МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ»

Проектирование инфокоммуникационных систем

Учебное пособие

Автор: Хоружников Сергей Эдуардович

Санкт-Петербург

2013

СОДЕРЖАНИЕ

Введение

Важной особенностью современной технологической ситуации в мире и в России является усиление роли инноваций на основе разработки и внедрения инфокоммуникаций, т.е. информационно-коммуникационных технологий, эффективное использование которых обеспечивает конкурентоспособность в любых областях науки и бизнеса.

В настоящее время при проектировании сложных систем активно используется методология, направленная на снижение коммерческих рисков посредством обнаружения ошибок на ранних стадиях разработки. Технические риски оцениваются и расставляются согласно приоритетам на ранних стадиях цикла разработки, а затем пересматриваются с течением времени и с развитием проекта в течение последующих итераций. Возможно появление новых целей в зависимости от приоритетов конкретных рисков.

Таким образом, в процессе проектирования наибольшее внимание уделяется начальным стадиям разработки проекта - анализу и моделированию.

Стандартным инструментом для реализации этих задач является унифицированный язык моделирования (UML). С помощью UML можно визуализировать, специфицировать, конструировать и документировать различные результаты анализа и проектирования инфокоммуникационных систем.

Если говорить о прямом назначении языка UML, то, прежде всего, следует отметить область разработки программных систем. Также использование UML особенно эффективно в следующих областях:

- информационные системы масштаба предприятия;
- банковские и финансовые услуги;
- телекоммуникации;
- транспорт;
- оборонная промышленность, авиация и космонавтика;
- розничная торговля;
- медицинская электроника;
- наука;
- распределенные Web-системы.

Таким образом, UML подходит для моделирования любых систем: от информационных систем масштаба предприятия до распределенных Web-приложений и встроенных систем реального времени. Это выразительный язык, позволяющий рассмотреть систему со всех точек зрения, имеющих отношение к ее разработке и последующему развертыванию. Несмотря на обилие выразительных возможностей, этот язык прост для понимания и использования.

1. Технология проектирования инфокоммуникационных систем

1.1 Инженерное проектирование, его объекты, цели, показатели и ограничения

Проектирование технического объекта – создание, преобразование и представление в принятой форме образа этого еще не существующего объекта.

Образ объекта или его составных частей может создаваться в воображении человека в результате творческого процесса или генерироваться в соответствии с некоторыми алгоритмами в процессе взаимодействия человека и ЭВМ.

В любом случае инженерное проектирование начинается при наличии выраженной потребности общества в некоторых технических объектах, которыми могут быть объекты строительства, промышленные изделия или процессы. Проектирование включает в себя разработку технического предложения и (или) технического задания (ТЗ), отражающих эти потребности, и реализацию ТЗ в виде проектной документации.

Обычно ТЗ представляют в виде некоторых документов, и оно является исходным (первичным) описанием объекта. Результатом проектирования, как правило, служит полный комплект документации, содержащий достаточные сведения для изготовления объекта в заданных условиях.

Эта документация и есть *проект*, точнее, окончательное описание объекта.

Более коротко, **проектирование** - процесс, заключающийся в получении и преобразовании исходного описания объекта в окончательное описание на основе выполнения комплекса работ исследовательского, расчетного и конструкторского характера.

Преобразование исходного описания в окончательное порождает ряд промежуточных описаний, подводящих итоги решения некоторых задач и используемых при обсуждении и принятии проектных решений для окончания или продолжения проектирования.

Проектирование, при котором все проектные решения или их часть получают путем взаимодействия человека и ЭВМ, называют *автоматизированным*, в отличие от ручного (без использования ЭВМ) или автоматического (без участия человека на промежуточных этапах).

Система, реализующая автоматизированное проектирование, представляет собой *систему автоматизированного проектирования САПР (в англоязычном написании CAD System — Computer Aided Design System).*

Система автоматизированного проектирования (САПР) — автоматизированная система, реализующая информационную технологию выполнения функций проектирования, представляет собой организационно-

техническую систему, предназначенную для автоматизации процесса проектирования, состоящую из персонала и комплекса технических, программных и других средств автоматизации его деятельности.

Автоматическое проектирование возможно лишь в отдельных частных случаях для сравнительно несложных объектов. Превалирующим в настоящее время является автоматизированное проектирование.

