDS (**P**rimary **D**ata **S**tructure) - структура первичных данных.
CDM (**C**onceptual **D**ata **M**odel) - концептуальная модель данных.
SPM (**S**ystem **P**rocess **M**odel) - модель процессов системы.
ISA (**I**nformation **S**ystem **A**rchitecture) - архитектура информационной системы.
ADM (**A**pplication **D**ata **M**odel) - модель данных приложения.
IPM (**I**nterface **P**resentation **M**odel) - модель представления интерфейса.
ISM (**I**nterface **S**pecification **M**odel) - модель спецификации интерфейса.

Рис. 3.3. Модели, создаваемые с помощью подхода DATARUN

Создаваемая ИС должна основываться на функциях, выполняемых организацией. Поэтому первая создаваемая модель - это модель бизнес-процессов, построение которой осуществляется в модуле Silverrun BPM. Для этой модели используется специальная нотация BPM. В процессе анализа и спецификации бизнес-функций выявляются основные информационные объекты, которые документируются как структуры данных, связанные с потоками и хранилищами модели. Источниками для создания структур являются используемые в организации документы, должностные инструкции, описания производственных операций. Эти данные вводятся в том виде, как они существуют в деятельности организации. Нормализация и удаление избыточности производится позже при построении концептуальной модели данных в модуле Silverrun ERX. После создания модели бизнес-процессов информация сохраняется в репозитории проекта.

В процессе обследования работы организации выявляются и документируются структуры первичных данных. Эти структуры заносятся в репозиторий модуля BPM при описании циркулирующих в организации документов, сообщений, данных. В модели бизнес-процессов первичные структуры данных связаны с потоками и хранилищами информации.

На основе структур первичных данных в модуле Silverrun ERX создается концептуальная модель данных (ER-модель). От структур первичных данных концептуальная модель отличается удалением избыточности, стандартизацией наименований понятий и нормализацией. Эти операции в модуле ERX выполняются при помощи встроенной экспертной системы. Цель концептуальной модели данных - описать используемую информацию без деталей возможной реализации в базе данных, но в хорошо структурированном нормализованном виде.

На основе модели бизнес-процессов и концептуальной модели данных проектируется архитектура ИС. Определяются входящие в систему приложения, для каждого приложения специфицируются используемые данные и реализуемые функции. Архитектура ИС создается в модуле Silverrun BPM с использованием специальной нотации ISA. Основное содержание этой модели - структурные компоненты системы и навигация между ними. Концептуальная модель данных разбивается на части, соответствующие входящим в состав системы приложениям.

Перед разработкой приложений должна быть спроектирована структура корпоративной базы данных. DATARUN предполагает использование базы данных, основанной на реляционной модели. Концептуальная модель данных после нормализации переносится в модуль реляционного

моделирования Silverrun RDM с помощью специального моста ERX-RDM. Преобразование модели из формата ERX в формат RDM происходит автоматически без вмешательства пользователя. После преобразования форматов получается модель реляционной базы данных. Эта модель детализируется в модуле Silverrun RDM определением физической реализации (типов данных СУБД, ключей, индексов, триггеров, ограничений ссылочной целостности). Правила обработки данных можно задавать как непосредственно на языке программирования СУБД, так и в декларативной форме, не привязанной к реализации. Мосты Silverrun к реляционным СУБД переводят эти декларативные правила на язык требуемой системы, что снижает трудоемкость программирования процедур сервера базы данных, а также позволяет из одной спецификации генерировать приложения для разных СУБД.

С помощью модели системных процессов детально документируется поведение каждого приложения. В модуле BPM создается модель системных процессов, определяющая, каким образом реализуются бизнес-процессы. Эта модель создается отдельно для каждого приложения и тесно связана с моделью данных приложения.

Приложение состоит из интерфейсных объектов (экранных форм, отчетов, процедур обработки данных). Каждый интерфейс системы (экранная форма, отчет, процедура обработки данных) имеет дело с подмножеством базы данных. В модели данных приложения (созданной в модуле RDM) создается подсхема базы данных для каждого интерфейса этого приложения. Уточняются также правила обработки данных, специфичные для каждого интерфейса. Интерфейс работает с данными в ненормализованном виде, поэтому спецификация данных, как ее видит интерфейс, оформляется как отдельная подсхема модели данных интерфейса.

Модель представления интерфейса - это описание внешнего вида интерфейса, как его видит конечный пользователь системы. Это может быть как документ, показывающий внешний вид экрана или структуру отчета, так и сам экран (отчет), созданный с помощью одного из средств визуальной разработки приложений - так называемых языков четвертого поколения (4GL - Fourth Generation Languages). Так как большинство языков 4GL позволяют быстро создавать работающие прототипы приложений, пользователь имеет возможность увидеть работающий прототип системы на ранних стадиях проектирования.

После создания подсхем реляционной модели для приложений проектируется детальная структура каждого приложения в виде схемы навигации экранов, отчетов, процедур пакетной обработки. На данном шаге эта структура детализируется до указания конкретных столбцов и таблиц базы данных, правил их обработки, вида экранных форм и отчетов. Полученная модель детально документирует приложение и непосредственно используется для программирования специфицированных интерфейсов.

Далее, с помощью средств разработки приложений происходит физическое создание системы: приложения программируются и интегрируются в информационную систему.

3.1.2. Инструментальное средство SE Companion

Инструментальное средство SE Companion [27] является средой, в которой реализован электронный вариант методологии DATARUN. Оно позволяет:

- создать гипертекстовое описание методологии в виде иерархии описания стадий, этапов и операций разработки;
- создать гипертекстовое описание всех методов и методик реализации процессов ЖЦ ПО;
- выделить из гипертекстового описания иерархию процессов ЖЦ ПО для планирования и управления процессом создания ПО (иерархию работ);
- изменять гипертекстовые описания ЖЦ и методов так, как это необходимо разработчику, иными словами, производить авторизацию методологии и отслеживать эти изменения в иерархии работ, предназначенной для управления проектом;
- привязать к процессам ЖЦ инструментальные средства поддержки этих процессов и обеспечить вызов инструментальных средств из соответствующих экранов гипертекстового справочника;
- обеспечить просмотр гипертекстовых экранов описания используемых методов из инструментальных средств;
- обеспечить поддержку процесса управления разработкой, в частности, за счет

взаимодействия со средством планирования работ MS Project, оценивания трудоемкости проекта, отслеживания выполнения работ, создания графиков работ, и др.

Особенно важными являются возможность авторизации методологии и интерактивный доступ любого разработчика к описанию любого метода или процесса в нужный ему момент времени. На современном этапе развития технологии, в условиях быстрого изменения как программных и аппаратных средств, так и задач бизнеса, методология создания, сопровождения и развития ПО не должна быть неизменной; она должна иметь возможность изменяться и настраиваться на новые технологии, методы и инструментальные средства. Современные разработчики больших ИС приобретают одну или несколько методологий поставщика, а затем создают на их основе собственные методологии и технологии, адаптированные к конкретным условиям (см. подраздел 1.3).

В SE Companion исходным документом, описывающим методологию (как процессы ЖЦ, так и все сопутствующие методы и методики), является файл в формате MS Word. Это обеспечивает возможности для описания методологии с любой степенью детализации, проведения разметки для создания гипертекста и авторизации методологии в принятом стандартном формате.

Гипертекстовое описание методологии и технологии создания ПО строится из описания процессов жизненного цикла, методов и методик, и представляет собой единый гипертекстовый документ в формате MS Help. Итоговое гипертекстовое описание получается в результате трансляции исходного документа. Все изменения и дополнения методологии производятся посредством корректировки и, возможно, дополнительной разметки исходного документа.

Описание методологии создания системы обычно состоит из раздела описания процессов ЖЦ и разделов описания методов и методик. В свою очередь, раздел описаний процессов состоит из иерархии описаний стадий, этапов и операций жизненного цикла с обязательным описанием выходных компонентов каждого процесса. Компоненты ПО создаются с применением методик и методов, описываемых в соответствующих разделах.

Минимальная конфигурация аппаратно-программных средств, требуемых SE Companion Authoring Tool:

- процессор: i486/33;
- оперативная память: 4 Мбайт для просмотра гипертекста, 12 Мбайт для авторизации;
- дисковая память: 20 Мбайт;
- операционные среды: Microsoft Windows 3.1, Microsoft Windows for Workgroups 3.11, Microsoft Windows NT 3.5, Microsoft Windows 95.

3.2. CASE-средства. Общая характеристика и классификация

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 ПО.

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 ПО и обладающее следующими основными

характерными особенностями:

- мощные графические средства для описания и документирования ИС, обеспечивающие удобный интерфейс с разработчиком и развивающие его творческие возможности;
- интеграция отдельных компонент CASE-средств, обеспечивающая управляемость процессом разработки ИС;
- использование специальным образом организованного хранилища проектных метаданных (репозитория).

Интегрированное CASE-средство (или комплекс средств, поддерживающих полный ЖЦ ПО) 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.

Требования к функциям отдельных компонент в виде критериев оценки CASE-средств приведены в разделе 4.2.

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:

- средства анализа (Upper CASE), предназначенные для построения и анализа моделей предметной области (Design/IDEF (Meta Software), BPwin (Logic Works));
- средства анализа и проектирования (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 (МакроПроджект)). 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:

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

Описание перечисленных CASE-средств приведено в разделе 5. 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.

4. Технология внедрения CASE-средств

Приведенная в данном разделе технология базируется в основном на стандартах IEEE [16,17] (IEEE - Institute of Electrical and Electronics Engineers - Институт инженеров по электротехнике и электронике). Термин "внедрение" используется в широком смысле и включает все действия от оценки первоначальных потребностей до полномасштабного использования CASE-средств в различных подразделениях организации-пользователя. Процесс внедрения CASE-средств состоит из следующих этапов [16]:

- определение потребностей в CASE-средствах;
- оценка и выбор CASE-средств;
- выполнение пилотного проекта;
- практическое внедрение CASE-средств.

Процесс успешного внедрения CASE-средств не ограничивается только их использованием. На самом деле он охватывает планирование и реализацию множества технических, организационных, структурных процессов, изменений в общей культуре организации, и основан на четком понимании возможностей CASE-средств.

На способ внедрения CASE-средств может повлиять специфика конкретной ситуации. Например, если заказчик предпочитает конкретное средство, или оно оговаривается требованиями контракта, этапы внедрения должны соответствовать такому предопределенному выбору. В иных ситуациях относительная простота или сложность средства, степень согласованности или конфликтности с существующими в организации процессами, требуемая степень интеграции с другими средствами, опыт и квалификация пользователей могут привести к внесению соответствующих корректив в процесс внедрения.

4.1. Определение потребностей в CASE-средствах

Данный этап (рисунок 4.1) включает достижение понимания потребностей организации и технологии последующего процесса внедрения CASE-средств. Он должен привести к выделению тех областей деятельности организации, в которых применение CASE-средств может принести реальную пользу. Результатом данного этапа является документ, определяющий стратегию внедрения CASE-средств.

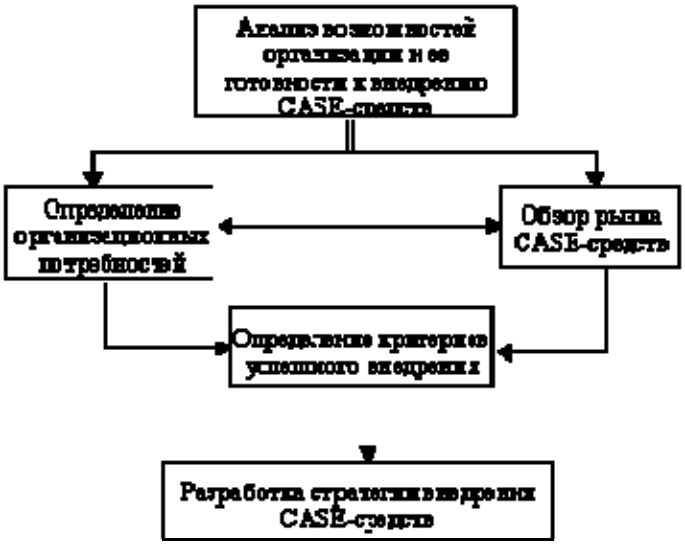


Рис. 4.1. Определение потребностей в CASE-средствах

4.1.1. Анализ возможностей организации

Первым действием данного этапа является анализ возможностей организации в отношении ее технологической базы, персонала и используемого ПО. Такой анализ может быть формальным или неформальным.

Формальные подходы определяются моделью оценки зрелости технологических процессов организации CMM (Capability Maturity Model), разработанной SEI (Software Engineering Institute),

а также стандартами ISO 9001: 1994, ISO 9003-3: 1991 и ISO 9004-2:1991. В центре внимания этих подходов находится анализ различных аспектов происходящих в организации процессов.

Для получения информации относительно положения и потребностей организации могут использоваться неформальные оценки и анкетирование. Список простых вопросов, которые могут помочь в неформальной оценке текущей практики использования ПО, технологии и персонала, приведен ниже.

Ответы на данные вопросы могут определить те области, где автоматизация может принести эффект. В противном случае может оказаться, что совершенствование процесса разработки и сопровождения ПО, программ обучения и других функций более предпочтительно, чем приобретение новых средств. Некоторые из этих усовершенствований могут оказаться необходимыми для получения максимальной выгоды от внедрения любых средств.

Данные вопросы являются, по существу, руководством по сбору информации, необходимой для определения степени готовности организации к внедрению CASE-технологии.

Общие вопросы

- используемая модель ЖЦ (каскадная или спиральная);
- используемые методы (структурные, объектно-ориентированные). Степень адаптации метода к потребностям организации; квалификация сотрудников;
- наличие документированных стандартов (формальных или неформальных) по анализу требований, спецификациям и проектированию, кодированию и тестированию;
- количественные метрики, используемые в процессе разработки ПО, их использование;
- виды документации, выпускаемой в процессе ЖЦ ПО;
- наличие группы поддержки средств проектирования.

Проекты, ведущиеся в организации

- средняя продолжительность проекта в человеко-месяцах;
- среднее количество специалистов, участвующих в проектах различных категорий (небольших, средних и крупных);
- средний размер проектов различных категорий в терминах кодовых метрик (например, в строках исходных кодов), способ измерения.

Технологическая база

Технологическая база организации включает не только технические средства, используемые при разработке ПО, но также языки, средства, методы и среду функционирования ПО. Эта база очень существенно влияет на выбор подходящих CASE-средств. Вопросы, касающиеся технологии, включают следующие:

- доступные вычислительные ресурсы, платформа разработки;
- уровень доступности ресурсов, узкие места, среднее время ожидания ресурсов;
- ПО, используемое в организации, и его характер (готовые программные продукты, собственные разработки);
- степень интеграции используемых программных продуктов, механизмы интеграции (существующие и планируемые);
- тип и уровень сетевых возможностей, доступных группе разработчиков;
- используемые языки программирования;
- средний процент вновь разрабатываемых, повторно используемых и реально эксплуатируемых приложений.

Персонал

Главной целью оценки персонала является определение его отношения к возможным изменениям (позитивного, нейтрального или негативного). Вопросы, касающиеся оценки персонала, включают следующие:

- реакция сотрудников организации (как отдельных людей, так и коллективов) на внедрение новой технологии. Наличие опыта успешных или безуспешных внедрений;
- наличие лидеров, способных серьезно повлиять на отношение к новым средствам;
- наличие стремления "снизу" к совершенствованию средств и технологии;
- объем обучения, необходимого для ориентации пользователей в новой технологии;
- стабильность и уровень текучести кадров.

Готовность

Целью оценки готовности организации является определение того, насколько она способна воспринять как немедленные, так и долгосрочные последствия внедрения CASE-средств. Вопросы, касающиеся оценки готовности, включают следующие:

- поддержка проекта со стороны высшего руководства;
- готовность организации к долгосрочному финансированию проекта;
- готовность организации к выделению необходимых специалистов для участия в процессе внедрения и к их обучению;
- готовность персонала к существенному изменению технологии своей работы;
- степень понимания персоналом масштаба изменений;
- готовность технических специалистов и менеджеров пойти на возможное кратковременное снижение продуктивности своей работы;
- готовность руководства к долговременному ожиданию отдачи от вложенных средств.

Оценка готовности организации к внедрению CASE-технологии должна быть откровенной и тщательной, поскольку в случае отсутствия такой готовности все усилия по внедрению потерпят крах.

4.1.2. Определение организационных потребностей

Организационные потребности следуют непосредственно из проблем организации и целей, которые она стремится достичь. Проблемы и цели могут быть связаны с управлением, производством продукции, экономикой, персоналом или технологией. Вопросы, касающиеся определения целей, потребностей и ожидаемых результатов, приведены ниже. Определение потребностей должно выполняться в сочетании с обзором рынка CASE-средств, поскольку информация о технологиях, доступных на рынке в данный момент, может оказать влияние на потребности.

Цели организации

Цели организации играют главную роль в определении ее конкретных потребностей и ожидаемых результатов. Для их понимания необходимо ответить на следующие вопросы:

- намерение организации использовать CASE-технологию для помощи в достижении определенных целей или ожиданий (например, определенного уровня CMM или сертификации в соответствии с ISO 9001);
- восприятие CASE-технологии как фактора, способствующего достижению стратегических целей организации;
- наличие у организации собственной программы совершенствования процесса разработки ПО;
- восприятие инициативы внедрения CASE-технологии как части более широкомасштабного проекта по созданию среды разработки ПО.

Потребности организации

Определение потребностей организации, связанных с использованием CASE-технологии, включает анализ целей и существующих возможностей. После того, как все потребности организации определены, каждой из них должен быть присвоен определенный приоритет, отражающий ее значимость для успешной деятельности организации. Если потребности, связанные с CASE-технологией, не обладают высшим приоритетом, имеет смысл отказаться от ее внедрения и сосредоточиться на потребностях с наивысшим приоритетом.

Целесообразно построить матрицу соответствия потребностей организации возможностям основных CASE-средств. Составление такой матрицы требует определенного уровня знаний рынка CASE-средств. В конечном счете каждая функция или возможность средства должна точно соответствовать некоторой потребности с определенным приоритетом.

Определению потребностей организации могут помочь ответы на следующие вопросы:

- каким образом продуктивность и качество деятельности организации сравниваются с аналогичными показателями подобных организаций (к сожалению, многие организации не располагают данными для такого сравнения);
- какие процессы ЖЦ ПО дают наилучшую (и, соответственно, наихудшую) отдачу; существуют ли конкретные процессы, которые могут быть усовершенствованы путем использования новых методов и средств.

Ожидаемые результаты

С внедрением CASE-средств обычно связывают большие ожидания. В ряде случаев эти ожидания оказываются нереалистичными и приводят к неудаче при внедрении.

Составление реалистичного перечня ожидаемых результатов является трудной задачей, поскольку он может зависеть от таких факторов, как тип внедряемых средств и характеристики внедряющей организации.

Ряд потенциально реалистичных и нереалистичных ожидаемых результатов, связанных с организацией в целом, пользователями, планированием, анализом, проектированием, разработкой и затратами, приведен ниже. Практически невозможно, чтобы в процессе одного внедрения CASE-средств были достигнуты все положительные результаты. Тем не менее, любая организация может выработать собственный подход к ожидаемым результатам, имея в виду, что данный перечень является всего лишь примером.

Реалистичные ожидания:

- повышение внимания к планированию деятельности, связанной с информационной технологией;
- поддержка реижиниринга бизнес-процессов;
- долговременное повышение продуктивности и качества деятельности организации;
- ускорение и повышение согласованности разработки приложений;
- снижение доли ручного труда в процессе разработки и/или эксплуатации;
- более точное соответствие приложений требованиям пользователей;
- отсутствие необходимости большой переделки приложений для повышения их эффективности;
- улучшение реакции службы эксплуатации на требования внесения изменений и усовершенствований;
- повышение качества документирования;
- улучшение коммуникации между пользователями и разработчиками;
- последовательное и постоянное повышение качества проектирования;
- более высокие возможности повторного использования разработок;
- кратковременное возрастание затрат, связанное с деятельностью по внедрению CASE-средств;
- последовательное снижение общих затрат;
- улучшение прогнозируемости затрат.

Нереалистичные ожидания:

- отсутствие воздействия на общую культуру и распределение ролей в организации;
- понимание проектных спецификаций неподготовленными пользователями;
- сокращение персонала, связанного с информационной технологией;
- уменьшение степени участия в проектах высшего руководства и менеджеров, а также экспертов предметной области, уменьшение степени участия пользователей в процессе разработки приложений;
- немедленное повышение продуктивности деятельности организации;
- достижение абсолютной полноты и непротиворечивости спецификаций;
- автоматическая генерация прикладных систем из проектных спецификаций;
- немедленное снижение затрат, связанных с информационной технологией;
- снижение затрат на обучение.

