Государственный Университет Молдовы

Факультет Математики и Информатики

Департамент Информатики

“Tehnologii CASE”

Эссе

Преподаватель: Гладей Анатолий

Студент: Маруневич Николай

Кишинев 2022

**Введение**

За последнее десятилетие сформировалось новое направ­ление в программотехнике — CASE (Computer-Aided Software/System Engineering) — в дословном переводе — разработка программного обеспечения информационных сис­тем при поддержке (с помощью) компьютера. В настоящее время не существует общепринятого определения CASE, тер­мин CASE используется в весьма широком смысле. Первона­чальное значение термина CASE, ограниченное вопросами автоматизации разработки только лишь программного обес­печения, в настоящее время приобрело новый смысл, охва­тывающий процесс разработки сложных автоматизированных информационных систем в целом. Теперь под термином CASE-средства понимаются программные средства, поддерживаю­щие процессы создания и сопровождения ИС, включая анализ и формулировку требований, проектирование приклад­ного 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 АИС.

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

♦ обеспечение разработки делового и коммерческого ПО, широкое применение CASE-технологий обусловлены массовостью этой прикладной области, в которой CASE применяется не только для разработки ПО, но и для создания моделей систем, помогающих решать задачи стратегического планирования, управления финансами, определения политики фирм, обучения персонала и др. (это направление получило свое собственное на­звание — бизнес-анализ);

♦ разработка системного и управляющего ПО. Активное применение CASE-технологий связано с большой слож­ностью данной проблематики и со стремлением повы­сить эффективность работ.

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

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

♦ улучшают качество создаваемого ПО за счет средств автоматического контроля (прежде всего контроля про­екта);

♦ позволяют за короткое время создавать прототип буду­щей системы, что позволяет на ранних этапах оценить ожидаемый результат;

♦ ускоряют процесс проектирования и разработки;

♦ освобождают разработчика от рутинной работы, позво­ляя ему целиком сосредоточиться на творческой части разработки;

♦ поддерживают развитие и сопровождение разработки;

♦ поддерживают технологии повторного использования компонента разработки.

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

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

**1. Жизненный цикл программного обеспечения информационной системы**

Одним из базовых понятий методологии проектирования АИС 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 [6].

Структура ЖЦ ПО 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 организовывать, 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.

Каждый процесс характеризуется определенными зада­чами и методами их решения, исходными данными, получен­ными на предыдущем этапе, 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.

Существующие модели ЖЦ определяют порядок испол­нения этапов в ходе разработки, а также критерии перехода от этапа к этапу. В соответствии с этим наибольшее распрос­транение получили три следующие модели ЖЦ:

♦ каскадная модель (1970—1980 гг.) — предлагает пере­ход на следующий этап после полного окончания работ по предыдущему этапу;

♦ поэтапная модель с промежуточным контролем (1980—1985 гг.) — итерационная модель разработки ПО с циклами обратной связи между этапами. Преимуще­ство такой модели заключается в том, что межэтап­ные корректировки обеспечивают меньшую трудоем­кость по сравнению с каскадной моделью, однако вре­мя жизни каждого из этапов растягивается на весь пе­риод разработки;

♦ спиральная модель (1986—1990 гг.) — делает упор на начальные этапы ЖЦ: анализ требований, проектиро­вание спецификаций, предварительное и детальное про­ектирование. На этих этапах проверяется и обосновыва­ется реализуемость технических решений путем созда­ния прототипов. Каждый виток спирали соответствует поэтапной модели создания фрагмента или версии про­граммного изделия, на нем уточняются цели и характе­ристики проекта, определяется его качество, планиру­ются работы следующего витка спирали. Таким обра­зом, углубляются и последовательно конкретизируют­ся детали проекта и в результате выбирается обосно­ванный вариант, который доводится до реализации.

Специалистами отмечаются следующие преимущества спиральной модели:

♦ накопление и повторное использование программных средств, моделей и прототипов;

♦ ориентация на развитие и модификацию ПО в процес­се его проектирования;

♦ анализ риска и издержек в процессе проектирования.