Проектирование сложных объектов основано на применении идей и принципов, изложенных в ряде теорий и подходов. Наиболее общим подходом является системный подход, идеями которого пронизаны различные методики проектирования сложных систем.

По ГОСТ 23501.101-87, составными структурными частями САПР являются *подсистемы*, обладающие всеми свойствами систем и создаваемые как самостоятельные системы. Каждая подсистема — это выделенная по некоторым признакам часть САПР, обеспечивающая выполнение некоторых функционально-законченных последовательностей проектных задач с получением соответствующих проектных решений и проектных документов.

По назначению подсистемы САПР разделяют на два вида: *проектирующие* и *обслуживающие*.

- *Обслуживающие подсистемы* — объектно-независимые подсистемы, реализующие функции, общие для подсистем или САПР в целом: обеспечивают функционирование проектирующих подсистем, оформление, передачу и вывод данных, сопровождение программного обеспечения и т. п., их совокупность называют системной средой (или оболочкой) САПР.
- *Проектирующие подсистемы* — объектно-ориентированные подсистемы, реализующие определенный этап проектирования или группу связанных проектных задач. В зависимости от отношения к объекту проектирования, делятся на:
    - *Объектные* — выполняющие проектные процедуры и операции, непосредственно связанные с конкретным типом объектов проектирования.
    - *Инвариантные* — выполняющие унифицированные проектные процедуры и операции, имеющие смысл для многих типов объектов проектирования.

Примерами проектирующих подсистем могут служить подсистемы геометрического трехмерного моделирования механических объектов, схемотехнического анализа, трассировки соединений в печатных платах.

Типичными обслуживающими подсистемами являются:

- подсистемы управления проектными данными
- обучающие подсистемы для освоения пользователями технологий, реализованных в САПР
- подсистемы графического ввода-вывода

- система управления базами данных (СУБД).

Компоненты и обеспечение

Каждая подсистема, в свою очередь состоит из компонентов, обеспечивающих функционирование подсистемы.

Компонент выполняет определенную функцию в подсистеме и представляет собой наименьший (неделимый) самостоятельно разрабатываемый или покупной элемент САПР (программа, файл модели транзистора, графический дисплей, инструкция и т. п.).

Совокупность однотипных компонентов образует *средство обеспечения* САПР.

Выделяют следующие виды обеспечения САПР:

- *Техническое обеспечение* (ТО) — совокупность связанных и взаимодействующих технических средств (ЭВМ, периферийные устройства, сетевое оборудование, линии связи, измерительные средства).
- *Математическое обеспечение* (МО), объединяющее математические методы, модели и алгоритмы, используемые для решения задач автоматизированного проектирования. По назначению и способам реализации делят на две части:
    - математические методы и построенные на них математические модели;
    - формализованное описание технологии автоматизированного проектирования.
- *Программное обеспечение* (ПО). Подразделяется на *общесистемное* и *прикладное*:
    - *прикладное ПО* реализует математическое обеспечение для непосредственного выполнения проектных процедур. Включает пакеты прикладных программ, предназначенные для обслуживания определенных этапов проектирования или решения групп однотипных задач внутри различных этапов (модуль проектирования трубопроводов, пакет схемотехнического моделирования, геометрический решатель САПР).
    - *общесистемное ПО* предназначено для управления компонентами *технического обеспечения* и обеспечения функционирования *прикладных программ*. Примером компонента *общесистемного ПО* является операционная система.
- *Информационное обеспечение* (ИО) — совокупность сведений, необходимых для выполнения проектирования. Состоит из описания стандартных проектных процедур, типовых проектных решений, комплектующих изделий и их моделей, правил и норм проектирования. Основная часть ИО САПР — базы данных.