Реализм в оценке ожидаемых затрат имеет особенно важное значение, поскольку он позволяет правильно оценить отдачу от инвестиций. Затраты на внедрение CASE-средств обычно недооцениваются. Среди конкретных статей затрат на внедрение можно выделить следующие:

- специалисты по планированию внедрения CASE-средств;
- выбор и установка;
- учет специфических требований персонала;
- приобретение CASE-средств и обучение;
- настройка;
- подготовка документации, стандартов и процедур использования средств;
- интеграция с другими средствами и существующими данными;

- освоение средств разработчиками;
- технические средства;
- обновление версий.

Важно также осознавать, что улучшение деятельности организации, являющееся следствием использования CASE-технологии, может быть неочевидным в течение самого первого проекта, использующего новую технологию. Продуктивность и другие характеристики деятельности организации могут первоначально даже ухудшиться, поскольку на освоение новых средств и внесение необходимых изменений в процесс разработки требуется некоторое время. Таким образом, ожидаемые результаты должны рассматриваться с учетом вероятной отсрочки в улучшении проектных характеристик.

Каждая потребность должна иметь определенный приоритет, зависящий от того, насколько критической она является для достижения успеха в организации. В конечном счете, должно четко прослеживаться воздействие каждой функции или возможности приобретаемых средств на удовлетворение конкретных потребностей.

Результатом данного действия является формулировка потребностей с их приоритетами, которая используется на этапе оценки и выбора в качестве "пользовательских потребностей".

4.1.3. Анализ рынка CASE-средств

Потребности организации в CASE-средствах должны соразмеряться с реальной ситуацией на рынке или собственными возможностями разработки. Исследование рынка проводится путем изучения литературы по CASE-средствам, посещения конференций и семинаров, проводимых поставщиками (их перечень приведен в конце данного обзора) и пользователями CASE-средств. При проведении данного анализа необходимо выяснить возможность интеграции конкретного CASE-средства с другими средствами, используемыми (или планируемыми к использованию) организацией. Кроме того, важно получить достоверную информацию о средствах, основанную на реальном пользовательском опыте и сведениях от пользовательских групп.

4.1.4. Определение критериев успешного внедрения

Определяемые критерии должны позволять количественно оценивать степень удовлетворения каждой из потребностей, связанных с внедрением. Кроме того, по каждому критерию должно быть определено его конкретное оптимальное значение. На определенных этапах внедрения эти критерии должны анализироваться для того, чтобы определить текущую степень удовлетворения потребностей.

Как правило, большинство организаций осуществляет внедрение CASE-средств для того, чтобы повысить продуктивность процессов разработки и сопровождения ПО, а также качество результатов разработки. Однако, ряд организаций не занимаются и не занимались ранее сбором количественных данных по указанным параметрам. Отсутствие таких данных затрудняет количественную оценку воздействия, оказываемого внедрением CASE-средств. В этом случае рекомендуется разработка соответствующих метрик. Информация о таких метриках приведена в стандартах IEEE Std 1045-1992 (IEEE Standard for Software Productivity Metrics) и IEEE Std 1061-1992 (IEEE Standard for a Software Quality Metrics Methodology).

В том случае, если базовые метрические данные отсутствуют, организация зачастую может извлечь полезную информацию из своих проектных архивов.

Помимо продуктивности и качества, полезную информацию о состоянии внедрения CASE-средств также могут дать и другие характеристики организационных процессов и персонала. Например, оценка степени успешности внедрения может включать процент проектов, использующих CASE-средства, рейтинговые оценки уровня квалификации специалистов, связанные с использованием CASE-средств и результаты опросов персонала по поводу отношения к использованию CASE-средств. Другие примеры проектных характеристик, которые могут быть оценены количественно, включают следующие:

- согласованность проектных результатов;
- точность стоимостных и плановых оценок;
- изменчивость внешних требований;

- соблюдение стандартов организации;
- степень повторного использования существующих компонентов ПО;
- объем и виды необходимого обучения;
- типы и моменты обнаружения проектных ошибок;
- вычислительные ресурсы, используемые CASE-средствами.

4.1.5. Разработка стратегии внедрения CASE-средств

Стратегия внедрения должна обеспечивать удовлетворение потребностей и критериев, определенных ранее. Стратегия включает следующие составляющие:

- организационные потребности;
- базовые метрики, необходимые для последующего сравнения результатов;
- критерии успешного внедрения, связанные с удовлетворением организационных потребностей, включая ожидаемые результаты последовательных этапов процесса внедрения;
- подразделения организации, в которых должно выполняться внедрение CASE-средств;
- влияние, оказываемое на другие подразделения организации;
- стратегии и планы оценки и выбора, пилотного проектирования и перехода к полномасштабному внедрению;
- основные факторы риска;
- ориентировочный уровень расходов и источники финансирования процесса внедрения CASE-средств;
- ключевой персонал и другие ресурсы.

Необходимо отметить, что внедрение новой технологии может включать важные и сложные изменения в культуре организации. Существенное внимание должно уделяться ролям различных групп, вовлеченных в процесс таких изменений. Наиболее существенные роли включают следующие:

- спонсор (обычно из числа менеджеров высшего уровня). Данная роль является критической для поддержки проекта и обеспечения необходимого финансирования. Спонсор должен обладать четким пониманием необходимости серьезных усилий, связанных с внедрением CASE-средств, и длительности периода ожидания осязаемых результатов;
- исполнитель - обычно лицо (или группа лиц), осознающее потенциальные возможности новой технологии, пользующееся авторитетом среди технического персонала и способное возглавить процесс внедрения новой технологии;
- целевая группа - обычно включает менеджеров и технический персонал, которые будут привлечены к непосредственному использованию CASE-средств, а также специалистов, которые будут привлечены косвенно, таких, как специалисты по документированию, персонал поддержки сети и заказчики. Должны быть определены потребности каждой из таких групп и план их эффективного удовлетворения.

В общем случае, внедрение CASE-средств должно управляться и финансироваться таким же образом, как и любой проект разработки ПО. Стратегия внедрения может быть пересмотрена в случае появления дополнительной информации.

Существует несколько подходов к разработке стратегии внедрения CASE-средств. Относительные преимущества того или иного подхода перед другими должны рассматриваться в контексте специфики конкретной организации. Особое значение при этом придается персоналу организации и процессу разработки ПО.

Нисходящий подход к разработке стратегии признает важность исследования всех типов CASE-средств и документирования процессов разработки и сопровождения ПО в данной организации до того, как определяются требования к CASE-средствам. При этом выполняется общий анализ процесса создания и сопровождения ПО в организации. Данный подход зачастую влечет за собой общую реорганизацию процессов создания и сопровождения ПО в той степени, в какой это связано с CASE-средствами. Результатом такой реорганизации становится крупномасштабная стратегия автоматизации процессов создания и сопровождения ПО.

Преимущество нисходящего подхода заключается в том, что он охватывает все процессы создания и сопровождения ПО, обеспечивая максимально возможную их автоматизацию. Другим преимуществом является приобретение интегрированного (или интегрируемого) набора средств, поскольку каждая отдельная поставка подчиняется общей стратегии. Нисходящий подход также может быть легко интегрирован в общую стратегию развития процесса создания и сопровождения ПО, в которой внедрение CASE-средств является только одним из аспектов.

Недостатки данного подхода заключаются в следующем:

- нисходящий подход требует для своей реализации значительных людских и финансовых ресурсов;
- в общем случае, широкомасштабный подход такого рода не позволяет пользователям достаточно быстро приступить к практическому использованию средств;
- нисходящий подход может привести к относительно серьезным изменениям существующих в организации процессов. Реализацией такого подхода труднее управлять, и, кроме того, он содержит в себе повышенный риск провала, ведущего к тому, что CASE-средства "кладутся на полку".

Нисходящий подход рекомендуется для относительно зрелых организаций с устоявшимся процессом создания и сопровождения ПО, которые стремятся вложить все необходимые ресурсы в полностью законченную работу. Чтобы повысить вероятность успеха, требуется принятие серьезных обязательств со стороны как руководства, так и потенциальных пользователей.

Восходящий подход начинается с определения некоторого средства или типа средств, которые потенциально могут помочь организации в улучшении выполнения текущей работы. Организация может затем оценить возможное воздействие средств на процесс разработки и сопровождения ПО.

Преимущества данного подхода заключаются в следующем:

- небольшая автоматизация может быть выполнена при минимальных затратах;
- автоматизация может быть выполнена за короткий промежуток времени, позволяя быстро устранить известные недостатки в существующих процессах;
- небольшой масштаб восходящей стратегии позволяет лучше фокусировать и контролировать воздействие, оказываемое на существующие процессы.
- Недостатки данного подхода заключаются в следующем:
- средства, приобретаемые как результат отдельных взятых применений данного подхода, могут плохо интегрироваться между собой. Это может привести к необходимости выполнения большого объема ручной работы;
- в то время как конкретные, сравнительно небольшие проблемы решаются достаточно быстро, до решения фундаментальных проблем, связанных с широким кругом процессов разработки ПО, дело обычно не доходит.

Восходящий подход рекомендуется для организаций с узко специфическими потребностями в автоматизации, не нуждающихся в общем совершенствовании процессов. В некоторых случаях может оказаться не слишком практичным приступать к такому совершенствованию, не определив самые насущные потребности в автоматизации. В то время как данный подход может помочь организации удовлетворить самые насущные потребности и развить основные процессы, остается существенная опасность того, что выбранное средство не окажет существенного воздействия на такие факторы, как качество и продуктивность.

Наиболее рациональная стратегия может сочетать характеристики обоих подходов. Например, нисходящие методы могут использоваться для определения стандартов качества организации, потребностей в средствах и ожидаемых результатов, тогда как восходящие методы могут использоваться для оценки и выбора конкретных CASE-средств, разработки планов внедрения и контроля его результатов.

4.2. Оценка и выбор CASE-средств

4.2.1. Общие сведения

Модель процесса оценки и выбора [17], рассматриваемая ниже (рисунок 4.2), описывает наиболее общую ситуацию оценки и выбора, а также показывает зависимость между ними. Как можно видеть, оценка и выбор могут выполняться независимо друг от друга или вместе, каждый из этих процессов требует применения определенных критериев.

Процесс оценки и выбора может преследовать несколько целей, включая одну или более из следующих:

- оценка нескольких CASE-средств и выбор одного или более из них;
- оценка одного или более CASE-средств и сохранение результатов для последующего использования;
- выбор одного или более CASE-средств с использованием результатов предыдущих оценок.

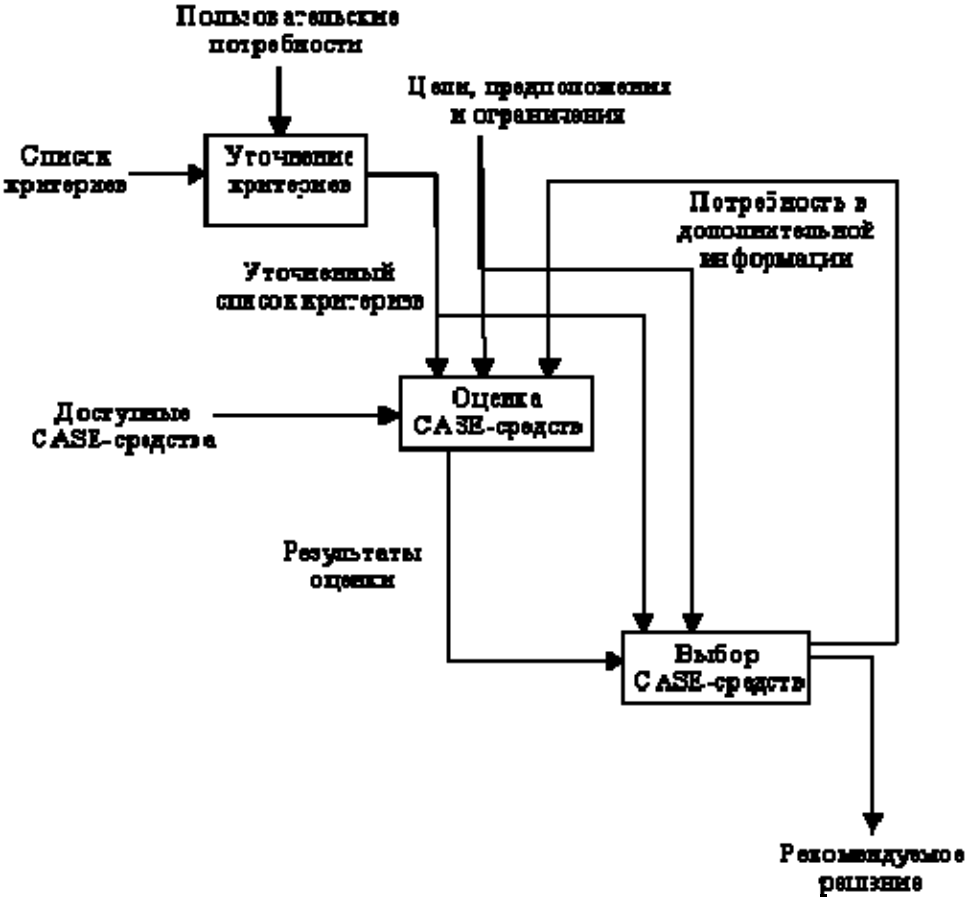


Рис. 4.2. Модель процесса оценки и выбора

Как видно из рисунка, входной информацией для процесса оценки является:

- определение пользовательских потребностей;
- цели и ограничения проекта;
- данные о доступных CASE-средствах;
- список критериев, используемых в процессе оценки.

Результаты оценки могут включать результаты предыдущих оценок. При этом не следует забывать, что набор критериев, использовавшихся при предыдущей оценке, должен быть совместимым с текущим набором. Конкретный вариант реализации процесса (оценка и выбор, оценка для будущего выбора или выбор, основанный на предыдущих оценках) определяется перечисленными выше целями.

Элементы процесса включают:

- цели, предположения и ограничения, которые могут уточняться в ходе процесса;
- потребности пользователей, отражающие количественные и качественные требования пользователей к CASE-средствам;

- критерии, определяющие набор параметров, в соответствии с которыми производится оценка и принятие решения о выборе;
- формализованные результаты оценок одного или более средств;
- рекомендуемое решение (обычно либо решение о выборе, либо дальнейшая оценка).

Процесс оценки и/или выбора может быть начат только тогда, когда лицо, группа или организация полностью определила для себя конкретные потребности и формализовала их в виде количественных и качественных требований в заданной предметной области. Термин "пользовательские требования" далее означает именно такие формализованные требования.

Пользователь должен определить конкретный порядок действий и принятия решений с любыми необходимыми итерациями. Например, процесс может быть представлен в виде дерева решений с его последовательным обходом и выбором подмножеств кандидатов для более детальной оценки. Описание последовательности действий должно определять поток данных между ними.

Определение списка критериев основано на пользовательских требованиях и включает:

- выбор критериев для использования из приведенного далее перечня;
- определение дополнительных критериев;
- определение области использования каждого критерия (оценка, выбор или оба процесса);
- определение одной или более метрик для каждого критерия оценки;
- назначение веса каждому критерию при выборе.

4.2.2. Процесс оценки

Целью процесса оценки является определение функциональности и качества CASE-средств для последующего выбора. Оценка выполняется в соответствии с конкретными критериями, ее результаты включают как объективные, так и субъективные данные по каждому средству.

Процесс оценки включает следующие действия:

- формулировка задачи оценки, включая информацию о цели и масштабах оценки;
- определение критериев оценки, вытекающее из определения задачи;
- определение средств-кандидатов путем просмотра списка кандидатов и анализа информации о конкретных средствах;
- оценка средств-кандидатов в контексте выбранных критериев. Необходимые для этого данные могут быть получены путем анализа самих средств и их документации, опроса пользователей, работы с демо-версиями, выполнения тестовых примеров, экспериментального применения средств и анализа результатов предшествующих оценок;
- подготовка отчета по результатам оценки.

Одним из важнейших критериев в процессе оценки может быть потенциальная возможность интеграции каждого из средств-кандидатов с другими средствами, уже находящимися в эксплуатации или планируемыми к использованию в данной организации.

Масштаб оценки должен устанавливать требуемый уровень детализации, необходимые ресурсы и степень применимости ее результатов. Например, оценка должна выполняться для набора из одного или более конкретных CASE-средств; CASE-средств, поддерживающих один или более конкретных процессов создания и сопровождения ПО или CASE-средств, поддерживающих один или более проектов или типов проектов.

Список CASE-средств - возможных кандидатов формируется из различных источников: обзоров рынка ПО, информации поставщиков, обзоров CASE-средств и других подобных публикаций.

Следующим шагом является получение информации о CASE-средствах или получение их самих или и то, и другое. Эта информация может состоять из оценок независимых экспертов, сообщений и отчетов поставщиков CASE-средств, результатов демонстрации возможностей CASE-средств со стороны поставщиков и информации, полученной непосредственно от реальных пользователей. Сами CASE-средства могут быть получены путем приобретения, в виде оценочной копии или другими способами.

Оценка и накопление соответствующих данных может выполняться следующими способами:

- анализ CASE-средств и документации поставщика;
- опрос реальных пользователей;
- анализ результатов проектов, использовавших данные CASE-средства;
- просмотр демонстраций и опрос демонстраторов;
- выполнение тестовых примеров;
- применение CASE-средств в пилотных проектах;
- анализ любых доступных результатов предыдущих оценок.

Существуют как объективные, так и субъективные критерии. Результаты оценки в соответствии с конкретным критерием могут быть двоичными, находиться в некотором числовом диапазоне, представлять собой просто числовое значение или иметь какую-либо другую форму.

Для объективных критериев оценка должна выполняться путем воспроизводимой процедуры, чтобы любой другой специалист, выполняющий оценку, мог получить такие же результаты. Если используются тестовые примеры, их набор должен быть заранее определен, унифицирован и документирован.

По субъективным критериям CASE-средство должно оцениваться группой специалистов, использующих одни и те же критерии. Необходимый уровень опыта специалистов или групп должен быть заранее определен.

Результаты оценки должны быть стандартным образом документированы (для облегчения последующего использования) и, при необходимости, утверждены.

Отчет по результатам оценки должен содержать следующую информацию:

- введение. Общий обзор процесса и перечень основных результатов;
- предпосылки. Цель оценки и желаемые результаты, период времени, в течение которого выполнялась оценка, определение ролей и соответствующего опыта специалистов, выполнявших оценку;
- подход к оценке. Описание общего подхода, включая полученные CASE-средства, информацию, определяющую контекст и масштаб оценки, а также любые предположения и ограничения;
- информация о CASE-средствах. Она должна включать следующее: 1) наименование CASE-средства; 2) версию CASE-средства; 3) данные о поставщике, включая контактный адрес и телефон; 4) конфигурацию технических средств; 5) стоимостные данные; 6) описание CASE-средства, включающее поддерживаемые данным средством процессы создания и сопровождения ПО, программную среду CASE-средства (в частности, поддерживаемые языки программирования, операционные системы, совместимость с базами данных), функции CASE-средства, входные/выходные данные и область применения;
- этапы оценки. Конкретные действия, выполняемые в процессе оценки, должны быть описаны со степенью детализации, необходимой как для понимания масштаба и глубины оценки, так и для ее повторения при необходимости;
- конкретные результаты. Результаты оценки должны быть представлены в терминах критериев оценки. В тех случаях, когда отчет охватывает целый ряд CASE-средств или результаты данной оценки будут сопоставляться с аналогичными результатами других оценок, необходимо обратить особое внимание на формат представления результатов, способствующий такому сравнению. Субъективные результаты должны быть отделены от объективных и должны сопровождаться необходимыми пояснениями;
- выводы и заключения;
- приложения. Формулировка задачи оценки и уточненный список критериев.

4.2.3. Процесс выбора

Процессы оценки и выбора тесно взаимосвязаны друг с другом. По результатам оценки цели выбора и/или критерии выбора и их веса могут потребовать модификации. В таких случаях может потребоваться повторная оценка. Когда анализируются окончательные результаты оценки и к ним применяются критерии выбора, может быть рекомендовано приобретение CASE-средства или набора CASE-средств. Альтернативой может быть отсутствие адекватных CASE-средств, в этом случае рекомендуется разработать новое CASE-средство, модифицировать существующее или отказаться от внедрения.

Процесс выбора тесно взаимосвязан с процессом оценки и включает следующие действия:

- формулировка задач выбора, включая цели, предположения и ограничения;
- выполнение всех необходимых действий по выбору, включая определение и ранжирование критериев, определение средств-кандидатов, сбор необходимых данных и применение ранжированных критериев к результатам оценки для определения средств с наилучшими показателями. Для многих пользователей важным критерием выбора является интегрируемость CASE-средства с существующей средой;
- выполнение необходимого количества итераций с тем, чтобы выбрать (или отвергнуть) средства, имеющие сходные показатели;
- подготовка отчета по результатам выбора.