Главная особенность индустрии создания ПО состоит в концентрации сложности на начальных этапах ЖЦ (анализ, проектирование) при относительно невысокой сложности и трудоемкости последующих этапов. Более того, нерешенные вопросы и ошибки, допущенные на этапах анализа и проек­тирования, порождают на последующих этапах трудные, ча­сто неразрешимые проблемы и, в конечном счете, приводят к неуспеху всего проекта.

**2. RAD** **-технологии прототипного создания приложений**

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 три элемента:

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

♦ короткий, но тщательно проработанный производствен­ный график (от 2 до б мес);

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

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

Жизненный цикл ПО по методологии RAD состоит из четырех фаз:

♦ фазы анализа и планирования требований;

♦ фазы проектирования;

♦ фазы построения;

♦ фазы 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 АИС, 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 АИС на подсистемы, поддающиеся реализации одной командой раз­работчиков за приемлемое для RAD-проектов время — примерно 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 репо-зитория 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 АИС не является абсо­лютной. Возможны различные варианты, зависящие, напри­мер, от начальных условий, в которых ведется разработка: разрабатывается ли совершенно новая система; было ли про­ведено информационное обследование организации и суще­ствует ли модель ее деятельности; существует ли в органи­зации некоторая АИС, которая может быть использована в качестве начального прототипа или должна быть интегриро­вана с разрабатываемой и т. п.

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 временил), 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. Основные принципы методоло­гии RAD:

♦ разработка приложений итерациями;

♦ необязательность полного завершения работ на каж­дом из этапов жизненного цикла;

♦ обязательное вовлечение пользователей в процесс раз­работки АИС;

♦ необходимое применение CASE-средств, обеспечива­ющих целостность проекта;

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

♦ необходимое использование генераторов кода;

♦ использование прототипирования, позволяющего 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.

**3. Структурный метод разработки программного обеспечения**

Сущность структурного подхода к разработке АИС 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 методологии структурного анализа базируются на ряде общих принципов, часть из которых регламентирует организацию работ на начальных этапах ЖЦ, а часть исполь­зуется при выработке рекомендаций по организации работ. В качестве двух базовых принципов используются следую­щие: принцип "разделяй и властвуй" и принцип иерархического упорядочивания. Первый является принципом решения трудных проблем путем разбиения их на множество мень­ших независимых задач, более легких для понимания и ре­шения. Второй принцип декларирует, что устройство этих частей также существенно для понимания. Уровень уяснения проблемы резко повышается при представлении ее частей в виде древовидных иерархических структур, т. е. система мо­жет быть понята и построена по уровням, каждый из кото­рых добавляет новые детали.

Выделение двух базовых принципов инженерии программ­ного обеспечения не означает, что остальные принципы яв­ляются второстепенными, игнорирование любого из них мо­жет привести к непредсказуемым последствиям (в том числе и к неуспеху всего проекта). Отметим основные из таких прин­ципов.

1. Принцип абстрагирования — заключается в выделе­нии существенных с некоторых позиций аспектов системы и отвлечении от несущественных с целью представления про­блемы в простом общем виде.

2. Принцип формализации — заключается в необходимо­сти строгого методического подхода к решению проблемы.

3. Принцип "упрятывания" — заключается в упрятыва­нии несущественной на конкретном этапе информации: каж­дая часть "знает" только необходимую ей информацию.

4. Принцип концептуальной общности — заключается в следовании единой философии на всех этапах ЖЦ (структур­ный анализ — структурное проектирование — структурное программирование — структурное тестирование).

5. Принцип полноты — заключается в контроле присут­ствия лишних элементов.

6. Принцип непротиворечивости — заключается в обо­снованности и согласованности элементов.

7. Принцип логической независимости — заключается в концентрации внимания на логическом проектировании для обеспечения независимости от физического проектирования.