- *Лингвистическое обеспечение* (ЛО) — совокупность языков, используемых в САПР для представления информации о проектируемых объектах, процессе и средствах проектирования, а также для осуществления диалога проектировщик-ЭВМ и обмена данными между техническими средствами САПР. Включает термины, определения, правила формализации естественного языка, методы сжатия и развертывания.
    - В лингвистическом обеспечении выделяют класс различного типа языков проектирования и моделирования (VHDL, VERILOG, UML, GPSS).
- *Методическое обеспечение* (МетО) — описание технологии функционирования САПР, методов выбора и применения пользователями технологических приемов для получения конкретных результатов. Включает в себя теорию процессов, происходящих в проектируемых объектах, методы анализа, синтеза систем и их составных частей, различные методики проектирования. Иногда к МетО относят также *МО* и *ЛО*.
- *Организационное обеспечение* (ОО) — совокупность документов, определяющих состав проектной организации, связь между подразделениями, организационную структуру объекта и системы автоматизации, деятельность в условиях функционирования системы, форму представления результатов проектирования… В ОО входят штатные расписания, должностные инструкции, правила эксплуатации, приказы, положения и т. п.
- *Эргономическое обеспечение* объединяет взаимосвязанные требования, направленные на согласование психологических, психофизиологических, антропометрических характеристик и возможностей человека с техническими характеристиками средств автоматизации и параметрами рабочей среды на рабочем месте.
- *Правовое обеспечение* состоит из правовых норм, регламентирующих правоотношения при функционировании САПР, и юридический статус результатов её функционирования.

Принципы создания инфокоммуникационных систем

Давно известны основные принципы, на которые необходимо опираться в процессе создания любых сложных систем, такие как, принцип системного подхода; разумной типизации проектных решений; непрерывного развития системы; минимизации ввода-вывода информации.

Развитие технической основы создания компьютеров и IT-технологий привело к дополнению этих принципов и к ним можно отнести следующие: системность, развитие (открытость), совместимость, стандартизация (унификация) и эффективность.

Принцип системности заключается в том, что при декомпозиции должны быть установлены такие связи между структурными элементами системы, которые

обеспечивают цельность инфокоммуникационной системы и ее взаимодействие с другими системами.

Системный подход предполагает учет всех этих взаимосвязей, анализ отдельных частей системы как ее самостоятельных структурных составляющих и параллельно выявление роли каждой из них в функционировании всей системы в целом. Таким образом, реализуются процессы анализа и синтеза, фундаментальный смысл которых состоит в разложении целого на составные части и воссоединение целого из частей.

Принцип развития (открытости) заключается в том, что исходя из перспектив развития объекта автоматизации, система должна создаваться с учетом возможности пополнения и обновления функций и ее состава без нарушения ее функционирования.

Очевидно, что внесение изменений в систему, обусловленных самыми различными причинами (внедрением новых информационных технологий, изменением законодательства, организационной перестройкой внутри фирмы и т. п.), должно осуществляться только путем дополнения системы без переделки уже созданного, т. е. не нарушать ее функционирования.

Принцип совместимости заключается в том, что при создании систем должны быть реализованы информационные интерфейсы, благодаря которым она может взаимодействовать с другими системами в соответствии с установленными правилами.

Принцип стандартизации (унификации) заключается в том, что при создании систем должны быть рационально применены типовые, унифицированные и стандартизованные элементы, проектные решения, пакеты прикладных программ, комплексы, компоненты.

Задачи необходимо разрабатывать таким образом, чтобы они подходили к возможно более широкому кругу объектов.

Принцип эффективности заключается в достижении рационального соотношения между затратами на создание систем и целевыми эффектами, включая конечные результаты, получаемые в результате автоматизации.

Все перечисленные принципы естественным образом могут быть реализованы в рамках UML.

Системный подход к проектированию систем[1]

С точки зрения системного подхода к проектированию сложных систем необходимо рассматривать объект исследований как целостное множество элементов в совокупности отношений и связей между ними.

Говоря о системном подходе, можно говорить о некотором способе организации действий, таком, который охватывает любой род деятельности,

[1] Практическое занятие 2. Системное проектирование инфокоммуникационных систем

выявляя закономерности и взаимосвязи с целью их более эффективного использования.

К настоящему времени сформировалась новая проектная идеология, получившая название системного проектирования.

Системное проектирование комплексно решает поставленные задачи, принимает во внимание взаимодействие и взаимосвязь отдельных объектов-систем и их частей, как между собой, так и с внешней средой, учитывает социально-экономические и экологические последствия их функционирования. Системное проектирование основывается на тщательном совместном рассмотрении объекта проектирования и процесса проектирования, которые в свою очередь включают еще ряд важных частей, показанных на рис.1.1.