В процессе выбора возможно получение двух результатов:

- рекомендаций по выбору конкретного CASE-средства;
- запроса на получение дополнительной информации к процессу оценки.

Масштаб выбора должен устанавливать требуемый уровень детализации, необходимые ресурсы, график и ожидаемые результаты. Существует ряд параметров, которые могут быть использованы для определения масштаба, включая:

- использование предварительного отбора (например, отбор только средств, работающих на конкретной платформе);
- использование ранее полученных результатов оценки, результатов оценки из внешних источников или комбинации того и другого;

В том случае, если предыдущие оценки выполнялись с использованием различных наборов критериев или выполнялись с использованием конкретных критериев, но различными способами, результаты оценок должны быть представлены в согласованной форме. После завершения данного шага оценка каждого CASE-средства должна быть представлена в рамках единого набора критериев и должна быть непосредственно сопоставима с другими оценками.

Алгоритмы, обычно используемые для выбора, могут быть основаны на масштабе или ранге. Алгоритмы, основанные на масштабе, вычисляют единственное значение для каждого CASE-средства путем умножения веса каждого критерия на его значение (с учетом масштаба) и сложения всех произведений. CASE-средство с наивысшим результатом получает первый ранг. Алгоритмы, основанные на ранге, используют ранжирование CASE-средств - кандидатов по отдельным критериям или группам критериев в соответствии со значениями критериев в заданном масштабе. Затем, аналогично предыдущему, ранги сводятся вместе и вычисляются общие значения рангов.

При анализе результатов выбора предполагается, что процесс выбора завершен, CASE-средство выбрано и рекомендовано к использованию. Тем не менее, может потребоваться более точный анализ для определения степени зависимости значений ключевых критериев от различий в значениях характеристик CASE-средств - кандидатов. Такой анализ позволит определить, насколько результат ранжирования CASE-средств зависит от оптимальности выбора весовых коэффициентов критериев. Он также может использоваться для определения существенных различий между CASE-средствами с очень близкими значениями критериев или рангами.

Если ни одно из CASE-средств не удовлетворяет минимальным критериям, выбор (возможно, вместе с оценкой) может быть повторен для других CASE-средств - кандидатов.

Если различия между самыми предпочтительными кандидатами несущественны, дополнительная информация может быть получена путем повторного выбора (возможно, вместе с оценкой) с использованием дополнительных или других критериев.

Рекомендации по выбору должны быть строго обоснованы. В случае отсутствия адекватных CASE-средств, как было отмечено выше, рекомендуется разработать новое CASE-средство, модифицировать существующее или отказаться от внедрения.

4.2.4. Критерии оценки и выбора

Критерии формируют базис для процессов оценки и выбора и могут принимать различные формы, включая:

- числовые меры в широком диапазоне значений, например, объем требуемой памяти;
- числовые меры в ограниченном диапазоне значений, например, простота освоения,

выраженная в баллах от 1 до 5;

- двоичные меры (истина/ложь, да/нет), например, способность генерации документации в формате Postscript;
- меры, которые могут принимать одно или более из конечных множеств значений, например, платформы, для которых поддерживается CASE-средство.

Типичный процесс оценки и/или выбора может использовать набор критериев различных типов.

Структура набора критериев приведена на рисунке 4.3. Каждый критерий должен быть выбран и адаптирован экспертом с учетом особенностей конкретного процесса. В большинстве случаев только некоторые из множества описанных ниже критериев оказываются приемлемыми для использования, при этом также добавляются дополнительные критерии. Выбор и уточнение набора используемых критериев является критическим шагом в процессе оценки и/или выбора.

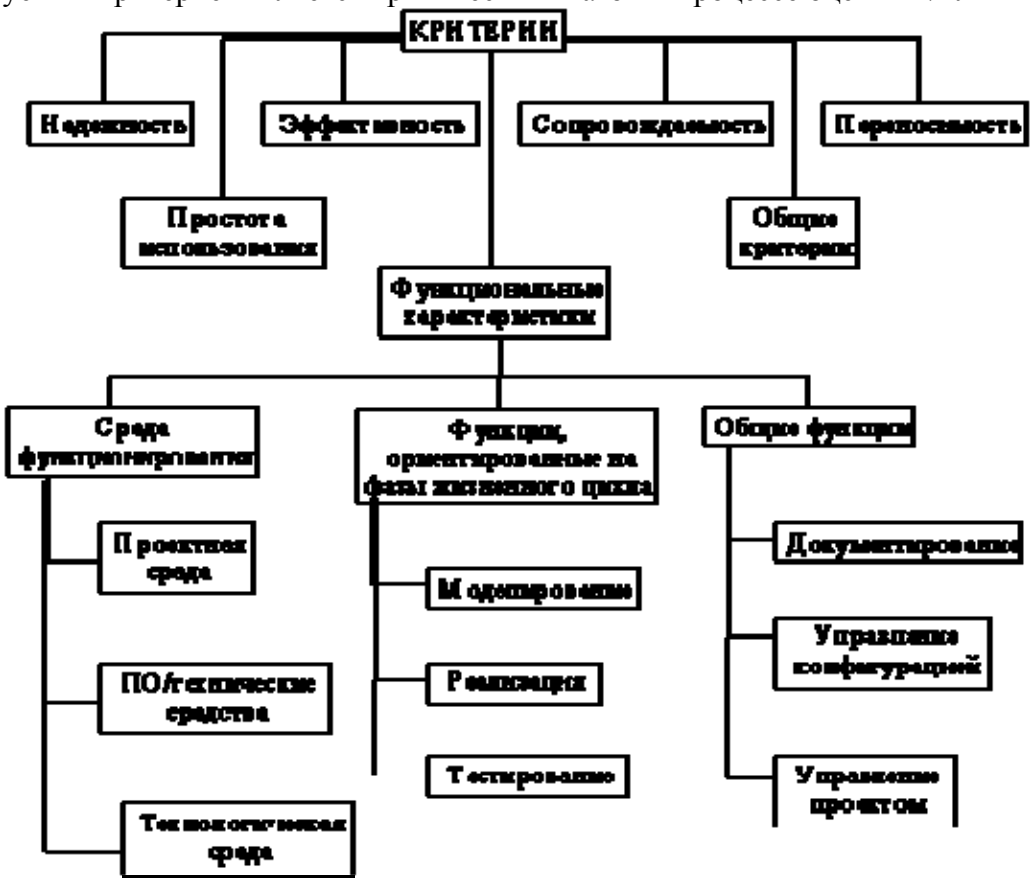


Рис. 4.3. Структура набора критериев

Функциональные характеристики

Критерии первого класса предназначены для определения функциональных характеристик CASE-средства. Они в свою очередь подразделяются на ряд групп и подгрупп.

1. Среда функционирования:
    a. Проектная среда:
        - *поддержка процессов жизненного цикла*. Определяет набор процессов ЖЦ, которые поддерживает CASE-средство. Примерами таких процессов являются анализ требований, проектирование, реализация, тестирование и оценка, сопровождение, обеспечение качества, управление конфигурацией и управление проектом, причем они зависят от принятой пользователем модели ЖЦ.
        - область применения. Примерами являются системы обработки транзакций, системы реального времени, информационные системы и т.д.
        - размер поддерживаемых приложений. Определяет ограничения на такие величины, как количество строк кода, уровней вложенности, размер базы данных, количество элементов данных, количество объектов конфигурационного управления.

b. ПО/технические средства:
   - требуемые технические средства. Оборудование, необходимое для функционирования CASE-средства, включая тип процессора, объем оперативной и дисковой памяти.
   - поддерживаемые технические средства. Элементы оборудования, которые могут использоваться CASE-средством, например, устройства ввода/вывода.
   - требуемое ПО. ПО, необходимое для функционирования CASE-средства, включая операционные системы и графические оболочки.
   - поддерживаемое ПО. Программные продукты, которые могут использоваться CASE-средством.

c. Технологическая среда:
   - соответствие стандартам технологической среды. Такие стандарты касаются языка, базы данных, репозитория, коммуникаций, графического интерфейса пользователя, документации, разработки, управления конфигурацией, безопасности, стандартов обмена информацией и интеграции по данным, по управлению и по пользовательскому интерфейсу.
   - совместимость с другими средствами. Способность к взаимодействию с другими средствами, включая непосредственный обмен данными (примерами таких средств являются текстовые процессоры, базы данных и другие CASE-средства). Возможность преобразования репозитория или его части в стандартный формат для обработки другими средствами.
   - поддерживаемая методология. Набор методов и методик, поддерживаемых CASE-средством. Примерами являются структурный или объектно-ориентированный анализ и проектирование.
   - поддерживаемые языки. Все языки, используемые CASE-средством. Примерами таких языков являются языки программирования (Кобол, Ада, С), языки баз данных и языки запросов (DDL, SQL), графические языки (Postscript, HPGL), языки спецификации проектных требований и интерфейсы операционных систем (языки управления заданиями).

2. Функции, ориентированные на фазы жизненного цикла:

a. Моделирование:

Данные критерии определяют способность выполнения функций, необходимых для спецификации требований к ПО и преобразованию их в проект:

- построение диаграмм. Возможность создания и редактирования диаграмм различных типов, представляющих интерес для пользователя. Наиболее распространенные типы диаграмм описаны в разделе 2.
- графический анализ. Возможность анализа графических объектов, а также хранения и представления проектной информации в графическом представлении. В большинстве случаев графические анализаторы интегрированы со средствами построения диаграмм.
- ввод и редактирование спецификаций требований и проектных спецификаций. К спецификациям такого рода относятся описания функций, данных, интерфейсов, структуры, качества, производительности, технических средств, среды, затрат и графиков.
- язык спецификации требований и проектных спецификаций. Возможность импорта, экспорта и редактирования спецификаций с использованием формального языка.
- моделирование данных. Возможность ввода и редактирования информации, описывающей элементы данных системы и их отношения.
- моделирование процессов. Возможность ввода и редактирования информации, описывающей процессы системы и их отношения.
- проектирование архитектуры ПО. Проектирование логической структуры ПО - структуры модулей, интерфейсов и др.
- имитационное моделирование. Возможность динамического моделирования

различных аспектов функционирования системы на основе спецификаций требований и/или проектных спецификаций, включая внешний интерфейс и производительность (например, время отклика, коэффициент использования ресурсов и пропускную способность).

- прототипирование. Возможность проектирования и генерации предварительного варианта всей системы или ее отдельных компонент на основе спецификаций требований и/или проектных спецификаций. Прототипирование в основном касается внешнего пользовательского интерфейса и осуществляется при непосредственном участии пользователей.
- генерация экранных форм. Возможность генерации экранных форм на основе спецификаций требований и/или проектных спецификаций.
- возможность трассировки. Возможность сквозного анализа функционирования системы от спецификации требований до конечных результатов (установления и отслеживания соответствий и связей между функциональными и другими внешними требованиями к ИС, техническими решениями и результатами проектирования). Прямая трассировка (проверка учета всех требований) и обратная трассировка (поиск проектных решений, не связанных ни с какими внешними требованиями).
- синтаксический и семантический контроль проектных спецификаций. Контроль синтаксиса диаграмм и типов их элементов, контроль декомпозиции функций, проверка спецификаций на полноту и непротиворечивость.
- другие виды анализа. Конкретные дополнительные виды анализа могут включать алгоритмы, потоки данных, нормализацию данных, использование данных, пользовательский интерфейс.
- автоматизированное проектирование отчетов.

b. Реализация:

Реализация затрагивает функции, связанные с созданием исполняемых элементов системы (программных кодов) или модификацией существующей системы. Многие из перечисленных ниже критериев зависят от конкретных языков и включают следующие:

- синтаксически управляемое редактирование. Возможность ввода и редактирования исходных кодов на одном или нескольких языках с одновременным синтаксическим контролем.
- генерация кода. Возможность генерации кодов на одном или нескольких языках на основе проектных спецификаций. Типы генерируемого кода могут включать обычный программный код, схему базы данных, запросы, экраны/меню.
- компиляция кода.
- конвертирование исходного кода. Возможность преобразования кода из одного языка в другой.
- анализ надежности. Возможность количественно оценивать параметры надежности ПО, такие, как количество ошибок и др.
- реверсный инжиниринг. Возможность анализа существующих исходных кодов и формирования на их основе проектных спецификаций.
- реструктуризация исходного кода. Возможность модификации формата и/или структуры существующего исходного кода.
- анализ исходного кода. Примерами такого анализа могут быть определение размера кода, вычисление показателей сложности, генерация перекрестных ссылок и проверка на соответствие стандартам.
- отладка. Типичные функции отладки - трассировка программ, выделение узких мест и наиболее часто используемых фрагментов кода и т.д.

c. Тестирование:

Критерии тестирования включают следующие:

- описание тестов. Типичные возможности включают генерацию тестовых данных, алгоритмов тестирования, требуемых результатов и т.д.

- фиксация и повторение действий оператора. Возможность фиксировать данные, вводимые оператором с помощью клавиатуры, мыши и т.д., редактировать их и воспроизводить в тестовых примерах.
- автоматический запуск тестовых примеров.
- регрессионное тестирование. Возможность повторения и модификации ранее выполненных тестов для определения различий в системе и/или среде.
- автоматизированный анализ результатов тестирования. Типичные возможности включают сравнение ожидаемых и реальных результатов, сравнение файлов, статистический анализ результатов и др.
- анализ тестового покрытия. Оснащенность средствами контроля исходного кода и анализ тестового покрытия. Проверяются, в частности, обращения к операторам, процедурам и переменным.
- анализ производительности. Возможность анализа производительности программ. Анализируемые параметры производительности могут включать использование центрального процессора, памяти, обращения к определенным элементам данных и/или сегментам кода, временные характеристики и т.д.
- анализ исключительных ситуаций в процессе тестирования.
- динамическое моделирование среды. В частности, возможность автоматически генерировать моделируемые входные данные системы.

3. Общие функции:

Приведенные ниже критерии определяют функции CASE-средств, охватывающие всю совокупность фаз ЖЦ. Поддержка всех этих функций осуществляется посредством репозитория.

a. Документирование:
   - редактирование текстов и графики. Возможность вводить и редактировать данные в текстовом и графическом формате.
   - редактирование с помощью форм. Возможность поддерживать формы, определенные пользователями, вводить и редактировать данные в соответствии с формами.
   - возможности издательских систем.
   - поддержка функций и форматов гипертекста.
   - соответствие стандартам документирования.
   - автоматическое извлечение данных из репозитория и генерация документации по спецификациям пользователя.
b. Управление конфигурацией:
   - контроль доступа и изменений. Возможность контроля доступа на физическом уровне к элементам данных и контроля изменений. Контроль доступа включает возможности определения прав доступа к компонентам, а также извлечения элементов данных для модификации, блокировки доступа к ним на время модификации и помещения обратно в репозиторий.
   - отслеживание модификаций. Фиксация и ведение журнала всех модификаций, внесенных в систему в процессе разработки или сопровождения.
   - управление версиями. Ведение и контроль данных о версиях системы и всех ее коллективно используемых компонентах.
   - учет состояния объектов конфигурационного управления. Возможность получения отчетов о всех последовательных версиях, содержимом и состоянии различных объектов конфигурационного управления.
   - генерация версий и модификаций. Поддержка пользовательского описания последовательности действий, требуемых для формирования версий и модификаций, и автоматическое выполнение этих действий.
   - архивирование. Возможность автоматического архивирования элементов данных для последующего использования.
c. Управление проектом:
   - управление работами и ресурсами. Контроль и управление процессом

проектирования ИС в терминах структуры заданий и назначения исполнителей, последовательности их выполнения, завершенности отдельных этапов проекта и проекта в целом. Возможность поддержки плановых данных, фактических данных и их анализа. Типичные данные включают графики (с учетом календаря, рабочих часов, выходных и др.), компьютерные ресурсы, распределение персонала, бюджет и др.

- оценка. Возможность оценивать затраты, график и другие проектные параметры, вводимые пользователями.
- управление процедурой тестирования. Поддержка управления процедурами и программой тестирования, например, управления расписанием планируемых процедур, фиксация и запись результатов тестирования, генерация отчетов и т.д.
- управление качеством. Ввод соответствующих данных, их анализ и генерация отчетов.
- корректирующие действия. Поддержка управления корректирующими действиями, включая обработку сообщений о проблемных ситуациях.

4.2.4.1. Надежность

- администрирование репозитория. Контроль и обеспечение целостности проектных данных.
- автоматическое резервирование (определяемое поставщиком или планируемое пользователем).
- безопасность. Защита от несанкционированного доступа.
- обработка ошибок. Обнаружение ошибок в работе системы, извещение пользователя, корректное завершение работы или сохранение состояния к моменту прерывания.
- анализ отказов в критических приложениях.

4.2.4.2. Простота использования

- удобство пользовательского интерфейса. Удобство расположения и представления часто используемых элементов экрана, способов ввода данных и др.
- локализация (в соответствии с требованиями данной страны).
- простота освоения. Трудовые и временные затраты на освоение средств.
- адаптируемость к конкретным требованиям пользователя. Адаптируемость к различным алфавитам, режимам текстового и графического представления (слева-направо, сверху-вниз), различным форматам даты, способам ввода/вывода (экранным формам и форматам), изменениям в методологии (изменениям графических нотаций, правил, свойств и состава предопределенных объектов) и др.
- качество документации (полнота, понятность, удобочитаемость, полезность и др.).
- доступность и качество учебных материалов. Они могут включать компьютерные учебные материалы, учебные пособия, курсы.
- требования к уровню знаний. Квалификация и опыт, необходимые для эффективного использования CASE-средств.
- простота работы с CASE-средством (как для начинающих, так и для опытных пользователей).
- унифицированность пользовательского интерфейса (по отношению к другим средствам, использующимся в данной организации).
- онлайновые подсказки (полнота и качество).
- качество диагностики (понятность и полезность диагностических сообщений для пользователя).
- допустимое время реакции на действия пользователя (в зависимости от среды).
- простота установки и обновления версий.

4.2.4.3. Эффективность

- требования к техническим средствам. Требования к оптимальному размеру внешней и

оперативной памяти, типу и производительности процессора, обеспечивающим приемлемый уровень производительности.

- эффективность рабочей нагрузки. Эффективность выполнения CASE-средством своих функций в зависимости от интенсивности работы пользователя (например, количество нажатий клавиш или кнопки мыши, требуемое для выполнения определенных функций).
- производительность. Время, затрачиваемое CASE-средством для выполнения конкретных задач (например, время ответа на запрос, время анализа 100000 строк кода). В некоторых случаях данные оценки производительности можно получить из внешних источников.

4.2.4.4. Сопровождаемость

- уровень поддержки со стороны поставщика (скорость разрешения проблем, поставки новых версий, обеспечение дополнительных возможностей).
- трассируемость обновлений (простота освоения отличий новых версий от существующих).
- совместимость обновлений (совместимость новых версий с существующими, включая, например, совместимость по входным или выходным данным).
- сопровождаемость конечного продукта (простота внесения изменений в ПО и документацию).

4.2.4.5. Переносимость

- совместимость с версиями ОС (возможность работы в среде различных версий одной и той же ОС, простота модификации CASE-средства для работы с новыми версиями ОС).
- переносимость данных между различными версиями CASE-средства.
- соответствие стандартам переносимости. Такие стандарты включают документацию, коммуникации и пользовательский интерфейс, оконный интерфейс, языки программирования, языки запросов и др.

4.2.4.6. Общие критерии

Приведенные ниже критерии являются общими по своей природе и не принадлежат к совокупности показателей качества, приведенной в стандарте ISO/IEC 9126: 1991.

- затраты на CASE-средство. Включают стоимость приобретения, установки, начального сопровождения и обучения. Следует учитывать цену для всех необходимых конфигураций (включая единственную копию, несколько копий, локальную лицензию, лицензию для предприятия, сетевую лицензию).
- оценочный эффект от внедрения CASE-средства (уровень продуктивности, качества и т.д.). Такая оценка может потребовать экономического анализа.
- профиль дистрибьютора. Общие показатели возможностей дистрибьютора. Профиль дистрибьютора может включать величину его организации, стаж в бизнесе, финансовое положение, список любых дополнительных продуктов, деловые связи (в частности, с другими дистрибьюторами данного средства), планируемая стратегия развития.
- сертификация поставщика. Сертификаты, полученные от специализированных организаций в области создания ПО (например, SEI и ISO), удостоверяющие, что квалификация поставщика в области создания и сопровождения ПО удовлетворяет некоторым минимально необходимым или вполне определенным требованиям. Сертификация может быть неформальной, например, на основе анализа качества работы поставщика.
- лицензионная политика. Доступные возможности лицензирования, право копирования (носителей и документации), любые ограничения и/или штрафные санкции за вторичное использования (подразумевается продажа пользователем CASE-средства продуктов, в состав которых входят некоторые компоненты CASE-средства, использовавшиеся при разработке продуктов).
- экспортные ограничения.
- профиль продукта. Общая информация о продукте, включая срок его существования, количество проданных копий, наличие, размер и уровень деятельности пользовательской

группы, система отчетов о проблемах, программа развития продукта, совокупность применений, наличие ошибок и др.