8. Принцип независимости данных — заключается в том, что модели данных должны быть проанализированы и спро­ектированы независимо от процессов их логической обработ­ки, а также от их физической структуры и распределения.

9. Принцип структурирования данных — заключается в том, что данные должны быть структурированы и иерар­хически организованы.

10. Принцип доступа конечного пользователя — заклю­чается в том, что пользователь должен иметь средства дос­тупа к базе данных, которые он может использовать непос­редственно (без программирования).

Соблюдение указанных принципов необходимо при орга­низации работ на начальных этапах ЖЦ независимо от типа разрабатываемого ПО и используемых при этом методологий. Руководствуясь всеми принципами в комплексе, можно на более ранних стадиях разработки понять, что будет пред­ставлять собой создаваемая система, обнаружить промахи и недоработки, что, в свою очередь, облегчит работы на после­дующих этапах ЖЦ и понизит стоимость разработки.

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;

♦ DFD (Data Flow Diagrams) — диаграммы потоков дан­ных;

♦ ERD (Entity-Relationship Diagrams) — диаграммы"сущ-ность—связь";

♦ STD (State Trasition Diagrams) — диаграммы переходов состояний.

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 АИС независимо от того, является ли она существую­щей или вновь разрабатываемой. Состав диаграмм в каждом конкретном случае зависит от необходимой полноты описа­ния системы.

*Методология SADT*

Методология SADT разработана Дугласом Россом, на ее основе разработана, в частности, известная методология IDEFO (Icam Definition), которая является основной частью программы Icam (Интеграция компьютерных и промышлен­ных технологий), проводимой по инициативе 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 описывается посредством интерфейсных дуг, выражающих "ограничения", которые, в свою очередь, определяют, когда и каким образом функции выполня­ются и управляются;

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

Правила SADT включают:

♦ ограничение количества блоков на каждом уровне де­композиции (правило 3—б 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.

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 (рис. 1.6.1).

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

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, так называемый 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. 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.

Некоторые дуги присоединены к блокам диаграммы обо­ими концами, у других же один конец остается неприсоеди-ненным. 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.

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.

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

*Моделирование потоков данных (процессов)*

Основным средством моделирования функциональных требований АИС являются диаграммы потоков данных (DFD:— Data Flow Diagrams). С их помощью эти требования разбива­ются на функциональные компоненты (процессы) и представ­ляются в виде сети, связанной потоками данных. Главная цель таких средств — продемонстрировать, как каждый процесс преобразует свои входные данные в выходные, а также вы­явить отношения между этими процессами.

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 процессы 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.

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 АИС.

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. В различных нотациях процесс может изображаться на диаграммах по-разному. 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.

В последнее время принято использовать еще и поле физической реализации, информация в котором показывает, какое подразделение организации, программа или аппарат­ное устройство выполняет данный процесс.

Хранилище (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 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. 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. Каж­дый поток данных имеет имя, отражающее его содержание.

Первым шагом при построении иерархии DFD является построение контекстных диаграмм. Обычно при проецирова­нии относительно простых АИС 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. Для сложных АИС 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 АИС как между собой, так и с внешними входными и выход­ными потоками данных и внешними объектами (источниками и приемниками информации), с которыми взаимодействует АИС.

Разработка контекстных диаграмм решает проблему стро­гого определения функциональной структуры АИС 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 DFD. Каждый про­цесс на DFD, в свою очередь, может быть детализирован при помощи DFD 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 DFD. 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 DFD 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. 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.

*Моделирование данных*

Цель моделирования данных состоит в обеспечении раз­работчика АИС 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 (см. подразд. 2.2).

Нотация ERD была впервые введена П. Ченом (P. Chen) и получила дальнейшее развитие в работах Баркера.

*Методология IDEF 1*

Метод IDEF1, разработанный Т. Рэмеем (Т. 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. IDEF IX-диаграммы используются рядом распространенных CASE-средств (в частности, ERWin, Design/IDEF).