Рис. 1.1 Основные части проектирования

Основные принципы системного подхода:

- *Целостность*, позволяющая рассматривать одновременно систему как единое целое и в то же время как подсистему для вышестоящих уровней.
- *Иерархичность строения*, то есть наличие множества элементов, расположенных на основе подчинения элементов низшего уровня элементам высшего уровня.
- *Структуризация*, позволяющая анализировать элементы системы и их взаимосвязи в рамках конкретной организационной структуры. Как правило, процесс функционирования системы обусловлен не столько свойствами её отдельных элементов, сколько свойствами самой структуры.
- *Множественность*, позволяющая использовать множество и математических моделей для описания отдельных элементов и системы в целом.
- *Системность*, свойство объекта обладать всеми признаками системы.

Таким образом, процесс визуализации, специфицирования, конструирования и документирования инфокоммуникационных систем необходимо рассматривать с различных точек зрения. Все, кто имеет отношение к проекту, например, конечные пользователи, аналитики, разработчики, системные интеграторы, технические писатели и менеджеры проектов, преследуют собственные интересы, и каждый смотрит на создаваемую систему, по-разному в различные моменты ее жизни. Системная архитектура является наиболее важным артефактом, который

используется для управления всевозможными точками зрения и тем самым способствует итеративной и инкрементной разработке системы на всем протяжении ее жизненного цикла.

Можно сформулировать понятие архитектуры как совокупность существенных решений относительно:

- организации системы в целом;
- выбора структурных элементов, составляющих систему;
- поведения этих элементов, специфицированного в кооперациях с другими элементами;
- составления из этих структурных и поведенческих элементов все более и более крупных подсистем;
- архитектурного стиля, направляющего и определяющего всю организацию системы: статические и динамические элементы, их интерфейсы, кооперации и способ их объединения.

1.2 Общая схема проектирования инфокоммуникационных систем

Для большинства приложений характерно следующее наиболее крупное выделение уровней:

• системный уровень, на котором решают наиболее общие задачи проектирования систем, машин и процессов; результаты проектирования представляют в виде структурных схем, генеральных планов, схем размещения оборудования, диаграмм потоков данных и т. п.;

• макроуровень, на котором проектируют отдельные устройства, узлы машин и приборов; результаты представляют в виде функциональных, принципиальных и кинематических схем, сборочных чертежей и т. п.;

• микроуровень, на котором проектируют отдельные детали и элементы машин и приборов.

В каждом приложении число выделяемых уровней и их наименования могут быть различными. Так, в радиоэлектронике микроуровень часто называют компонентным, макроуровень - схемотехническим. Между схемотехническим и системным уровнями вводят уровень, называемый функционально логическим. В вычислительной технике системный уровень подразделяют на уровни проектирования ЭВМ (вычислительных систем) и вычислительных сетей. В машиностроении имеются уровни деталей, узлов, машин, комплексов.

В зависимости от последовательности решения задач иерархических уровней различают нисходящее, восходящее и смешанное проектирование (стили проектирования).

Последовательность решения задач от нижних уровней к верхним характеризует восходящее проектирование, обратная последовательность приводит к нисходящему проектированию, в смешанном стиле имеются элементы как восходящего, так и нисходящего проектирования.

В большинстве случаев для сложных систем предпочитают нисходящее проектирование. Отметим, однако, что при наличии заранее спроектированных составных блоков (устройств) можно говорить о смешанном проектировании.

Неопределенность и нечеткость исходных данных при нисходящем проектировании (так как еще не спроектированы компоненты) или исходных требований при восходящем проектировании (поскольку ТЗ имеется на всю систему, а не на ее части) обусловливают необходимость прогнозирования недостающих данных с последующим их уточнением, т. е. последовательного приближения к окончательному решению (итерационность проектирования).

Наряду с декомпозицией описаний на иерархические уровни применяют разделение представлений о проектируемых объектах на аспекты.

Аспект описания (страта) — описание системы или ее части с некоторой оговоренной точки зрения, определяемой функциональными, физическими или иного типа отношениями между свойствами и элементами.

Различают функциональный, информационный, структурный и поведенческий (процессный) аспекты.

Функциональное описание относят к функциям системы и чаще всего представляют его функциональными схемами.

Информационное описание включает в себя основные понятия предметной области (сущности), словесное пояснение или числовые значения характеристик (атрибутов) используемых объектов, а также описание связей между этими понятиями и характеристиками. Информационные модели можно представлять графически (графы, диаграммы сущность - отношение), в виде таблиц или списков.

Структурное описание относится к морфологии системы, характеризует составные части системы и их межсоединения и может быть представлено структурными схемами, а также различного рода конструкторской документацией.