- поддержка поставщика. Доступность, реактивность и качество услуг, предоставляемых поставщиком для пользователей CASE-средств. Такие услуги могут включать телефонную "горячую линию", местную техническую поддержку, поддержку в самой организации.
- доступность и качество обучения. Обучение может проводиться на территории поставщика, пользователя или где-либо в другом месте.
- адаптация, требуемая для внедрения CASE-средств в организации пользователя. Примером может быть определение способа использования централизованного CASE-средства с единой, общей БД в распределенной среде.

4.2.5. Пример подхода к определению критериев выбора CASE-средств

Пример подхода к определению критериев выбора CASE-средств приведен в [1]. Предполагается, что CASE-средства будут использованы в крупном типовом проекте ИС, обладающем характеристиками, перечисленными во введении. В общем случае стратегия выбора CASE-средств для конкретного применения зависит от целей, потребностей и ограничений будущего проекта ИС (включая квалификацию участвующих в процессе проектирования специалистов), которые, в свою очередь, определяют используемую методологию проектирования.

Следует подчеркнуть, что определяющим фактором при выборе тех или иных инструментальных средств является используемая методология и технология проектирования, а не наоборот. С этой точки зрения бессмысленно сравнивать CASE-средства сами по себе в отрыве от методологии, поскольку ИС можно в принципе разработать любыми средствами.

Традиционно при обсуждении проблемы выбора CASE-средств большое внимание уделялось особенностям реализации той или иной методологии анализа предметной области (E-R, IDEF0, IDEF1X, Gane/Sarson, Yourdon, Barker и др.). Безусловно, богатство изобразительных и описательных средств дает возможность на этапах стратегического планирования и анализа построить наиболее полную и адекватную модель предметной области. С другой стороны, если говорить о конечных результатах - базах данных и приложениях, то обнаруживается, что часть описаний в них практически не отражается, оставаясь чисто декларативной (на выходе мы в любом случае получим описание БД в табличном представлении с минимальным набором ограничений целостности и исполнимый код приложений, большую часть которых составляют экранные формы, не выводимые непосредственно из моделей предметной области). Опытные аналитики и проектировщики всегда с большими или меньшими трудозатратами придут к нужному конечному результату независимо от того, какая конкретно методология или ее разновидность реализована в данном инструменте. Это, конечно, не означает, что методология не важна, напротив, отсутствие или неполнота описательных средств могут с самого начала значительно затруднить работу над проектом. Однако, зачастую на первом плане оказываются другие критерии, невыполнение которых может породить гораздо большие трудности.

Как было отмечено в подразделе 1.3, технология проектирования должна быть поддержана комплексом согласованных CASE-средств, обеспечивающих автоматизацию процессов, выполняемых на всех стадиях ЖЦ. Может создаться впечатление, что если можно сформировать необходимую аппаратную платформу из компонентов различных фирм-производителей, то так же просто можно выбрать и скомплексировать разные инструментальные средства, каждое из которых является одним из мировых лидеров в своем классе. Однако для инструментальных средств в настоящее время, в отличие от оборудования, отсутствуют международные стандарты на основные свойства конечных продуктов (программ, баз данных и их сопряжение). Поскольку составные части проекта должны быть интегрированы в единый продукт, следовательно, имеет смысл рассматривать не любые, а только сопряженные инструментальные средства, которые в принципе могут быть ориентированы - даже внутри одного класса - на разные методологии; при этом необходимо отбирать в состав комплекса CASE-средств средства, поддерживающие по крайней мере близкие методологии, если не одну и ту же.

Исходя из перечисленных выше соображений, в качестве основных критериев выбора CASE-средств принимаются следующие критерии:

1. Поддержка полного жизненного цикла ИС с обеспечением эволюционности ее развития

Полный жизненный цикл ИС должен поддерживаться комплексом инструментальных средств, перечисленных в разделе 3.2, с учетом следующих особенностей:

- коллективного характера разработки моделей, проектных спецификаций и приложений территориально распределенными группами специалистов с использованием различных инструментальных средств (включая их интеграцию, тестирование и отладку);
- необходимости адаптации типового проекта к различным системно-техническим платформам (техническим средствам, операционным системам и СУБД) и организационно-экономическим особенностям объектов внедрения;
- необходимости интеграции с существующими разработками (включая реинжиниринг приложений и конвертирование БД). Для существующих ИС должен обеспечиваться плавный переход из старой среды эксплуатации в новую с минимальными переделками и поддержкой эксплуатируемых баз данных и приложений, внедренных до начала работ по созданию новой системы.

2. Обеспечение целостности проекта и контроля за его состоянием

Данное требование означает наличие единой технологической среды создания, сопровождения и развития ИС, а также целостность репозитория. Единая технологическая среда должна обеспечиваться за счет использования единственного CASE-средства для поддержки моделей ИС, а также за счет наличия программно-технологических интерфейсов между отдельными инструментальными средствами, сертифицированных и поддерживаемых фирмами-разработчиками соответствующих средств. В частности, интерфейс между CASE-средствами и средствами разработки приложений должен выполнять две основные функции: а) непосредственный переход в рамках единой среды от описания логики приложения, реализованного CASE-средством, к разработке пользовательского интерфейса (экранных форм); б) перенос описания БД из репозитория CASE-средства в репозиторий средства разработки приложений и обратно. Вся информация о проекте должна автоматически помещаться в базу проектных данных, при этом должны поддерживаться согласованность, непротиворечивость, полнота и минимальная избыточность проекта, а также корректность операций его редактирования. Это может быть достигнуто при условии исключения или существенного ограничения возможности актуализации репозитория разными средствами. В рамках CASE-средства должен обеспечиваться контроль соответствия декомпозиций диаграмм, а также контроль соответствия диаграмм различных типов (например, диаграмм потоков данных и ER-диаграмм).

Невыполнение требования целостности в условиях разобщенности разработчиков и временной протяженности крупного проекта может означать утрату контроля за его состоянием.

3. Независимость от программно-аппаратной платформы и СУБД

Требование определяется неоднородностью среды функционирования ИС. Такая независимость может иметь две составляющих: независимость среды разработки и независимость среды эксплуатации приложений. Она обеспечивается за счет наличия совместимых версий CASE-средств для различных платформ и драйверов соответствующих сетевых протоколов, менеджеров транзакций и СУБД.

4. Поддержка одновременной работы групп разработчиков

Развитые CASE-средства должны обладать возможностями разделения полномочий персонала разработчиков и объединения отдельных работ в общий проект. Должна обеспечиваться одновременная (в заданной сетевой конфигурации) работа проектировщиков БД и разработчиков приложений (разработчики приложений в такой ситуации могут начинать работу с базой данных, не дожидаясь полного завершения ее проектирования CASE-средствами). При этом все группы специалистов должны быть обеспечены адекватным инструментарием, а внесение изменений в проект различными разработчиками должно быть согласованным и корректным. Каждый разработчик должен иметь возможность работы со своим личным репозиторием, являющимся фрагментом или копией общего репозитория. Должны обеспечиваться

содержательная интеграция всех изменений, вносимых разработчиками, в общем репозитории, одновременная доступность для разработчика общего и личного репозиториев и простота переноса объектов между ними.

5. Возможность разработки приложений "клиент-сервер" требуемой конфигурации

Подразумевается сочетание наличия развитой графической среды разработки приложений (многооконность, разнообразие стандартных графических объектов, разнообразие используемых шрифтов и т.д.) с возможностью декомпозиции (partitioning) приложения на "клиентскую" часть, реализующую пользовательский экранный интерфейс и "серверную" часть. При этом должна обеспечиваться возможность перемещения отдельных компонентов приложения между "клиентом" и "сервером" на наиболее подходящую платформу, обеспечивающую максимальную эффективность функционирования приложения в целом, а также возможность использования сервера приложений (менеджера транзакций).

6. Открытая архитектура и возможности экспорта/импорта

Открытая и общедоступная информация об используемых форматах данных и прикладных программных интерфейсах должна позволять интегрировать инструментальные средства третьих фирм и относительно безболезненно переходить от одной системы к другой. Возможности экспорта/импорта означают, что спецификации, полученные на этапах анализа, проектирования и реализации для одной ИС, могут быть использованы для проектирования другой ИС. Повторное проектирование и реализация могут быть обеспечены при помощи средств экспорта/импорта спецификаций в различные CASE-средства.

7. Качество технической поддержки в России, стоимость приобретения и поддержки, опыт успешного использования

Имеется в виду наличие квалифицированных дистрибьюторов и консультантов, быстрота обслуживания пользователей, высокое качество технической поддержки и обучения продукту и методологии его применения для больших коллективов разработчиков (наличие сведений о практике использования системы, качество документации, укомплектованность примерами и обучающими курсами, наличие пилотных проектов). Затраты на обучение новым технологиям значительны, однако потери от использования современных сложных технологий необученными специалистами могут оказаться значительно выше.

Кроме того, фирмы-поставщики инструментальных средств должны быть устойчивыми, так как технология выбирается не на один год, а также должны обеспечивать хорошую поддержку на территории России (горячая линия, консультации, обучение, консалтинг), возможно, через дистрибьюторов.

Что касается стоимости, следует учитывать возможность получения бесплатной временной лицензии, стоимость лицензии на одно рабочее место CASE-средств, скидки, предоставляемые фирмой в случае приобретения большого количества лицензий, необходимость приобретения run-time версий для эксплуатации приложений и т.д. В то же время стоимость продукта должна рассматриваться не сама по себе, а с учетом ее соответствия возможностям продукта.

8. Простота освоения и использования

Учитываются следующие характеристики:

- соответствие инструмента особенностям и потенциальным возможностям коллектива разработчиков;
- доступность пользовательского интерфейса;
- время, необходимое для обучения;
- простота установки;
- качество документации;
- объем ручного труда при сопровождении ИС.

9. Обеспечение качества проектной документации

Это требование относится к возможностям CASE-средств анализировать и проверять описания и документацию на полноту и непротиворечивость, а также на соответствие принятым в

данной методологии стандартам и правилам (включая ГОСТ, ЕСПД). В результате анализа должна формироваться информация, указывающая на имеющиеся противоречия или неполноту в проектной документации. Должна быть также обеспечена возможность создавать новые формы документов, определяемые пользователями.

10. Использование общепринятых, стандартных нотаций и соглашений

Для того, чтобы проект мог выполняться разными коллективами разработчиков, необходимо использование стандартных методов моделирования и стандартных нотаций, которые должны быть оформлены в виде нормативов до начала процесса проектирования. Несоблюдение проектных стандартов ставит разработчиков в зависимость от фирмы-производителя данного средства, делает затруднительным формальный контроль корректности проектных решений и снижает возможности привлечения дополнительных коллективов разработчиков, смены исполнителей и отчуждения проекта, поскольку число специалистов, знакомых с данным методом (нотацией) может быть ограниченным.

В результате выполненного анализа может оказаться, что ни одно доступное средство не удовлетворяет в нужной мере всем основным критериям и не покрывает все потребности проекта. В этом случае может применяться набор средств, позволяющий построить на их базе единую технологическую среду.

4.3. Выполнение пилотного проекта

Перед полномасштабным внедрением выбранного CASE-средства в организации выполняется пилотный проект, целью которого является экспериментальная проверка правильности решений, принятых на предыдущих этапах, и подготовка к внедрению.

Пилотный проект представляет собой первоначальное реальное использование CASE-средства в предназначенной для этого среде и обычно подразумевает более широкий масштаб использования CASE-средства по отношению к тому, который был достигнут во время оценки. Пилотный проект должен обладать многими из характеристик реальных проектов, для которых предназначено данное средство. Он преследует следующие цели:

- подтвердить достоверность результатов оценки и выбора;
- определить, действительно ли CASE-средство годится для использования в данной организации, и если да, то определить наиболее подходящую область его применения;
- собрать информацию, необходимую для разработки плана практического внедрения;
- приобрести собственный опыт использования CASE-средства.

Пилотный проект позволяет получить важную информацию, необходимую для оценки качества функционирования CASE-средства и его поддержки со стороны поставщика после того, как средство установлено.

Важной функцией пилотного проекта является принятие решения относительно приобретения или отказа от использования CASE-средства. Провал пилотного проекта позволяет избежать более значительных и дорогостоящих неудач в дальнейшем, поскольку пилотный проект обычно связан с приобретением относительно небольшого количества лицензий и обучением узкого круга специалистов.

Первоначальное использование новой CASE-технологии в пилотном проекте должно тщательно планироваться и контролироваться. Пилотный проект включает следующие шаги (рисунок 4.4).

Определение характеристик пилотного проекта

Пилотный проект должен обладать следующими характеристиками:

- Область применения. Чтобы облегчить окончательное определение области применения CASE-средства, предметная область пилотного проекта должна быть типичной для обычной деятельности организации. Пилотный проект должен помочь определить любую дополнительную технологию, обучение или поддержку, которые необходимы для перехода от пилотного проекта к широкомасштабному использованию средства. В рамках этих ограничений пилотный проект должен иметь небольшой, но значимый размер.
- Масштабируемость. Результаты, полученные в пилотном проекте, должны показать

масштабируемость средства. Цель - получить четкое представление о масштабах проектов, для которых данное средство применимо.

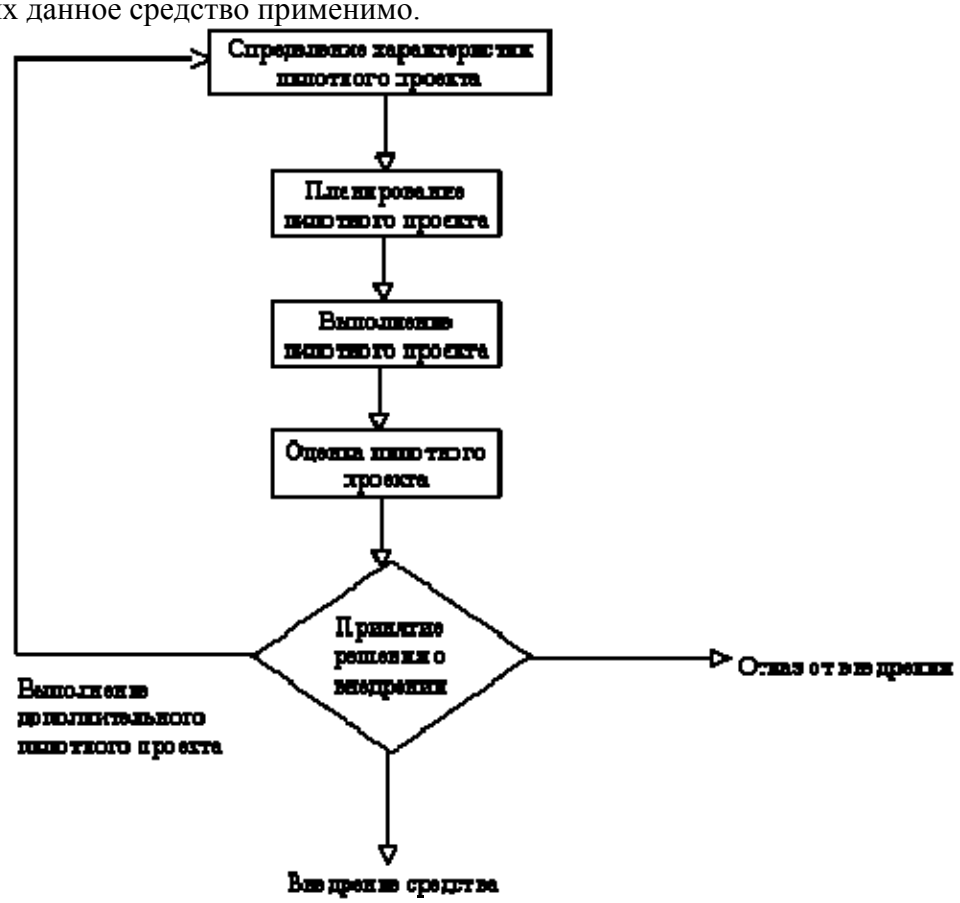


Рис. 4.4. Шаги пилотного проекта

- Представительность. Пилотный проект не должен быть необычным или уникальным для организации. CASE-средство должно использоваться для решения задач, относящихся к предметной области, хорошо понимаемой всей организацией.
- Критичность. Пилотный проект должен иметь существенную значимость, чтобы оказаться в центре внимания, но не должен быть критичным для успешной деятельности организации в целом. Необходимо осознавать, что первоначальное внедрение новой технологии подразумевает определенный риск. При выборе пилотного проекта приходится решать следующую дилемму: успех незначительного проекта может остаться незамеченным, с другой стороны, провал значимого проекта может вызвать чрезмерную критику.
- Авторитетность. Группа специалистов, участвующих в проекте, должна обладать высоким авторитетом, при этом результаты проекта будут всерьез восприняты остальными сотрудниками организации.
- Характеристики проектной группы. Проектная группа должна обладать готовностью к нововведениям, технической зрелостью и приемлемым уровнем опыта и знаний в данной технологии и предметной области. С другой стороны, группа должна отражать в миниатюре характеристики всей организации в целом.

В большинстве случаев существует баланс между желанием реализовать идеальный пилотный проект и реальными ограничениями организации. Организация должна выбрать пилотный проект таким образом, чтобы, во-первых, способ использования CASE-средства в нем совпадал с дальнейшими планами, и, во-вторых, перечисленные выше характеристики были сбалансированы с реальными условиями организации.

Кроме того, организация должна учитывать продолжительность пилотного проекта (и в целом процесса внедрения). Слишком продолжительный проект связан с риском потери интереса к нему со стороны руководства.

Планирование пилотного проекта

Планирование пилотного проекта должно по возможности вписываться в обычный процесс планирования проектов в организации. План должен содержать следующую информацию:

- цели, задачи и критерии оценки;
- персонал;
- процедуры и соглашения;
- обучение;
- график и ресурсы.

Цели, задачи и критерии оценки

Ожидаемые результаты пилотного проекта должны быть четко определены. Степень соответствия этим результатам представляет собой основу для последующей оценки проекта. Для определения целей, задач и критериев оценки необходимо выполнить следующие действия:

- описать проект в терминах ожидаемых результатов (т.е. конечного продукта). Описание должно включать форму представления и содержание результатов. Должны быть четко определены договорные требования и соответствующие стандарты.
- определить общие цели проекта. Примером цели может быть определение степени улучшения качества проектной документации в результате применения CASE-средств.
- определить конкретные задачи, реализующие поставленные цели. Каждой цели можно поставить в соответствие одну или несколько конкретных задач с количественно оцениваемыми результатами. Примером такой задачи может быть сравнительный анализ качества документации, полученной с помощью CASE-средства и без него. Документация может включать спецификацию требований к ПО, высокоуровневые и детальные проектные спецификации.
- определить критерии оценки результатов. Чтобы определить степень успеха пилотного проекта, необходимо использовать набор критериев, основанных на упомянутых выше задачах. Примером критерия может быть степень непротиворечивости проектной документации и контролируемости выполнения требований к ПО. Значения критериев должны сравниваться с базовыми значениями, полученными до выполнения пилотного проекта.

Персонал

Специалисты, выбранные для участия в пилотном проекте, должны иметь соответствующий авторитет и влияние и быть сторонниками новой технологии. Группа должна включать как технических специалистов, так и менеджеров, заинтересованных в новой технологии и разбирающихся в ее использовании. Группа должна обладать высокими способностями к коммуникации, знанием особенностей организационных процессов и процедур, а также предметной области. Группа не должна, тем не менее, состоять полностью из специалистов высшего звена, она должна представлять средний уровень организации.

Многие CASE-средства обеспечивают возможности, связанные с генерацией проектной документации и конфигурационным управлением. Специалисты, связанные с этими и другими смежными аспектами разработки и сопровождения ПО, также должны быть включены в состав группы.

После завершения пилотного проекта группа должна быть открыта для обмена информацией с остальными специалистами организации относительно возможностей нового средства и опыта, полученного при его использовании. Может оказаться желательным рассредоточить членов проектной группы по всей организации с целью распространения их опыта и знаний.

Процедуры и соглашения

Необходимо четко определить процедуры и соглашения, регулирующие использование CASE-средств в пилотном проекте. Эта задача скорее всего может оказаться более долгой и сложной, чем ожидается, при этом может оказаться необходимым привлечение сторонних экспертов. Примерами процедур и соглашений, которые могут повлиять на успех пилотного проекта, являются методология, технические соглашения (в частности, по наименованиям и структуре каталогов, стандарты проектирования и программирования - см. подраздел 1.3) и

организационные соглашения (в частности, учет использования ресурсов, авторизация, контроль изменений, процедуры экспертизы и подготовки отчетов, стандарты проверки качества).

В пилотном проекте по возможности должны использоваться принятые в организации процедуры и соглашения. С другой стороны, в течение пилотного проекта процедуры и соглашения имеют тенденцию к развитию и совершенствованию по мере накопления опыта применения средства. Существующие процедуры и соглашения могут оказаться неэффективными или чересчур ограничивающими. При этом те изменения, которые предлагается в них вносить, должны документироваться.

Обучение

Должны быть определены виды и объем обучения, необходимого для выполнения пилотного проекта. При планировании обучения нужно иметь в виду три вида потребностей: технические, управленческие и мотивационные. Ресурсы, требуемые для обучения (учебные аудитории и оборудование, преподаватели и учебные материалы), должны соответствовать плану пилотного проекта.

График обучения должен определять как специалистов, подлежащих обучению, так и виды обучения, которое они должны пройти. Обучение, которое проводится в период выполнения проекта, должно начинаться как можно быстрее после начала проекта. Обучение средствам, процессам или методам, которые не будут использоваться в течение нескольких месяцев после начала проекта, должно планироваться на то время, когда в них возникнет реальная потребность.

Поставщики CASE-средств обычно предлагают обучение использованию поставляемых ими средств. Помимо этого, для некоторых средств может быть необходимо обучение методологии. Некоторые виды обучения должны выполняться собственными силами. Такие виды обучения включают использование CASE-средства в контексте процессов, происходящих в организации, а также в совокупности с другими средствами в данной среде. Часть плана пилотного проекта, связанная с обучением, должна использоваться в качестве входа для плана практического внедрения.

При выборе необходимого обучения должны приниматься во внимание следующие факторы:

- квалификация преподавателей;
- соответствие обучения характеристикам конкретных групп специалистов (например, обзорные курсы для менеджеров, подробные курсы для разработчиков);
- возможность проведения курсов непосредственно на рабочих местах;
- возможность проведения расширенных курсов;
- возможность подготовки собственных преподавателей.

График и ресурсы

Должен быть разработан график, включающий ресурсы и сроки (этапы) проведения работ. Ресурсы включают персонал, технические средства, ПО и финансирование. Данные о персонале могут определять конкретных специалистов или требования к квалификации, необходимой для успешного выполнения пилотного проекта. Финансирование должно определяться отдельно по каждому виду работ: приобретение CASE-средств, установка, обучение, отдельные этапы проектирования.

Выполнение пилотного проекта

Пилотный проект должен выполняться в соответствии с планом. Организационная деятельность, связанная с выполнением пилотного проекта и подготовкой отчетов, должна выполняться в установленном порядке. Пилотная природа проекта требует специального внимания к вопросам приобретения, поддержки, экспертизы и обновления версий. Эти вопросы рассматриваются ниже.

Приобретение, установка и интеграция

После того, как CASE-средство выбрано, оно должно быть приобретено, интегрировано в проектную среду и настроено в соответствии с требованиями пилотного проекта. Границы этой деятельности зависят от тех действий, которые имели место в процессе оценки и выбора, а также от степени модификации средства, необходимой для его использования в проекте.

Процесс приобретения может включать подготовку контракта, переговоры, лицензирование и другую деятельность, которая выходит за рамки данных рекомендаций. Эта

деятельность требует затрат времени и человеческих ресурсов, которые должны быть учтены при планировании. План должен предусматривать возможность отказа от выбранного средства на данном этапе из-за договорных разногласий.

После того, как процесс приобретения завершен, средство должно быть установлено, оттестировано и принято в эксплуатацию. Тестирование позволяет убедиться, что поставленный продукт соответствует требованиям контракта, обладает необходимой полнотой и корректностью. Этап приемки может быть предусмотрен контрактом, его реальный срок может отличаться от того, который был предусмотрен первоначально в плане пилотного проекта. Особое внимание необходимо уделить соблюдению всех требований поставщика к параметрам среды функционирования CASE-средства.

После завершения приемки может потребоваться настройка и интеграция. Настройка может включать модификацию интерфейсов, связанную с требованиями специалистов проектной группы, а также установкой прав доступа и привилегий. Настройка должна оставаться в рамках тех возможностей, которые предоставляет само средство. Не следует заниматься модификацией готовых продуктов на уровне исходных кодов.

Если новое средство должно использоваться в совокупности с некоторыми другими средствами, необходимо определить взаимодействие средств и требуемую интеграцию. Для интеграции новых средств с существующими может потребоваться построение специальных оболочек. Сложная интеграция может потребовать привлечения сторонних экспертов.

Поддержка

Доступная поддержка должна включать (по соглашению) "горячую линию" поставщика и поддержку местного поставщика, поддержку в самой организации, контакты с опытными пользователями в других организациях и участие в работе групп пользователей.

Внутренняя поддержка должна осуществляться специалистами, знакомыми с установкой средств и работой с ними. Существует несколько возможных вариантов получения такой поддержки (например, от специалиста данной организации, имеющего опыт предшествующей работы со средством; участников процесса оценки и выбора или опытного консультанта). Такой тип поддержки должен специальным образом планироваться и администрироваться. Особое внимание должно быть уделено средствам, работающим в сетях или обладающих репозиториями, поддерживающими многопользовательскую работу.

Периодические экспертизы

Обычные процедуры экспертизы проектов, существующие в организации, должны выполняться и для пилотного проекта, при этом особое внимание должно уделяться именно пилотным аспектам проекта. Помимо этого, результаты экспертиз должны служить мерой успешного использования CASE-средств.

Обновление версий

Пользователи CASE-средства могут ожидать периодического обновления версий со стороны поставщика в течение выполнения пилотного проекта. При этом необходимо тщательное отношение к интеграции этих версий. Следует заранее оценить влияние этих обновлений на ход проекта. Новые версии могут как обеспечить новые возможности, так и породить новые проблемы. Например, новая версия может потребовать видоизмененного или дополнительного обучения, а также может оказать отрицательное воздействие на уже выполненную к этому моменту работу.

Оценка пилотного проекта

После завершения пилотного проекта его результаты необходимо оценить и сопоставить их с изначальными потребностями организации, критериями успешного внедрения CASE-средств, базовыми метриками и критериями успеха пилотного проекта. Такая оценка должна установить возможные проблемы и важнейшие характеристики пилотного проекта, которые могут повлиять на пригодность CASE-средства для организации. Она должна также указать проекты или структурные подразделения внутри организации, для которых данное средство является подходящим. Помимо этого, оценка может дать информацию относительно совершенствования процесса внедрения в дальнейшем.

В процессе оценки пилотного проекта организация должна определить свою позицию по следующим трем вопросам:

- Целесообразно ли внедрять CASE-средство ?
- Какие конкретные особенности пилотного проекта привели к его успеху (или неудаче) ?
- Какие проекты или подразделения в организации могли бы получить выгоду от использования средств ?

Принятие решения о целесообразности внедрения CASE-средств

На данном этапе процесса внедрения организация должна сделать существенные инвестиции в CASE-средства. Если средства удовлетворили или даже превысили ожидания организации, то решение о внедрении может быть принято достаточно просто и быстро.

С другой стороны, может оказаться, что в рамках пилотного проекта средства не оправдали тех ожиданий, которые на них возлагались, или же в пилотном проекте они использовались удовлетворительно, однако опыт показал, что дальнейшие вложения в средства не гарантируют успеха.

Возможны четыре категории результатов и соответствующих действий:

- Пилотный проект потерпел неудачу, и его анализ показал неадекватность ожиданий организации. В этом случае организация может пересмотреть результаты проекта в контексте более реалистичных ожиданий.
- Пилотный проект потерпел неудачу, и его анализ показал, что выбранные средства не удовлетворяют потребности организации. В этом случае организация может принять решение не внедрять данные средства, однако при этом также пересмотреть свои потребности и подход к оценке и выбору CASE-средств.
- Пилотный проект потерпел неудачу, и его анализ показал наличие таких проблем, как неудачный выбор пилотного проекта, неадекватное обучение и недостаток ресурсов. В этом случае может оказаться достаточно сложно принять решение о том, следует ли вновь выполнить пилотный проект, продолжить работу по внедрению или отказаться от CASE-средств. Однако, независимо от принятого решения, процесс внедрения нуждается в пересмотре и повышенном внимании.
- Пилотный проект завершился успешно, и признано целесообразным внедрять CASE-средства в некоторых подразделениях или, возможно, во всей организации в целом. В этом случае следующим шагом является определение наиболее подходящего масштаба внедрения.

В ряде случаев анализ пилотного проекта может показать, что причиной неудачи явился более чем один фактор. Последующие попытки внедрения CASE-технологии должны четко выявить все причины неудачи. В экстремальных случаях тщательный анализ может показать, что в настоящий момент организация просто не готова к успешному внедрению сложных CASE-средств. В такой ситуации организация может попытаться решить свои проблемы другими средствами.

Особенности пилотного проекта

Очень важно провести анализ пилотного проекта с тем, чтобы определить его элементы, являющиеся критическими для успеха, и определить степень отражения этими элементами деятельности организации в целом. Например, если в пилотном проекте участвуют самые лучшие программисты организации, он может закончиться успешно даже вопреки использованию CASE-средств, а не благодаря им. С другой стороны, CASE-средства могут быть применены для разработки приложения, для которого они явно не подходят по своим характеристикам. Тем не менее, такое использование могло бы указать на область наиболее рационального применения средств в данной организации.

Важнейшие характеристики пилотного проекта, не являющиеся представительными для организации в целом, могут включать следующие:

- Процессы в пилотном проекте в чем-либо отличаются от процессов во всей организации.
- Квалификация группы пилотного проекта не отражает квалификацию остальных специалистов организации.
- Ресурсы, выделенные на выполнение проекта, отличаются от тех, которые выделяются для обычных проектов.
- Предметная область или масштаб проекта не соответствуют другим проектам.

Выгода от использования CASE-средств

Пилотный проект следует сопоставить с деятельностью организации в целом с тем, чтобы определить наиболее существенное сходство и отличие. Например, если наиболее заинтересованные и квалифицированные участники проекта столкнулись с серьезными трудностями в освоении средств, то менее заинтересованным и квалифицированным программистам из других подразделений потребуется существенно большее обучение.

Пилотный проект может также показать, что средства целесообразно использовать для некоторых классов проектов и нецелесообразно для других. Например, средство формальной верификации может подходить для жизненно важных приложений и не подходить для менее критических приложений.

Перед разработкой плана перехода организация должна оценить ожидаемый эффект для различных подразделений или классов проектов. При этом следует учитывать, что некоторые подразделения могут не обладать необходимой квалификацией или ресурсами для использования CASE-средств.

Принятие решения о внедрении

Возможным решением должно быть одно из следующих:

- Внедрить средство. В этом случае рекомендуемый масштаб внедрения должен быть определен в терминах структурных подразделений и предметной области.
- Выполнить дополнительный пилотный проект. Такой вариант должен рассматриваться только в том случае, если остались конкретные неразрешенные вопросы относительно внедрения CASE-средства в организации. Новый пилотный проект должен быть таким, чтобы ответить на эти вопросы.
- Отказаться от средства. В этом случае причины отказа от конкретного средства должны быть определены в терминах потребностей организации или критериев, которые остались неудовлетворенными. Перед тем, как продолжить деятельность по внедрению CASE-средств, потребности организации должны быть пересмотрены на предмет своей обоснованности.
- Отказаться от использования CASE-средств вообще. Пилотный проект может показать, что организация либо не готова к внедрению CASE-средств, либо автоматизация данного аспекта процесса создания и сопровождения ПО не дает никакого эффекта для организации. В этом случае причины отказа от CASE-средств должны быть также определены в терминах потребностей организации или критериев, которые остались неудовлетворенными. При этом необходимо понимать отличие этого варианта от предыдущего, связанного с недостатками конкретного средства.

Результатом данного этапа является документ, в котором обсуждаются результаты пилотного проекта и детализируются решения по внедрению.

4.4. Переход к практическому использованию CASE-средств

Процесс перехода к практическому использованию CASE-средств начинается с разработки и последующей реализации плана перехода. Этот план может отражать поэтапный подход к переходу, начиная с тщательно выбранного пилотного проекта до проектов с существенно возросшим разнообразием характеристик.

Разработка плана перехода

План перехода должен включать следующее:

- Информацию относительно целей, критериев оценки, графика и возможных рисков, связанных с реализацией плана.
- Информацию относительно приобретения, установки и настройки CASE-средств.
- Информацию относительно интеграции каждого средства с существующими средствами, включая как интеграцию CASE-средств друг с другом, так и их интеграцию в процессы разработки и эксплуатации ПО, существующие в организации.
- Ожидаемые потребности в обучении и ресурсы, используемые в течение и после завершения процесса перехода.
- Определение стандартных процедур использования средств.

Цели, критерии оценки, график и риски, связанные с планом перехода

Данная информация должна включать следующее:

- Типы проектов, в которых в конечном счете будет использоваться средство.
- График перехода к практическому использованию средства в отдельных проектах, который включает необходимую подготовку к его широкому использованию.
- График внедрения средства в терминах количества пользователей, включая необходимое обучение.
- Возможные риски и непредвиденные обстоятельства.
- Источники существующих (базовых) данных и метрики для оценки изменений, вызванных использованием средств.

В дополнение к сказанному, следует уделить особое внимание вопросам контроля изменений. Роли высшего руководства, субъектов и объектов изменений должны быть уточнены по сравнению с пилотным проектом, поскольку технология подлежит широкому распространению в организации.

Подразумевается, что план перехода успешно выполнен, когда не требуется больше специального планирования поддержки использования средства. В этот момент использование средства согласуется с тем, что от него ожидалось, и план работы с ним включается в общий план текущей поддержки ПО, существующий в организации.

Приобретение, установка и настройка средств

Приобретение, установка и настройка, выполненные в рамках пилотного проекта, могут потребоваться в более широком масштабе. При этом необходима следующая информация:

- Совокупность программных компонент, документации и обучения, которые необходимо приобретать для каждой отдельной платформы.
- Механизм получения новых версий.
- Настройка средства, необходимая для выполнения существующих в организации процедур и соглашений.
- Лицо или подразделение, ответственное за установку, интеграцию, настройку и эксплуатацию средства.
- План конвертирования данных и снятия старых средств с эксплуатации.

Задачи приобретения, установки и настройки должны быть как можно быстрее переданы из группы пилотного проекта в существующую службу системной поддержки ПО организации.

Интеграция средства с существующими средствами и процессами

Интеграция нового средства с существующими средствами и процессами является важным шагом в полномасштабном внедрении средства. В большинстве случаев такая интеграция в процессе пилотного проектирования не осуществляется, однако накапливаемая в этом процессе информация может помочь в разработке планов интеграции. Для планирования интеграции необходима следующая информация:

- Наименования и версии существующих средств, с которыми должно интегрироваться новое средство.
- Описания данных, которые должны совместно использоваться новым и существующими средствами, а также предварительная информация об источниках этих данных.
- Описания других взаимосвязей между новым и существующими средствами (таких, как связи по передаче управления и порядку использования), а также предварительная информация о механизмах поддержки этих взаимосвязей.
- Оценки затрат, сроков и рисков, связанных с интеграцией (и, возможно, переходом от существующих средств и данных).
- Определение способа внедрения данного средства в деятельность по совершенствованию существующих процессов.
- Ожидаемые изменения в существующих процессах и продуктах, являющиеся следствием использования нового средства и оцениваемые, по возможности, количественно.

Риск, связанный с интеграцией нового средства с существующими средствами и процессами, снижается, если потребности в интеграции учитываются в процессе оценки и выбора средства.

Обучение и ресурсы, используемые в течение и после завершения процесса перехода

Данная информация должна включать следующее:

- Персонал (включая пользователей, администраторов и интеграторов), нуждающийся в обучении использованию средства.
- Вид обучения, необходимого для каждой категории пользователей и обслуживающего персонала, с учетом особой важности обучения совместному использованию различных средств, а также методам и процессам, связанным с данными средствами.
- Вид обучения, необходимого для различных специалистов (например, для группы тестирования и независимой службы сертификации).
- Частота обучения.
- Виды и доступность поддержки.

Определение стандартов и процедур использования средств

План перехода должен определять следующие стандарты и процедуры использования средств:

- Руководства по моделированию и проектированию.
- Соглашения по присвоению имен.
- Процедуры контроля качества и процессов приемки, включая расписание экспертиз и используемые методологии.
- Процедуры резервного копирования, защиты мастер-копий и конфигурирования базы данных.
- Процедуры интеграции с существующими средствами и базами данных.
- Процедуры совместного использования данных и контроля целостности БД.
- Стандарты и процедуры обеспечения секретности.
- Стандарты документирования.

Стандарты использования CASE-средств, выработанные во время пилотного проекта, должны использоваться в качестве отправной точки для разработки более полного набора стандартов использования средств в данной организации (см. подраздел 1.3). При этом должен учитываться опыт участников пилотного проекта.

Реализация плана перехода

Реализация плана перехода требует постоянного мониторинга использования CASE-средств, обеспечения текущей поддержки, сопровождения и обновления средств по мере необходимости.

Периодические экспертизы

Достигнутые результаты должны периодически подвергаться экспертизе в соответствии с графиком, план перехода должен корректироваться при необходимости. Постоянное внимание должно уделяться степени удовлетворения потребностей организации и критериев успешного внедрения CASE-средств.

Периодические экспертизы должны продолжаться и после завершения процесса внедрения. Такие экспертизы могут анализировать метрики и другую информацию, получаемую в процессе работы с CASE-средствами, чтобы определять, насколько хорошо они продолжают выполнять требуемые функции. Такие экспертизы могут также указать на необходимость дополнительной модификации процессов.

Текущая поддержка

Текущая поддержка необходима для следующего:

- Ответов на вопросы, связанные с использованием средств.
- Передачи информации о достигнутых успехах и полученных уроках другим специалистам организации.
- Модификации и совершенствования стандартов, соглашений и процедур, связанных с использованием средства.
- Интеграции новых средств с существующими и сопровождение интегрированных средств по мере появления новых версий.
- Помощи новым сотрудникам в освоении средств и связанных с ними процедур.
- Планирования и контроля обновления версий.
- Планирования внедрения новых возможностей средств в организационные процессы.

Действия, выполняемые в процессе перехода

Для поддержки процесса перехода к практическому использованию средств желательно выполнение следующих действий:

- Поддержка текущего обучения. Потребность в обучении может возникать периодически вследствие появления новых версий средств или вовлечения в проект новых сотрудников.
- Поддержка ролевых функций, связанных с процессом внедрения. Поскольку внедрение CASE-средств приводит к изменениям в культуре организации, необходимо в процессе внедрения выделить ряд ключевых ролей (такие, как руководство высшего уровня, проектная группа и целевые группы).
- Методики управления обновлением версий. Эти методики могут быть связаны с обновлением версий, процедурами установки, процедурами контроля качества для оценки новых версий, процедурами обновления базы данных, конфигурацией версий и средой поддержки (другие средства, операционная система и т.д.).
- Свободный доступ к информации. Должны быть определены механизмы, обеспечивающие свободный доступ к информации об опыте внедрения и извлеченных из этого уроках, включая доски объявлений, информационные бюллетени, пользовательские группы, семинары и публикации.
- Налаживание тесного рабочего взаимодействия с поставщиком. Такое взаимодействие позволяет организации быть в курсе планов поставщика и обеспечивать оперативное удовлетворение своих требований.

Для успешного внедрения CASE-средств в организации существенно важной является последовательность в их применении. Поскольку большинство систем разрабатываются коллективно, необходимо определить характер будущего использования средств как отдельными разработчиками, так и группами. Использование стандартных процедур позволит обеспечить плавный переход между отдельными стадиями ЖЦ ПО.

Как правило, все понимают, что обучение является центральным звеном, обеспечивающим нормальное использование CASE-средств в организации. Тем не менее, довольно распространенная ошибка заключается в том, что производится начальное обучение для группы неподготовленных пользователей, а затем все ограничивается минимальным текущим обучением. Участники пилотного проекта, получившие начальное обучение, могут быть высоко квалифицированными энтузиастами новой технологии, стремящимися использовать ее во что бы то ни стало. С другой стороны, для разработчиков, которые будут участвовать в проекте в дальнейшем, может потребоваться более интенсивное и глубокое обучение, а также текущая поддержка в использовании средства.