Поведенческое описание характеризует процессы функционирования (алгоритмы) системы и (или) технологические процессы создания системы. Иногда аспекты описаний связывают с подсистемами, функционирование которых основано на различных физических процессах.

Отметим, что в общем случае выделение страт может быть неоднозначным. Так, помимо указанного подхода очевидна целесообразность выделения таких аспектов, как функциональное (разработка принципов действия, структурных, функциональных, принципиальных схем), конструкторское (определение форм и пространственного расположения компонентов изделий), алгоритмическое (разработка алгоритмов и программного обеспечения) и технологическое (разработка технологических процессов) проектирование систем. Примерами страт в случае САПР могут служить также рассматриваемые далее виды обеспечения автоматизированного проектирования.

1.3 Этапы создания инфокоммуникационных систем

Стадии проектирования — наиболее крупные части проектирования как процесса, развивающегося во времени.

В общем случае выделяют стадии научно-исследовательских работ (НИР), эскизного проекта или опытно-конструкторских работ, технического, рабочего проектов, испытаний опытных образцов или опытных партий. Стадию НИР иногда называют предпроектными исследованиями или стадией технического предложения. Очевидно, что по мере перехода от стадии к стадии степень подробности и тщательность проработки проекта возрастают, и рабочий проект должен быть вполне достаточным для изготовления опытных или серийных образцов. Близким к определению стадии, но менее четко оговоренным понятием является понятие этапа проектирования.

Стадии (этапы) проектирования подразделяют на составные части, называемые проектными процедурами. Примерами проектных процедур могут служить подготовка деталировочных чертежей, анализ кинематики, моделирование переходного процесса, оптимизация параметров и другие проектные задачи. В свою очередь, проектные процедуры можно расчленить на более мелкие компоненты, называемые проектными операциями, например, при анализе прочности детали сеточными методами операциями могут быть построение сетки, выбор или расчет внешних воздействий, собственно моделирование полей напряжений и деформаций, представление результатов моделирования в графической и текстовой формах. Проектирование сводится к выполнению некоторых последовательностей проектных процедур - маршрутов проектирования.

Стадии и этапы работы согласно этой технологии описаны в действующем стандарте ГОСТ 34.601-90 (переиздание 01.09.2009).

Отметим те стадии, при реализации которых полезно бы использование тех или иных диаграмм UML (более подробно о диаграммах будет говориться в последующих лекциях). UML как язык специфицирования позволяет построить точные, недвусмысленные и полные модели. UML позволяет специфицировать все существенные решения, касающиеся анализа, проектирования и реализации, которые должны приниматься в процессе разработки и развертывания систем.

Стадия 1. Формирование требований к системе.

На этой стадии проектирования выделяют следующие этапы работ:

- обследование объекта и обоснование необходимости создания системы;
- формирование требований пользователей к системе;
- оформление отчета о выполненной работе и тактико-технического задания на разработку.

Стадия 2. Разработка концепции.

- изучение объекта автоматизации;
- проведение необходимых научно-исследовательских работ;

- разработка вариантов концепции системы, удовлетворяющих требованиям пользователей;
- оформление отчета и утверждение концепции.

Стадия 3. Техническое задание.

- разработка и утверждение технического задания на создание систем.

Стадия 4. Эскизный проект.

- разработка предварительных проектных решений по системе и ее частям;
- разработка эскизной документации на систему в целом и ее части.

Стадия 5. Технический проект.

- разработка проектных решений по системе и ее частям;
- разработка документации на систему и ее части;
- разработка и оформление документации на поставку комплектующих изделий;
- разработка заданий на проектирование в смежных частях проекта.

Для достижения сформулированных целей на первом этапе полезно построить модель в форме диаграммы вариантов использования (use case diagram), которая описывает функциональное назначение системы или, другими словами, то, что система будет делать в процессе своего функционирования. Диаграмма вариантов использования является исходным концептуальным представлением или концептуальной моделью системы в процессе ее проектирования и разработки.

При выборе объектно-ориентированного подхода к проектированию обязательной является диаграмма классов (class diagram), которая служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Диаграмма классов может отражать, в частности, различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывает их внутреннюю структуру и типы отношений.