В дополнение к этому следует отметить, что каждая категория персонала (например, администраторы средств, служба поддержки рабочих мест, интеграторы средств, служба сопровождения и разработчики приложений) нуждается в различном обучении.

Обучение не должно замыкаться только на пользователях CASE-средств, обучаться должны также те сотрудники организации, на деятельность которых так или иначе оказывает влияние использование CASE-средств.

В процессе дальнейшего использования средств в организации обучение должно стать частью процесса ориентации при найме новых сотрудников и привлечении сотрудников к проектам, в которых используются CASE-средства. Обучение должно стать неотъемлемой составной частью нормативных материалов, касающихся деятельности организации, которые предлагаются новым сотрудникам.

Одна общая ошибка, которая делается в процессе перехода, заключается в недооценке ресурсов, необходимых для поддержки постоянного использования сложных CASE-средств. Рост необходимых ресурсов вызывается тремя причинами:

- Сложностью средств,
- Частотой появления новых версий,
- Взаимодействием между средствами и внешней средой.

Сложность средств приводит к возрастанию потребностей в тщательном и продуманном обучении. Кроме того, многие CASE-средства могут использоваться только квалифицированными специалистами, умеющими сопровождать проектные базы данных и оперативно реагировать на возникающие проблемы. Высокая частота обновления версий средств может привести к

возникновению нетривиальных проблем, которые зачастую упускаются из виду. Такие обновления обычно пагубно отражаются на жестких планах и графиках работы. Взаимодействие между средствами и внешней по отношению к ним средой также может иногда порождать некоторые проблемы. Имеется в виду тот факт, что хотя многие средства достигли уровня минимальной несовместимости данных между отдельными версиями, проблемы обеспечения совместимости с другими элементами внешней среды остаются в силе.

Оценка результатов перехода

Программа постоянной оценки качества и продуктивности ПО имеет важное значение для следующего:

- Определения степени совершенствования процессов,
- Упреждения возможных стратегических просчетов,
- Своевременного отказа от использования устаревшей технологии.

Чтобы определить, насколько эффективно новое CASE-средство повышает продуктивность и/или качество, организация должна опираться на некоторые базовые данные. К сожалению, лишь немногие организации в настоящее время накапливают данные для реализации программы текущей количественной оценки и усовершенствования процессов. Для доказательства эффективности CASE-средств и их возможностей улучшать продуктивность необходимы такие базовые метрические данные, как:

- Использованное время,
- Время, выделенное персонально для конкретных специалистов,
- Размер, сложность и качество ПО,
- Удобство сопровождения.

Метрическая оценка должна начинаться с реальной оценки текущего состояния среды еще до начала внедрения CASE-средств и поддерживать процедуры постоянного накопления данных.

Период времени, в течение которого выполняется количественная оценка воздействия, оказываемого внедрением CASE-средств, является весьма значимой величиной с точки зрения определения степени успешности перехода. Некоторые организации, успешно внедрившие в конечном счете CASE-средства, столкнулись с кратковременными негативными эффектами в начале процесса. Другие, успешно начав, недооценили долговременные затраты на сопровождение и обучение. Вследствие этого, наиболее приемлемый временной интервал для оценки степени успешности внедрения должен быть достаточно большим, чтобы преодолеть любые негативные эффекты на начальном этапе, а также смоделировать будущие долговременные затраты. С другой стороны, данный интервал должен соответствовать целям организации и ожидаемым результатам.

В конечном счете, опыт, полученный при внедрении CASE-средств, может отчасти изменить цели организации и ожидания, возлагаемые на CASE-средства. Например, организация может сделать вывод, что средства целесообразно использовать для большего или меньшего круга пользователей и процессов в цикле создания и сопровождения ПО. Такие изменения в ожиданиях зачастую могут дать положительные результаты, но могут также привести к внесению соответствующих корректив в определение степени успешного внедрения CASE-средств в данной организации.

Результатом данного этапа является внедрение CASE-средств в повседневную практику организации, при этом больше не требуется какого-либо специального планирования. Кроме того, поддержка CASE-средств включается в план текущей поддержки ПО в данной организации.

5. Характеристики CASE-средств

5.1. Silverrun+JAM

5.1.1. Silverrun

CASE-средство Silverrun американской фирмы Computer Systems Advisers, Inc. (CSA) используется для анализа и проектирования ИС бизнес-класса [22] 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.

Структура и функции

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 ПО.

Взаимодействие с другими средствами

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 имеется 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 репозиторий. 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

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.

Среда функционирования

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

Для функционирования в среде Windows необходимо иметь компьютер с процессором модели не ниже i486 и оперативную память объемом не менее 8 Мб (рекомендуется 16 Мб). На диске полная инсталляция Silverrun занимает 20 Мб.

5.1.2. JAM

Средство разработки приложений JAM [28] (JYACC's Application Manager) - продукт фирмы JYACC (США). В настоящее время поставляется версия JAM 7 и готовится к выходу JAM 8.

Основной чертой JAM является его соответствие методологии RAD, поскольку он позволяет достаточно быстро реализовать цикл разработки приложения, заключающийся в формировании очередной версии прототипа приложения с учетом требований, выявленных на предыдущем шаге, и предъявить его пользователю.

Структура и функции

JAM имеет модульную структуру и состоит из следующих компонент:

- Ядро системы;
- JAM/DBi - специализированные модули интерфейса к СУБД (JAM/DBi-Oracle, JAM/DBi-Informix, JAM/DBi-ODBC и т.д.);
- JAM/RW - модуль генератора отчетов;
- JAM/CASEi - специализированные модули интерфейса к CASE-средствам (JAM/CASE-TeamWork, JAM/CASE-Innovator и т.д.);
- JAM/TPi - специализированные модули интерфейса к менеджерам транзакций (например, JAM/TPi-Server TUXEDO и т.д.);
- Jterm - специализированный эмулятор X-терминала.

Ядро системы (собственно, сам JAM) является законченным продуктом и может самостоятельно использоваться для разработки приложений. Все остальные модули являются дополнительными и самостоятельно использоваться не могут.

Ядро системы включает в себя следующие основные компоненты:

- редактор экранов. В состав редактора экранов входят: среда разработки экранов, визуальный репозиторий объектов, собственная СУБД JAM - JDB, менеджер транзакций, отладчик, редактор стилей;
- редактор меню;
- набор вспомогательных утилит;
- средства изготовления промышленной версии приложения.

При использовании JAM разработка внешнего интерфейса приложения представляет собой визуальное проектирование и сводится к созданию экранных форм путем размещения на них интерфейсных конструкций и определению экранных полей ввода/вывода информации. Проектирование интерфейса в JAM осуществляется с помощью редактора экранов. Приложения, разработанные в JAM, имеют многооконный интерфейс. Разработка отдельного экрана заключается в размещении на нем интерфейсных элементов, возможной (но не обязательной) их группировке и конкретизации различных их свойств, включающих визуальные характеристики (позиция, размер, цвет, шрифт и т.п.), поведенческие характеристики (многообразные фильтры, форматы, защита от ввода и т.п.) и ряд свойств, ориентированных на работу с БД.

Редактор меню позволяет разрабатывать и отлаживать системы меню. Реализована возможность построения пиктографических меню (так называемые toolbar). Назначение каждого конкретного меню тому или иному объекту приложения осуществляется в редакторе экранов.

В ядро JAM встроена однопользовательская реляционная СУБД JDB. Основным назначением JDB является прототипирование приложений в тех случаях, когда работа со штатной СУБД невозможна или нецелесообразна. В JDB реализован необходимый минимум возможностей реляционных СУБД за исключением индексов, хранимых процедур, триггеров и представлений (view). С помощью JDB можно построить БД, идентичную целевой БД (с точностью до отсутствующих в JDB возможностей) и разработать значительную часть приложения.

Отладчик позволяет проводить комплексную отладку разрабатываемого приложения. Осуществляется трассировка всех событий, возникающих в процессе исполнения приложения.

Утилиты JAM включают три группы:

- конверторы файлов экранов JAM в текстовые. JAM сохраняет экраны в виде двоичных файлов собственного формата. В ряде случаев (например для изготовления программной документации проекта) необходимо текстовое описание экранов;

- конфигурирование устройств ввода/вывода. JAM и приложения, построенные с его помощью, не работают непосредственно с устройствами ввода/вывода. Вместо этого JAM обращается к логическим устройствам ввода/вывода (клавиатура, терминал, отчет). Отображение логических устройств в физические осуществляется с помощью средств конфигурирования;
- обслуживание библиотек экранов (традиционные операции с библиотеками).

Одним из дополнительных модулей JAM является генератор отчетов. Компоновка отчета осуществляется в редакторе экранов JAM. Описание работы отчета осуществляется с помощью специального языка. Генератор отчетов позволяет определить данные, выводимые в отчет, группировку выводимой информации, форматирование вывода и др.

Приложения, разработанные с использованием JAM, не требуют так называемых исполнительных (run-time) систем и могут быть изготовлены в виде исполняемых модулей. Для этого разработчик должен иметь компилятор С и редактор связей. Для изготовления промышленной версии в состав JAM входит файл сборки (makefile), исходные тексты (на языке С) ряда модулей приложения и необходимые библиотеки.

JAM содержит встроенный язык программирования JPL (JAM Procedural Language), с помощью которого в случае необходимости можно написать модули, реализующие специфические действия. Данный язык является интерпретируемым, что упрощает отладку. Существует возможность обмена информацией между средой визуально построенного приложения и такими модулями. Кроме того, в JAM реализована возможность подключения внешних модулей, написанных на каком-либо языке, совместимым по вызовам функций с языком С.

С точки зрения реализации логики приложения JAM является событийно-ориентированной системой. В JAM определен набор событий, включающий открытие и закрытие окон, нажатие клавиши клавиатуры, срабатывание системного таймера, получение и передача управления каждым элементом экрана. Разработчик реализует логику приложения путем определения обработчика каждого события. Например, обработчик события "нажатие кнопки на экране" (мышью или с помощью клавиатуры) может открыть следующее экранное окно. Обработчиками событий в JAM могут быть как встроенные функции JAM, так и функции, написанные разработчиком на С или JPL. Набор встроенных функций включает в себя более 200 функций различного назначения. Встроенные функции доступны для вызовов из функций, написанных как на JPL, так и на С.

Промышленная версия приложения, разработанного с помощью JAM, включает в себя следующие компоненты:

- исполняемый модуль интерпретатора приложения. В этот модуль могут быть встроены функции, написанные разработчиками на языках 3-го поколения;
- экраны, составляющие само приложение (могут поставляться в виде отдельных файлов, в составе библиотек экранов или же быть встроены в тело интерпретатора);
- внешние JPL-модули. Могут поставляться в виде текстовых файлов или в прекомпилированном виде, причем прекомпилированные внешние JPL-модули могут быть как в виде отдельных файлов, так и в составе библиотек экранов;
- файлы конфигурации приложения - файлы конфигурации клавиатуры и терминала, файл системных сообщений, файл общей конфигурации.

Взаимодействие с другими средствами

Непосредственное взаимодействие с СУБД реализуют модули JAM/DBi (Data Base interface). Способы реализации взаимодействия в JAM разделяются на два класса: ручные и автоматические. При ручном способе разработчик приложения самостоятельно пишет запросы на SQL, в которых как источниками, так и адресатами приема результатов выполнения запроса могут быть как интерфейсные элементы визуально спроектированного внешнего уровня, так и внутренние, невидимые для конечного пользователя переменные. Автоматический режим, реализуемый менеджером транзакций JAM, осуществим для типовых и наиболее распространенных видов операций с БД, так называемых QBE (Query By Example - запросы по образцу), с учетом достаточно сложных взаимосвязей между таблицами БД и автоматическим

управлением атрибутами экранных полей ввода/вывода в зависимости от вида транзакции (чтение, запись и т.д.), в которой участвует сгенерированный запрос.

JAM позволяет строить приложения для работы более чем с 20 СУБД: ORACLE, Informix, Sybase, Ingres, InterBase, NetWare SQL Server, Rdb, DB2, ODBC-совместимые СУБД и др.

Отличительной чертой JAM является высокий уровень переносимости приложений между различными платформами (MS DOS/MS Windows, SunOS, Solaris (i80x86, SPARC), HP-UX, AIX, VMS/Open VMS и др.). Может потребоваться лишь "перерисовать" статические текстовые поля на экранах с русским текстом при переносе между средами DOS-Windows-UNIX. Кроме того, переносимость облегчается тем, что в JAM приложения разрабатываются для виртуальных устройств ввода/вывода, а не для физических. Таким образом при переносе приложения с платформы на платформу, как правило, требуется лишь определить соответствие между физическими устройствами ввода/вывода и их логическими представлениями для приложения.

Использование SQL в качестве средства взаимодействия с СУБД также создает предпосылки для обеспечения переносимости между СУБД. При условии переноса структуры самой БД в ряде случаев приложения могут не требовать никакой модификации, за исключением инициализации сеанса работы. Такая ситуация может сложиться в том случае, если в приложении не использовались специфические для той или иной СУБД расширения SQL.

При росте нагрузки на систему и сложности решаемых задач (распределенность и гетерогенность используемых ресурсов, количество одновременно подключенных пользователей, сложность логики приложения) применяется трехзвенная модель архитектуры "клиент-сервер" с использованием менеджеров транзакций. Компоненты JAM/TPi-Client и JAM/TPi-Server позволяют достаточно просто перейти на трехзвенную модель. При этом ключевую роль играет модуль JAM/TPi-Server, так как основная трудность внедрения трехзвенной модели заключается в реализации логики приложения в сервисах менеджеров транзакций.

Интерфейс JAM/CASE подобен интерфейсу к СУБД и позволяет осуществить обмен информацией между репозиторием объектов JAM и репозиторием CASE-средства аналогично тому, как структура БД импортируется в репозиторий JAM непосредственно из БД. Отличие заключается в том, что в случае интерфейса к CASE этот обмен является двунаправленным. Кроме модулей JAM/CASEi, существует также модуль JAM/CASEi Developer's Kit. С помощью этого модуля можно самостоятельно разработать интерфейс (т.е. специализированный модуль JAM/CASEi) для конкретного CASE-средства, если готового модуля JAM/CASEi для него не существует.

Мост (интерфейс) Silverrun-RDM <-> JAM реализует взаимодействие между CASE-средством Silverrun и JAM (перенос схемы базы данных и экранных форм приложения между CASE-средством Silverrun-RDM и JAM версии 7.0). Данный программный продукт имеет 2 режима работы:

- прямой режим (Silverrun-RDM->JAM) предназначен для создания объектов CASE-словаря и элементов репозитория JAM на основе представления схем в Silverrun-RDM. В этом режиме мост позволяет, исходя из представления моделей данных интерфейса в Silverrun-RDM, производить генерацию экранов и элементов репозитория JAM. Мост преобразует таблицы и отношения реляционных схем RDM в последовательность объектов JAM соответствующих типов. Методика построения моделей данных интерфейса в Silverrun-RDM предполагает применение механизма подсхем для прототипирования экранов приложения. По описанию каждой из подсхем RDM мост генерирует экранную форму JAM;
- обратный режим (JAM->Silverrun-RDM) предназначен для переноса модификаций объектов CASE-словаря в реляционную модель Silverrun-RDM.

Режим реинжиниринга позволяет переносить модификации всех свойств экранов JAM, импортированных ранее из RDM, в схему Silverrun. На этом этапе для контроля целостности базы данных не допускаются изменения схемы в виде добавления или удаления таблиц и полей таблиц.

Групповая работа

Ядро JAM имеет встроенный интерфейс к средствам конфигурационного управления (PVCS на платформе Windows и SCCS на платформе UNIX). Под управлением этих систем

передаются библиотеки экранов и/или репозитории. При отсутствии таких систем JAM самостоятельно реализует часть функций поддержки групповой разработки.

Использование PVCS (см. подраздел 5.6) является более предпочтительным по сравнению с SCCS, так как позволяет организовать единый архив модулей проекта для всех платформ. Так как JAM на платформе UNIX не имеет прямого интерфейса к архивам PVCS, то выборка модулей из архива и возврат их в архив производятся с использованием PVCS Version Manager. На платформе MS-Windows JAM имеет встроенный интерфейс к PVCS и действия по выборке/возврату производятся непосредственно из среды JAM.

Среда функционирования

JAM, как среда разработки, и приложения, построенные с его использованием, не являются ресурсоемкими системами. Например, на платформе MS-Windows достаточно иметь 8MB оперативной памяти и 50 MB дискового пространства для среды разработки. На UNIX-платформах требования к аппаратуре определяются самой операционной системой.

5.2. Vantage Team Builder (Westmount I-CASE) + Uniface

5.2.1. Vantage Team Builder (Westmount I-CASE)

Vantage Team Builder [14] представляет собой интегрированный программный продукт, ориентированный на реализацию каскадной модели ЖЦ ПО и поддержку полного ЖЦ ПО.

Структура и функции

Vantage Team Builder обеспечивает выполнение следующих функций:

- проектирование диаграмм потоков данных, "сущность-связь", структур данных, структурных схем программ и последовательностей экранных форм;
- проектирование диаграмм архитектуры системы - SAD (проектирование состава и связи вычислительных средств, распределения задач системы между вычислительными средствами, моделирование отношений типа "клиент-сервер", анализ использования менеджеров транзакций и особенностей функционирования систем в реальном времени);
- генерация кода программ на языке 4GL целевой СУБД с полным обеспечением программной среды и генерация SQL-кода для создания таблиц БД, индексов, ограничений целостности и хранимых процедур;
- программирование на языке C со встроенным SQL;
- управление версиями и конфигурацией проекта;
- многопользовательский доступ к репозиторию проекта;
- генерация проектной документации по стандартным и индивидуальным шаблонам;
- экспорт и импорт данных проекта в формате CDIF (CASE Data Interchange Format).

Vantage Team Builder поставляется в различных конфигурациях в зависимости от используемых СУБД (ORACLE, Informix, Sybase или Ingres) или средств разработки приложений (Uniface). Конфигурация Vantage Team Builder for Uniface отличается от остальных некоторой степенью ориентации на спиральную модель ЖЦ ПО за счет возможностей быстрого прототипирования, предоставляемых Uniface. Для описания проекта ИС используется достаточно большой набор диаграмм, конкретные варианты которого для наиболее распространенных конфигураций приведены ниже в таблице.

При построении всех типов диаграмм обеспечивается контроль соответствия моделей синтаксису используемых методов, а также контроль соответствия одноименных элементов и их типов для различных типов диаграмм.

При построении DFD обеспечивается контроль соответствия диаграмм различных уровней декомпозиции. Контроль за правильностью верхнего уровня DFD осуществляется с помощью матрицы списков событий (ELM). Для контроля за декомпозицией составных потоков данных используется несколько вариантов их описания: в виде диаграмм структур данных (DSD) или в нотации БНФ (форма Бэкуса-Наура).

Для построения SAD используется расширенная нотация DFD, дающая возможность вводить понятия процессоров, задач и периферийных устройств, что обеспечивает наглядность проектных решений.

Тип диаграммы	Обозначение	Vantage Team Builder for ORACLE	Vantage Team Builder for Informix	Vantage Team Builder for Uniface
Сущность-связь	ERD	+	+	+
Потоков данных	DFD	+	+	+
Структур данных	DSD	+	+	+
Архитектуры системы	SAD	+	+	+
Потоков управления	CSD	+	+	+
Типов данных	DTD	+	+	+
Структуры меню	MSD	+		
Последовательности блоков	BSD	+		
Последовательности форм	FSD		+	+
Содержимого форм	FCD		+	+
Переходов состояний	STD	+	+	+
Структурных схем	SCD	+	+	+

При построении модели данных в виде ERD выполняется ее нормализация и вводится определение физических имен элементов данных и таблиц, которые будут использоваться в процессе генерации физической схемы данных конкретной СУБД. Обеспечивается возможность определения альтернативных ключей сущностей и полей, составляющих дополнительные точки входа в таблицу (поля индексов), и мощности отношений между сущностями.

Наличие универсальной системы генерации кода, основанной на специфицированных средствах доступа к репозиторию проекта, позволяет поддерживать высокий уровень исполнения проектной дисциплины разработчиками: жесткий порядок формирования моделей; жесткая структура и содержимое документации; автоматическая генерация исходных кодов программ и т.д. - все это обеспечивает повышение качества и надежности разрабатываемых ИС.

Для подготовки проектной документации могут использоваться издательские системы FrameMaker, Interleaf или Word Perfect. Структура и состав проектной документации могут быть настроены в соответствии с заданными стандартами. Настройка выполняется без изменения проектных решений.

При разработке достаточно крупной ИС вся система в целом соответствует одному проекту как категории Vantage Team Builder. Проект может быть декомпозирован на ряд систем, каждая из которых соответствует некоторой относительно автономной подсистеме ИС и разрабатывается независимо от других. В дальнейшем системы проекта могут быть интегрированы.

Процесс проектирования ИС с использованием Vantage Team Builder реализуется в виде 4-х последовательных фаз (стадий) - анализа, архитектуры, проектирования и реализации, при этом законченные результаты каждой стадии полностью или частично переносятся (импортируются) в следующую фазу. Все диаграммы, кроме ERD, преобразуются в другой тип или изменяют вид в соответствии с особенностями текущей фазы. Так, DFD преобразуются в фазе архитектуры в SAD, DSD - в DTD. После завершения импорта логическая связь с предыдущей фазой разрывается, т.е. в диаграммы могут вноситься все необходимые изменения.

Взаимодействие с другими средствами

Конфигурация Vantage Team Builder for Uniface обеспечивает совместное использование двух систем в рамках единой технологической среды проектирования, при этом схемы БД (SQL-модели) переносятся в репозиторий Uniface, и, наоборот, прикладные модели, сформированные средствами Uniface, могут быть перенесены в репозиторий Vantage Team Builder. Возможные рассогласования между репозиториями двух систем устраняются с помощью специальной

утилиты. Разработка экранных форм в среде Uniface выполняется на базе диаграмм последовательностей форм (FSD) после импорта SQL-модели. Технология разработки ИС на базе данной конфигурации показана на рисунке 5.1.

Структура репозитория (хранящегося непосредственно в целевой СУБД) и интерфейсы Vantage Team Builder являются открытыми, что в принципе позволяет интеграцию с любыми другими средствами.

Среда функционирования

Vantage Team Builder функционирует на всех основных UNIX-платформах (Solaris, SCO UNIX, AIX, HP-UX) и VMS.

Vantage Team Builder можно использовать в конфигурации "клиент-сервер", при этом база проектных данных может располагаться на сервере, а рабочие места разработчиков могут быть клиентами.

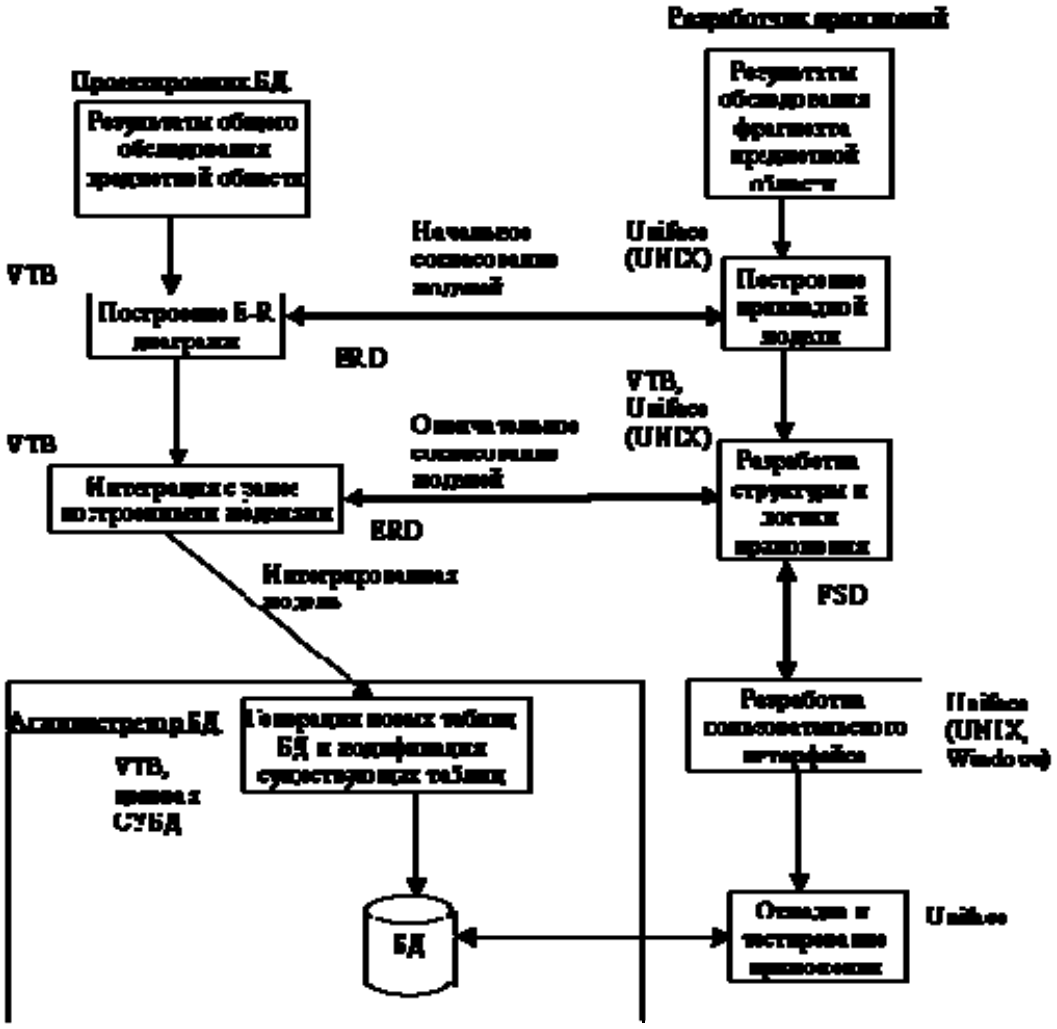


Рис. 5.1. Взаимодействие Vantage Team Builder и Uniface

5.2.2. Uniface

Uniface 6.1 [15] - продукт фирмы Compuware (США) - представляет собой среду разработки крупномасштабных приложений в архитектуре "клиент-сервер" и имеет следующую компонентную архитектуру:

- Application Objects Repository (репозиторий объектов приложений) содержит метаданные, автоматически используемые всеми остальными компонентами на протяжении жизненного цикла ИС (прикладные модели, описания данных, бизнес-правил, экранных форм, глобальных объектов и шаблонов). Репозиторий может храниться в любой из баз данных, поддерживаемых Uniface;
- Application Model Manager поддерживает прикладные модели (E-R модели), каждая из которых представляет собой подмножество общей схемы БД с точки зрения данного приложения, и включает соответствующий графический редактор;

- Rapid Application Builder - средство быстрого создания экранных форм и отчетов на базе объектов прикладной модели. Оно включает графический редактор форм, средства прототипирования, отладки, тестирования и документирования. Реализован интерфейс с разнообразными типами оконных элементов управления (Open Widget Interface) для существующих графических интерфейсов - MS Windows (включая VBX), Motif, OS/2. Универсальный интерфейс представления (Universal Presentation Interface) позволяет использовать одну и ту же версию приложения в среде различных графических интерфейсов без изменения программного кода;
- Developer Services (службы разработчика) - используются для поддержки крупных проектов и реализуют контроль версий (Uniface Version Control System), права доступа (разграничение полномочий), глобальные модификации и т.д. Это обеспечивает разработчиков средствами параллельного проектирования, входного и выходного контроля, поиска, просмотра, поддержки и выдачи отчетов по данным системы контроля версий;
- Deployment Manager (управление распространением приложений) - средства, позволяющие подготовить созданное приложение для распространения, устанавливать и сопровождать его (при этом платформа пользователя может отличаться от платформы разработчика). В их состав входят сетевые драйверы и драйверы СУБД, сервер приложений (полисервер), средства распространения приложений и управления базами данных. Uniface поддерживает интерфейс практически со всеми известными программно-аппаратными платформами, СУБД, CASE-средствами, сетевыми протоколами и менеджерами транзакций;
- Personal Series (персональные средства) - используются для создания сложных запросов и отчетов в графической форме (Personal Query и Personal Access - PQ/PA), а также для переноса данных в такие системы, как WinWord и Excel;
- Distributed Computing Manager - средство интеграции с менеджерами транзакций Tuxedo, Encina, CICS, OSF DCE.

Объявленная в конце 1996 г. версия Uniface 7 полностью поддерживает распределенную модель вычислений и трехзвенную архитектуру "клиент-сервер" (с возможностью изменения схемы декомпозиции приложений на этапе исполнения). Приложения, создаваемые с помощью Uniface 7, могут исполняться в гетерогенных операционных средах, использующих различные сетевые протоколы, одновременно на нескольких разнородных платформах (в том числе и в Internet).

В состав компонент Uniface 7 входят:

- Uniface Application Server - сервер приложений для распределенных систем;
- WebEnabler - серверное ПО для эксплуатации приложений в Internet и Intranet;
- Name Server - серверное ПО, обеспечивающее использование распределенных прикладных ресурсов;
- PolyServer - средство доступа к данным и интеграции различных систем.

В список поддерживаемых СУБД входят DB2, VSAM и IMS; PolyServer обеспечивает также взаимодействие с ОС MVS.

Среда функционирования Uniface - все основные UNIX - платформы и MS Windows.

5.3. Designer/2000 + Developer/2000

CASE-средство Designer/2000 2.0 фирмы ORACLE [23] является интегрированным CASE-средством, обеспечивающим в совокупности со средствами разработки приложений Developer/2000 поддержку полного ЖЦ ПО для систем, использующих СУБД ORACLE.

Структура и функции

Designer/2000 представляет собой семейство методологий и поддерживающих их программных продуктов. Базовая методология Designer/2000 (CASE*Method) - структурная методология проектирования систем, полностью охватывающая все этапы жизненного цикла ИС [8,9]. В соответствии с этой методологией на этапе планирования определяются цели создания системы, приоритеты и ограничения, разрабатывается системная архитектура и план разработки ИС. В процессе анализа строятся модель информационных потребностей (диаграмма "сущность-

связь"), диаграмма функциональной иерархии (на основе функциональной декомпозиции ИС), матрица перекрестных ссылок и диаграмма потоков данных.

На этапе проектирования разрабатывается подробная архитектура ИС, проектируется схема реляционной БД и программные модули, устанавливаются перекрестные ссылки между компонентами ИС для анализа их взаимного влияния и контроля за изменениями.

На этапе реализации создается БД, строятся прикладные системы, производится их тестирование, проверка качества и соответствия требованиям пользователей. Создается системная документация, материалы для обучения и руководства пользователей. На этапах эксплуатации и сопровождения анализируются производительность и целостность системы, выполняется поддержка и, при необходимости, модификация ИС;

Designer/2000 обеспечивает графический интерфейс при разработке различных моделей (диаграмм) предметной области. В процессе построения моделей информация о них заносится в репозиторий. В состав Designer/2000 входят следующие компоненты:

- Repository Administrator - средства управления репозиторием (создание и удаление приложений, управление доступом к данным со стороны различных пользователей, экспорт и импорт данных);
- Repository Object Navigator - средства доступа к репозиторию, обеспечивающие многооконный объектно-ориентированный интерфейс доступа ко всем элементам репозитория;
- Process Modeller - средство анализа и моделирования деловой деятельности, основывающееся на концепциях реинжиниринга бизнес-процессов (BPR - Business Process Reengineering) и глобальной системы управления качеством (TQM - Total Quality Management);
- Systems Modeller - набор средств построения функциональных и информационных моделей проектируемой ИС, включающий средства для построения диаграмм "сущность-связь" (Entity-Relationship Diagrammer), диаграмм функциональных иерархий (Function Hierarchy Diagrammer), диаграмм потоков данных (Data Flow Diagrammer) и средство анализа и модификации связей объектов репозитория различных типов (Matrix Diagrammer);
- Systems Designer - набор средств проектирования ИС, включающий средство построения структуры реляционной базы данных (Data Diagrammer), а также средства построения диаграмм, отображающих взаимодействие с данными, иерархию, структуру и логику приложений, реализуемую хранимыми процедурами на языке PL/SQL (Module Data Diagrammer, Module Structure Diagrammer и Module Logic Navigator);
- Server Generator - генератор описаний объектов БД ORACLE (таблиц, индексов, ключей, последовательностей и т.д.). Помимо продуктов ORACLE, генерация и реинжиниринг БД может выполняться для СУБД Informix, DB/2, Microsoft SQL Server, Sybase, а также для стандарта ANSI SQL DDL и баз данных, доступ к которым реализуется посредством ODBC;
- Forms Generator (генератор приложений для ORACLE Forms). Генерируемые приложения включают в себя различные экранные формы, средства контроля данных, проверки ограничений целостности и автоматические подсказки. Дальнейшая работа с приложением выполняется в среде Developer/2000;
- Repository Reports - генератор стандартных отчетов, интегрированный с ORACLE Reports и позволяющий русифицировать отчеты, а также изменять структурное представление информации.

Репозиторий Designer/2000 представляет собой хранилище всех проектных данных и может работать в многопользовательском режиме, обеспечивая параллельное обновление информации несколькими разработчиками. В процессе проектирования автоматически поддерживаются перекрестные ссылки между объектами словаря и могут генерироваться более 70 стандартных отчетов о моделируемой предметной области. Физическая среда хранения репозитория - база данных ORACLE.

Генерация приложений, помимо продуктов ORACLE, выполняется также для Visual Basic.

Взаимодействие с другими средствами

Designer/2000 можно интегрировать с другими средствами, используя открытый интерфейс приложений API (Application Programming Interface). Кроме того, можно использовать средство ORACLE CASE Exchange для экспорта/импорта объектов репозитория с целью обмена информацией с другими CASE-средствами.

Developer/2000 обеспечивает разработку переносимых приложений, работающих в графической среде Windows, Macintosh или Motif. В среде Windows интеграция приложений Developer/2000 с другими средствами реализуется через механизм OLE и управляющие элементы VBX. Взаимодействие приложений с другими СУБД (DB/2, DB2/400, Rdb) реализуется с помощью средств ORACLE Client Adapter для ODBC, ORACLE Open Gateway и API.

Среда функционирования

Среда функционирования Designer/2000 и Developer/2000 - Windows 3.x, Windows 95, Windows NT.

5.4. Локальные средства (ERwin, BPwin, S-Designor, CASE.Аналитик)

ERwin - средство концептуального моделирования БД [24], использующее методологию IDEF1X (см. подраздел 2.5). ERwin реализует проектирование схемы БД, генерацию ее описания на языке целевой СУБД (ORACLE, Informix, Ingres, Sybase, DB/2, Microsoft SQL Server, Progress и др.) и реинжиниринг существующей БД. ERwin выпускается в нескольких различных конфигурациях, ориентированных на наиболее распространенные средства разработки приложений 4GL. Версия ERwin/OPEN полностью совместима со средствами разработки приложений PowerBuilder и SQLWindows и позволяет экспортировать описание спроектированной БД непосредственно в репозитории данных средств.

Для ряда средств разработки приложений (PowerBuilder, SQLWindows, Delphi, Visual Basic) выполняется генерация форм и прототипов приложений.

Сетевая версия Erwin ModelMart обеспечивает согласованное проектирование БД и приложений в рамках рабочей группы.

BPwin - средство функционального моделирования, реализующее методологию IDEF0 (см. подраздел 2.2).

Возможные конфигурации и ориентировочная стоимость средств (без технической поддержки) приведены в таблице.

Конфигурация	Стоимость, $
ERwin/ERX	3,295
Bpwin	2,495
ERwin/ERX for PowerBuilder, Visual Basic, Progress	3,495
ERwin/ERX for Delphi	4,295
ERwin/Desktop for PowerBuilder, Visual Basic	495
ERwin/ERX for SQLWindows / Designer/2000 / Solaris	3,495 / 5,795 / 6,995
ModelMart 5 / 10 user	11,995 / 19,995
Erwin/OPEN for ModelMart	3,995

S-Designor 4.2 представляет собой CASE-средство для проектирования реляционных баз данных [25]. По своим функциональным возможностям и стоимости он близок к CASE-средству ERwin, отличаясь внешне используемой на диаграммах нотацией. S-Designor реализует стандартную методологию моделирования данных и генерирует описание БД для таких СУБД, как ORACLE, Informix, Ingres, Sybase, DB/2, Microsoft SQL Server и др. Для существующих систем выполняется реинжиниринг БД.

S-Designor совместим с рядом средств разработки приложений (PowerBuilder, Uniface, TeamWindows и др.) и позволяет экспортировать описание БД в репозитории данных средств. Для PowerBuilder выполняется также прямая генерация шаблонов приложений.

CASE.Аналитик 1.1 [3] является практически единственным в настоящее время конкурентоспособным отечественным CASE-средством функционального моделирования и

реализует построение диаграмм потоков данных в соответствии с методологией, описанной в подразделе 2.3. Его основные функции:

- построение и редактирование DFD;
- анализ диаграмм и проектных спецификаций на полноту и непротиворечивость;
- получение разнообразных отчетов по проекту;
- генерация макетов документов в соответствии с требованиями ГОСТ 19.XXX и 34.XXX.

Среда функционирования: процессор - 386 и выше, основная память - 4 Мб, дисковая память - 5 Мб, MS Windows 3.x или Windows 95.

Ориентировочная стоимость:

- однопользовательская версия - 605 $;
- многопользовательская версия (одно рабочее место) - 535 $.

База данных проекта реализована в формате СУБД Paradox и является открытой для доступа.

С помощью отдельного программного продукта (Catherine) выполняется обмен данными с CASE-средством ERwin. При этом из проекта, выполненного в CASE.Аналитике, экспортируется описание структур данных и накопителей данных, которое по определенным правилам формирует описание сущностей и их атрибутов.

5.5. Объектно-ориентированные CASE-средства (Rational Rose)

Rational Rose - CASE-средство фирмы Rational Software Corporation (США) - предназначено для автоматизации этапов анализа и проектирования ПО, а также для генерации кодов на различных языках и выпуска проектной документации [21]. 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 (UML - Unified Modeling Language) претендует на роль стандарта 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.

Структура и функции

В основе работы Rational Rose лежит построение различного рода диаграмм и спецификаций, определяющих логическую и физическую структуры модели, ее статические и динамические аспекты. В их число входят диаграммы классов, состояний, сценариев, модулей, процессов [21].

В составе Rational Rose можно выделить 6 основных структурных компонент: репозиторий, графический интерфейс пользователя, средства просмотра проекта (browser), средства контроля проекта, средства сбора статистики и генератор документов. К ним добавляются генератор кодов (индивидуальный для каждого языка) и анализатор для C++, обеспечивающий реинжиниринг - восстановление модели проекта по исходным текстам программ.

Репозиторий представляет собой объектно-ориентированную базу данных. Средства просмотра обеспечивают "навигацию" по проекту, в том числе, перемещение по иерархиям классов и подсистем, переключение от одного вида диаграмм к другому и т. д. Средства контроля и сбора статистики дают возможность находить и устранять ошибки по мере развития проекта, а не после завершения его описания. Генератор отчетов формирует тексты выходных документов на основе содержащейся в репозитории информации.

Средства автоматической генерации кодов программ на языке C++, используя информацию, содержащуюся в логической и физической моделях проекта, формируют файлы заголовков и файлы описаний классов и объектов. Создаваемый таким образом скелет программы может быть уточнен путем прямого программирования на языке C++. Анализатор кодов C++

реализован в виде отдельного программного модуля. Его назначение состоит в том, чтобы создавать модули проектов в форме Rational Rose на основе информации, содержащейся в определяемых пользователем исходных текстах на C++. В процессе работы анализатор осуществляет контроль правильности исходных текстов и диагностику ошибок. Модель, полученная в результате его работы, может целиком или фрагментарно использоваться в различных проектах. Анализатор обладает широкими возможностями настройки по входу и выходу. Например, можно определить типы исходных файлов, базовый компилятор, задать, какая информация должна быть включена в формируемую модель и какие элементы выходной модели следует выводить на экран. Таким образом, Rational Rose/C++ обеспечивает возможность повторного использования программных компонент.

В результате разработки проекта с помощью CASE-средства Rational Rose формируются следующие документы:

- диаграммы классов;
- диаграммы состояний;
- диаграммы сценариев;
- диаграммы модулей;
- диаграммы процессов;
- спецификации классов, объектов, атрибутов и операций
- заготовки текстов программ;
- модель разрабатываемой программной системы.

Последний из перечисленных документов является текстовым файлом, содержащим всю необходимую информацию о проекте (в том числе необходимую для получения всех диаграмм и спецификаций).

Тексты программ являются заготовками для последующей работы программистов. Они формируются в рабочем каталоге в виде файлов типов .h (заголовки, содержащие описания классов) и .cpp (заготовки программ для методов). Система включает в программные файлы собственные комментарии, которые начинаются с последовательности символов //##. Состав информации, включаемой в программные файлы, определяется либо по умолчанию, либо по усмотрению пользователя. В дальнейшем эти исходные тексты развиваются программистами в полноценные программы.