Следующей диаграммной, которая может быть построена, является диаграмма последовательности (sequence diagram). С помощью этой диаграммы описывают поведение взаимодействующих групп объектов. Как правило, для каждого случая (прецедента) составляется своя диаграмма последовательности. На диаграмме отображаются объекты, сообщения, которыми они обмениваются, и временная шкала обмена этими сообщениями. По горизонтали можно проследить последовательность возникновения сообщений. По вертикали для каждого объекта отображается набор входящих и исходящих сообщений в порядке их возникновения.

На заключительных этапах разработки могут быть реализованы диаграммы компонентов и развертывания. На диаграмме компонентов представляется организация совокупности компонентов и существующие между ними зависимости. Диаграммы компонентов относятся к статическому виду системы с

точки зрения реализации. Они могут быть соотнесены с диаграммами классов, так как компонент обычно отображается на один или несколько классов, интерфейсов или кооперацией. На диаграмме развертывания отображается конфигурация обрабатывающих узлов системы и размещенных в них компонентов. Диаграммы развертывания также относятся к статическому виду архитектуры системы с точки зрения развертывания. Они связаны с диаграммами компонентов, поскольку в узле обычно размещаются один или несколько компонентов.

Тем не менее, если хочется извлечь из языка UML наибольшую пользу, лучше всего применять процесс, который имеет следующие свойства:

- управляется вариантами использования;
- сконцентрирован на архитектуре;
- является итеративным и инкрементным.

Управляемость вариантами использования означает, что прецеденты должны быть основным артефактом, на основании которого устанавливается желаемое поведение системы, проверяется и подтверждается правильность выбранной системной архитектуры, производится тестирование и осуществляется взаимодействие между участниками проекта.

Процесс называют сконцентрированным (или точнее ориентированным) на архитектуре (Architecture-centric), когда системная архитектура является решающим фактором при разработке концепций, конструировании, управлении и развитии создаваемой системы.

Итеративным (Iterative) называется процесс, который предполагает управление потоком исполняемых версий системы. Инкрементный (Incremental) процесс подразумевает постоянное развитие системной архитектуры при выпуске новых версий, причем каждая следующая версия усовершенствована в сравнении с предыдущей. Процесс, являющийся одновременно итеративным и инкрементным, называется управляемым рисками (Risk-driven), поскольку при этом в каждой новой версии серьезное внимание уделяется выявлению факторов, представляющих наибольший риск для успешного завершения проекта, и сведению их до минимума. В следующем пункте данный процесс рассматривается подробнее.

1.4 Жизненный цикл изделий, процессы жизненного цикла

Методология проектирования информационных систем описывает процесс создания и сопровождения систем в виде *жизненного цикла* (ЖЦ) ИС, представляя его как некоторую последовательность стадий и выполняемых на них процессов. Для каждого этапа определяются состав и последовательность выполняемых работ, получаемые результаты, методы и средства, необходимые для выполнения работ, роли и ответственность участников и т.д. Такое формальное описание ЖЦ ИС позволяет спланировать и организовать процесс коллективной разработки и обеспечить управление этим процессом.

Жизненный цикл ИС можно представить как ряд событий, происходящих с системой в процессе ее создания и использования.

Модель жизненного цикла отражает различные состояния системы, начиная с момента возникновения необходимости в данной ИС и заканчивая моментом ее полного выхода из употребления.

Модель жизненного цикла - структура, содержащая процессы, действия и задачи, которые осуществляются в ходе разработки, функционирования и сопровождения программного продукта в течение всей жизни системы, от определения требований до завершения ее использования.

В настоящее время известны и используются следующие *модели жизненного цикла*:

Каскадная модель (см. рис. 1.1) предусматривает последовательное выполнение всех этапов проекта в строго фиксированном порядке. Переход на следующий этап означает полное завершение работ на предыдущем этапе.