Взаимодействие с другими средствами и организация групповой работы

Rational Rose интегрируется со средством PVCS для организации групповой работы и управления проектом и со средством SoDA - для документирования проектов. Интеграция Rational Rose и SoDA обеспечивается средствами SoDA.

Для организации групповой работы в Rational Rose возможно разбиение модели на управляемые подмодели. Каждая из них независимо сохраняется на диске или загружается в модель. В качестве подмодели может выступать категория классов или подсистема.

Для управляемой подмодели предусмотрены операции:

- загрузка подмодели в память;
- выгрузка подмодели из памяти;
- сохранение подмодели на диске в виде отдельного файла;
- установка защиты от модификации;
- замена подмодели в памяти на новую.

Наиболее эффективно групповая работа организуется при интеграции Rational Rose со специальными средствами управления конфигурацией и контроля версий (PVCS). В этом случае защита от модификации устанавливается на все управляемые подмодели, кроме тех, которые выделены конкретному разработчику. В этом случае признак защиты от записи устанавливается для файлов, которые содержат подмодели, поэтому при считывании "чужих" подмоделей защита их от модификации сохраняется и случайные воздействия окажутся невозможными.

Среда функционирования

Rational Rose функционирует на различных платформах: IBM PC (в среде Windows), Sun SPARC stations (UNIX, Solaris, SunOS), Hewlett-Packard (HP UX), IBM RS/6000 (AIX).

Для работы системы необходимо выполнение следующих требований:

- Платформа Windows - процессор 80386SX или выше (рекомендуется 80486), память8Мб

(рекомендуется 12Мб), пространство на диске 8Мб + 1-3Мб для одной модели.

- Платформа UNIX - память 32+(16*число пользователей)Мб, пространство на диске 30Мб + 20 при инсталляции + 1-3Мб для одной модели.

  Совместимость по версиям обеспечивается на уровне моделей.

5.6. Вспомогательные средства поддержки жизненного цикла ПО

5.6.1. Средства конфигурационного управления

Цель конфигурационного управления (КУ) - обеспечить управляемость и контролируемость процессов разработки и сопровождения ПО. Для этого необходима точная и достоверная информация о состоянии ПО 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.

PVCS Version Manager [18] предназначен для управления всеми компонентами проекта и ведения планомерной многоверсионной и многоплатформенной разработки силами команды разработчиков в условиях одной или нескольких локальных сетей. Понятие "проект" трактуется как совокупность файлов. В процессе работы над проектом промежуточное состояние файлов периодически сохраняется в архиве проекта, ведутся записи о времени сохранения, соответствии друг другу нескольких вариантов разных файлов проекта. Кроме этого, фиксируются имена разработчиков, ответственных за тот или иной файл, состав файлов промежуточных версий проекта и др. Это позволяет вернуться при необходимости к какому-либо из предыдущих состояний файла (например, при обнаружении ошибки, которую в данный момент трудно исправить).

PVCS Version Manager предназначен для использования в рабочих группах. Система блокировок, реализованная в PVCS Version Manager позволяет предотвратить одновременное внесение изменений в один и тот же файл. В то же время, PVCS Version Manager позволяет разработчикам работать с собственными версиями общего файла с полуавтоматическим разрешением конфликтов между ними.

Доступ к архивам PVCS Version Manager возможен не только через сам Version Manager, но и из более чем 50 инструментальных средств, в том числе MS Visual C и MS Visual Basic, Uniface, PowerBuilder, SQL Windows, JAM, Delphi, Paradox и др.

Результатом работы PVCS Version Manager является созданный средствами файловой системы репозиторий, хранящий в компактной форме все рабочие версии программного продукта вместе с необходимыми комментариями и метками.

PVCS Version Manager функционирует в среде MS Windows, Windows 95, Windows NT, OS/2, SunOS, Solaris, HP-UX, AIX и SCO UNIX и может исполняться на любом персональном компьютере с процессором 80386 или выше, рабочих станциях Sun, HP и IBM (RS-6000).

Другим средством конфигурационного управления является PVCS Tracker [19] - специализированная надстройка над офисной электронной почтой, предназначенная для обработки сообщений об ошибках в продукте, доставке их исполнителям и контроля за исполнением. Интеграция с PVCS Version Manager дает возможность связывать с сообщениями те или иные компоненты проекта. Отчетные возможности PVCS Tracker включают множество разновидностей графиков и диаграмм, отражающих состояние проекта и процесса его отладки, срезы по различным компонентам проекта, разработчикам и тестировщикам. С их помощью можно наглядно показать текущее состояние работы над проектом и ее временные тенденции.

Персонал, работающий с PVCS Tracker делится на пять групп в зависимости от их обязанностей: пользователи, разработчики, группа тестирования и контроля качества, группа

технической поддержки и сопровождения, управленческий персонал. Этим пяти группам персонала соответствуют пять предопределенных групп PVCS Tracker:

- пользователи (Submitters) - имеют ограниченные права на внесение замечаний и сообщений об ошибках в базу данных PVCS Tracker;
- разработчики (Development Engineers) - имеют право производить основные операции с требованиями и замечаниями в базе данных PVCS Tracker. Если разработчики делятся на подгруппы, то для каждой подгруппы могут быть заданы отдельные списки прав доступа;
- тестировщики (Quality Engineers) - имеют право производить основные операции с требованиями и замечаниями;
- сопровождение (Support Engineers) - имеют право вносить любые замечания, требования и рекомендации в базу данных, но не имеют прав по распределению работ и изменению их приоритетности и сроков исполнения;
- руководители (Managers) - имеют право распределять работы между исполнителями и принимать решения о их надлежащем исполнении. Руководителям разных групп могут заданы различные права доступа к базе данных PVCS Tracker.

В дополнение к этим пяти предопределенным группам, существует группа администратора базы данных и 11 дополнительных групп, которые могут быть настроены в соответствии со специфическими должностными обязанностями сотрудников, использующих PVCS Tracker.

Требование или замечание поступающее в PVCS Tracker проходит четыре этапа обработки:

- регистрация - внесение замечания в базу данных;
- распределение - назначение ответственного исполнителя и сроков исполнения;
- исполнение - устранение замечания, которое в свою очередь может вызвать дополнительные замечания или требования на дополнительные работы;
- приемка - приемка работ и снятие их с контроля или направление на доработку.

Требования и замечания, поступающие в базу данных PVCS Tracker оформляются в виде специальной формы, которая может содержать до 18 полей выбора стандартных значений и до 12 произвольных текстовых строк. При разработке формы следует определить оптимальный набор информации, характерный для всех записей в базе данных.

Для получения содержательной информации о ходе разработки PVCS Tracker позволяет получать три типа статистических отчетов: частотные, тренды и диаграммы распределения.

Частотные отчеты содержат информацию о частоте поступающих замечаний за один час тестирования программного продукта. Однако универсального частотного отчета не существует, т.к. на оценку качества влияют тип методов тестирования, серьезность выявленных ошибок и значение дефектных модулей для функционирования всей системы. Малое число фатальных ошибок, приводящих к полной остановке разработки, хуже большого числа замечаний к внешнему виду интерфейса пользователя. Следовательно, частотные отчеты должны быть настроены на выявление какого-либо конкретного аспекта качества для того, чтобы их можно было использовать для прогнозирования окончания работ над проектом.

Тренды содержат информацию об изменениях того или иного показателя во времени и характеризуют стабильность и непрерывность процесса разработки. Они позволяют ответить на вопросы:

- успевает ли группа разработчиков справляться с поступающими замечаниями;
- улучшается ли качество программного продукта и какова динамика этого процесса;
- как повлияло то или иное решение (увеличение числа разработчиков, введение скользящего графика, внедрение нового метода тестирования) на работу группы и т.п.

Диаграммы распределения - наиболее разнообразные и полезные для осуществления оперативного руководства формы отчетов. Они позволяют ответить на вопросы: какой метод тестирования более эффективен, какие модули вызывают наибольшее число нареканий, кто из разработчиков лучше справляется с конкретным типом заданий, нет ли перекоса в распределении работ между исполнителями, нет ли модулей, тестированию которых было уделено недостаточно внимания и т.д.

PVCS Tracker предназначен для использования в рабочих группах, объединенных в общую сеть. В этом случае центральная база или проект PVCS Tracker находится на общедоступном

сервере сети, доступ к которому реализуется посредством ODBC-драйверов, входящих в состав PVCS Tracker. Главной особенностью PVCS Tracker по сравнению с обычным приложением СУБД является его способность автоматически уведомлять пользователя о поступлении интересующей его или относящейся к его компетенции информации и гибкая система распределения полномочий внутри рабочей группы. При необходимости PVCS Tracker может использовать для уведомления удаленных членов группы электронную почту.

PVCS Tracker поддерживает групповую работу в локальных сетях и взаимодействует с СУБД dBase, ORACLE, SQL Server и SYBASE посредством ODBC.

PVCS Tracker может быть интегрирован с любой системой электронной почты, поддерживающей стандарты VIM, MAPI или SMTP.

PVCS Version Manager и PVCS Tracker окружены вспомогательными компонентами: PVCS Configuration Builder и PVCS Notify.

PVCS Configuration Builder предназначен для сборки окончательного продукта из компонент проекта. PVCS Configuration Builder позволяет описывать процесс сборки как на стандартном языке MAKE, так и на собственном внутреннем языке, имеющем существенно большие возможности. PVCS Configuration Builder позволяет осуществлять сборку программного продукта на основании файлов, хранящихся в репозитории PVCS Version Manager.

Обычная процедура сборки программного продукта с помощью PVCS Configuration Builder состоит из трех шагов:

- строится файл зависимостей между исходными модулями;
- в полученный файл вносятся изменения с целью его настройки и оптимизации;
- осуществляется сборка программного продукта из исходных модулей.

Результатом работы PVCS Configuration Builder является специальный файл, описывающий оптимальный алгоритм сборки программного продукта, построенный на основе анализа дерева зависимостей между исходными модулями.

PVCS Notify обеспечивает автоматическую рассылку сообщений об ошибках из базы данных пакета PVCS Tracker по рабочим станциям назначения. При этом используется офисная система электронной почты cc:Mail или Microsoft Mail. PVCS Notify расширяет возможности PVCS Tracker и используется только совместно с ним.

PVCS Notify настраивается из среды PVCS Tracker. Настройка включает в себя определение интервала времени, через который PVCS Notify проверяет содержимое базы данных, определение критериев отбора записей для рассылки уведомлений, определение списков адресов для рассылки. После настройки PVCS Notify начинает работу в автономном режиме, автоматически рассылая уведомления об изменениях в базе данных PVCS Tracker.

PVCS Notify предназначен для использования в больших рабочих группах, часть членов которых хотя и доступна только через средства электронной почты, однако должна иметь оперативную информацию о требованиях на изменение программного продукта, замечаниях, ошибках, ходе и результатах его тестирования.

Результатом работы PVCS Notify являются оформленные в соответствии с одним из стандартов почтовые сообщения, готовые для рассылки посредством системы электронной почты.

5.6.2. Средства документирования

Для создания документации в процессе разработки ИС 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 ПО. 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.

SoDA не зависит от применяемых инструментальных средств. Связь с приложениями осуществляется через стандартный программный интерфейс API. Переход на новые инструментальные средства не влечет за собой дополнительных затрат по документированию проекта.

SoDA содержит набор шаблонов документов, определяемых стандартом на программное обеспечение DOD 2167A. На их основе можно без специального программирования создавать новые формы документов, определяемые пользователями.

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. Разные версии документации могут быть для наглядности отмечены своими отличительными признаками. В системе создаются таблицы требований к проекту, по которым можно проследить, как реализуются эти требования. Разные виды документации, сопровождающие различные этапы ЖЦ, связаны между собой, и можно проследить состояние проекта от первоначальных требований до анализа, проектирования, кодирования и тестирования программного продукта.

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.

SoDA требует по крайней мере 32 MB оперативной памяти, 100-300 MB для установки и 64 MB рабочего пространства на диске.

5.6.3. 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) [20] 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.

Основными компонентами QA являются:

- QA Partner - среда для разработки, компиляции и выполнения тестов;
- QA Planner - модуль для разработки планов тестирования и обработки результатов. Для создания и выполнения тестов в процессе работы QA Planner вызывается QA Partner;
- Agent - модуль, поддерживающий работу в сети.

Процесс тестирования состоит из следующих этапов:

- создание плана тестирования;
- связывание плана с тестами;
- пометка и выполнение тестов;
- получение отчетов о тестировании и управление результатами.

Создание тестового плана в QA Planner включает в себя составление схемы тестовых требований и выделение уровней детализации. Для этого необходимо определить все, что должно быть протестировано, подготовить функциональную декомпозицию приложения, оценить, сколько тестов необходимо для каждой функции и характеристики, определить, сколько из них

будет реализовано в зависимости от доступных ресурсов и времени. Эта информация используется для создания схемы тестовых требований.

Для связывания плана с тестами необходимо создать управляющие предложения (скрипты) на специальном языке 4Test и тесты, которые выполняют требования плана, и связать компоненты любым способом. Для избежания перегруженности тестов используют управление тестовыми данными.

При выполнении плана результаты записываются в формате, похожем на план. Все результаты связаны с планом. Есть возможность просмотреть или скрыть общую информацию о выполнении, слить файлы результатов, разметить неудавшиеся тесты, сравнить результаты предыдущего выполнения тестов, выполнить или отменить отчет.

Одним из атрибутов теста является имя его разработчика, что позволяет при необходимости выполнять тесты, созданные конкретным разработчиком.

Комплекс QA занимает на жестком диске не более 21MB. Поддерживаемые платформы: Windows 3.x, Windows 95, Windows NT, OS/2, Macintosh, VMS, HP-UX, AIX, Solaris.

5.7. Примеры комплексов CASE-средств

В заключение приведем примеры комплексов CASE-средств обеспечивающих поддержку полного ЖЦ ПО. Здесь хотелось бы еще раз отметить нецелесообразность сравнения отдельно взятых CASE-средств, поскольку ни одно из них не решает в целом все проблемы создания и сопровождения ПО. Это подтверждается также полным набором критериев оценки и выбора, которые затрагивают все этапы ЖЦ ПО. Сравниваться могут комплексы методологически и технологически согласованных инструментальных средств, поддерживающие полный ЖЦ ПО и обеспеченные необходимой технической и методической поддержкой со стороны фирм-поставщиков. По мнению автора, на сегодняшний день наиболее развитым из всех поставляемых в России комплексов такого рода является комплекс технологий и инструментальных средств создания ИС, основанный на методологии и технологии DATARUN. В состав комплекса входят следующие инструментальные средства:

- CASE-средство Silverrun;
- средство разработки приложений JAM;
- мост Silverrun-RDM <-> JAM;
- комплекс средств тестирования QA;
- менеджер транзакций Tuxedo;
- комплекс средств планирования и управления проектом SE Companion;
- комплекс средств конфигурационного управления PVCS;
- объектно-ориентированное CASE-средство Rational Rose;
- средство документирования SoDA.

Примерами других подобных комплексов являются:

- Vantage Team Builder for Uniface + Uniface (фирмы "DataX/Florin" и "ЛАНИТ");
- комплекс средств, поставляемых и используемых фирмой "ФОРС":
- CASE-средства Designer/2000 (основное), ERwin, Bpwin и Oowin (альтернативные);
- средства разработки приложений Developer/2000, ORACLE Power Objects (основные) и Usoft Developer (альтернативное);
- средство настройки и оптимизации ExplainSQL (Platinum);
- средства администрирования и сопровождения SQLWatch, DBVision, SQL Spy, TSReorg и др. (Platinum);
- средство документирования ORACLE Book.
- комплекс средств на основе продуктов фирмы CENTURA:
- CASE-средства ERwin, Bpwin и Oowin (объектно-ориентированный анализ);
- средства разработки приложений SQLWindows и TeamWindows;
- средство тестирования и оптимизации приложений "клиент-сервер" SQLBench (ARC);
- средства эксплуатации и сопровождения Quest и Crystal Reports.

Литература

1. Вендров А.М. Один из подходов к выбору средств проектирования баз данных и приложений. "СУБД", 1995, №3.
2. Зиндер Е.З. Бизнес-реинжиниринг и технологии системного проектирования. Учебное пособие. М., Центр Информационных Технологий, 1996
3. Калянов Г.Н. CASE. Структурный системный анализ (автоматизация и применение). М., "Лори", 1996.
4. Марка Д.А., МакГоуэн К. Методология структурного анализа и проектирования. М., "МетаТехнология", 1993.
5. Международные стандарты, поддерживающие жизненный цикл программных средств. М., МП "Экономика", 1996
6. Создание информационной системы предприятия. "Computer Direct", 1996, N2
7. Шлеер С., Меллор С. Объектно-ориентированный анализ: моделирование мира в состояниях. Киев, "Диалектика", 1993.
8. Barker R. CASE*Method. Entity-Relationship Modelling. Copyright Oracle Corporation UK Limited, Addison-Wesley Publishing Co., 1990.
9. Barker R. CASE*Method. Function and Process Modelling. Copyright Oracle Corporation UK Limited, Addison-Wesley Publishing Co., 1990.
10. Boehm B.W. A Spiral Model of Software Development and Enhancement. ACM SIGSOFT Software Engineering Notes, Aug. 1986
11. Chris Gane, Trish Sarson. Structured System Analysis. Prentice-Hall, 1979.
12. Edward Yourdon. Modern Structured Analysis. Prentice-Hall, 1989.
13. Tom DeMarco. Structured Analysis and System Specification. Yourdon Press, New York, 1978.
14. Westmount I-CASE User Manual. Westmount Technology B.V., Netherlands, 1994.
15. Uniface V6.1 Designers' Guide. Uniface B.V., Netherlands, 1994.
16. IEEE Std 1348-1995. IEEE Recommended Practice for the Adoption of CASE Tools.
17. IEEE Std 1209-1992. IEEE Recommended Practice for the Evaluation and Selection of CASE Tools.
18. PVCS Version Manager. User's Guide.
19. PVCS Tracker. User's Guide.
20. QA Partner. User's Guide.
21. Новоженов Ю.В. Объектно-ориентированные технологии разработки сложных программных систем. М., 1996.
22. Панащук С.А. Разработка информационных систем с использованием CASE-системы Silverrun. "СУБД", 1995, №3.
23. Горчинская О.Ю. Designer/2000 - новое поколение CASE-продуктов фирмы ORACLE. "СУБД", 1995, №3.
24. Горин С.В., Тандоев А.Ю. Применение CASE-средства Erwin 2.0 для информационного моделирования в системах обработки данных. "СУБД", 1995, №3.
25. Горин С.В., Тандоев А.Ю. CASE-средство S-Designor 4.2 для разработки структуры базы данных. "СУБД", 1996, №1.
26. DATARUN Concepts. Computer Systems Advisers Research Ltd., 1994.
27. SE Companion Installation and Administration Manual. SECA Inc., 1995.
28. Петров Ю.К. JAM - инструментальное средство разработки приложений в информационных системах архитектуры "клиент/сервер", построенных на базе РСУБД. "СУБД", 1995, №3.

Фирмы-поставщики CASE-средств

Наименование фирмы	Телефон	Перечень поставляемых средств
"Алконс-Софт"	(095) 362-74-43, 362-51-38, 918-13-80	S-Designor, ERwin, PowerBuilder
АО "Аргуссофт Компани"	(095) 288-36-02, 288-21-45, 288-15-36	DATARUN, Silverrun, PVCS, JAM, QA, SoDA, Rational Rose, SE-Companion
"DataX/Florin"	(095) 158-95-20	Vantage Team Builder, ObjectTeam, New Era
"Интерфейс"	(095) 135-55-00, 135-25-19	BPwin, ERwin, SQL Windows, Designer/2000, Developer/2000, DataBase Designer, Visible Analyst Workbench, EasyCASE, CASE.Аналитик, S-Designor
АО "ЛАНИТ"	(095) 267-30-38	Uniface, CASE /4/0
"ЛОИС" (Пермь)	(3422) 48-69-09	Vantage Team Builder, ObjectTeam, New Era
"LVS"	(095) 330-16-06, 330-37-33	Designer/2000, Developer/2000, DataBase Designer
МетаТехнология"	(095) 253-38-22	Design/IDEF, PowerBuilder
"МакроПроджект"	(095) 233-50-47	CASE.Аналитик, ERwin
"ORACLE-СНГ"	(095) 258-41-80	Designer/2000, Developer/2000, DataBase Designer
"Родник Софт"	(095) 113-70-01	SoDA, Rational Rose
"ФОРС"	(095) 973-43-07	Designer/2000, Developer/2000, DataBase Designer, ERwin
АО "ЭКСИНТ"	(095) 242-94-90	PRO-IV WORKBENCH