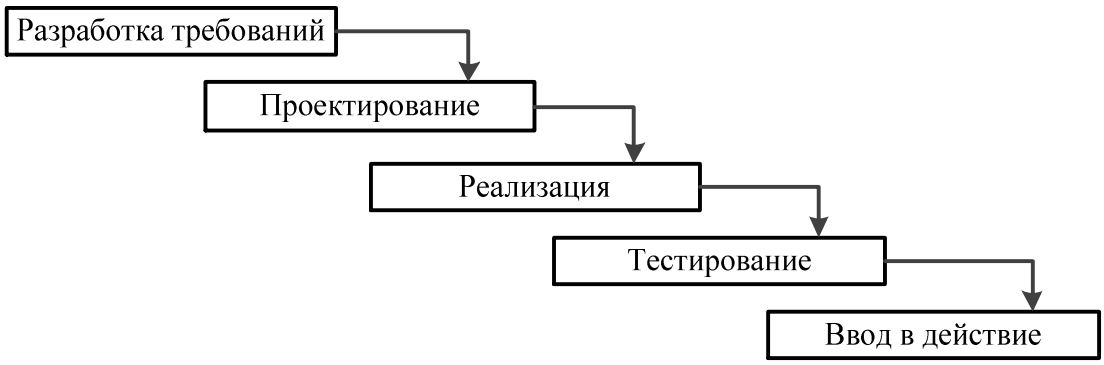

Рис. 1.2 Каскадная модель ЖЦ ИС

Поэтапная модель с промежуточным контролем (см. рис.1.2). Разработка ИС ведется итерациями с циклами обратной связи между этапами. Межэтапные корректировки позволяют учитывать реально существующее взаимовлияние результатов разработки на различных этапах; время жизни каждого из этапов растягивается на весь период разработки.

Рис. 1.3 Поэтапная модель с промежуточным контролем

Спиральная модель (см. рис. 1.3). На каждом витке спирали выполняется создание очередной версии продукта, уточняются требования проекта, определяется его качество и планируются работы следующего витка.Особое внимание уделяется начальным этапам разработки - анализу и проектированию, где реализуемость тех или иных технических решений проверяется и обосновывается посредством создания прототипов (макетирования).

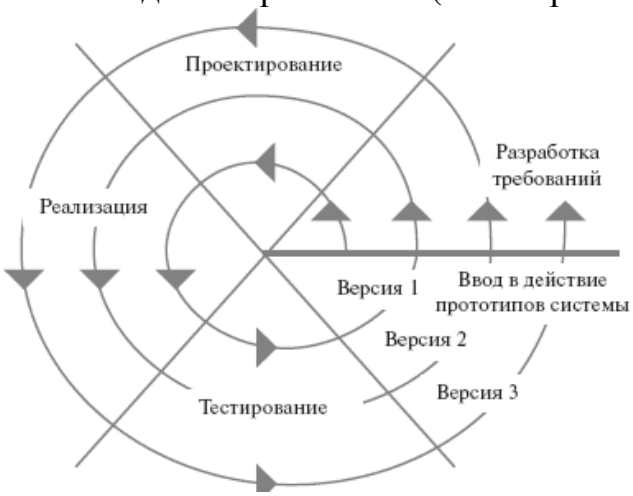

Рис. 1.4 Спиральная модель ЖЦ ИС

В ранних проектах достаточно простых ИС каждое приложение представляло собой единый, функционально и информационно независимый блок. Для разработки такого типа приложений эффективным оказался каскадный способ. Каждый этап завершался после полного выполнения и документального оформления всех предусмотренных работ.

Можно выделить следующие положительные стороны применения каскадного подхода:

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

Каскадный подход хорошо зарекомендовал себя при построении относительно простых ИС, когда в самом начале разработки можно достаточно точно и полно сформулировать все требования к системе. Основным недостатком этого подхода является то, что реальный процесс создания системы никогда полностью не укладывается в такую жесткую схему, постоянно возникает потребность в возврате к предыдущим этапам и уточнении или пересмотре ранее принятых решений. В результате реальный процесс создания ИС оказывается соответствующим *поэтапной модели с промежуточным контролем*.

Однако и эта схема не позволяет оперативно учитывать возникающие изменения и уточнения требований к системе. Согласование результатов разработки с пользователями производится только в точках, планируемых после

завершения каждого этапа работ, а общие требования к ИС зафиксированы в виде технического задания на все время ее создания. Таким образом, пользователи зачастую получают систему, не удовлетворяющую их реальным потребностям.

Спиральная модель ЖЦ была предложена для преодоления перечисленных проблем. На этапах анализа и проектирования реализуемость технических решений и степень удовлетворения потребностей заказчика проверяется путем создания прототипов.

Каждый виток спирали соответствует созданию работоспособного фрагмента или версии системы. Это позволяет уточнить требования, цели и характеристики проекта, определить качество разработки, спланировать работы следующего витка спирали. Таким образом углубляются и последовательно конкретизируются детали проекта и в результате выбирается обоснованный вариант, который удовлетворяет действительным требованиям заказчика и доводится до реализации.

Итеративная разработка отражает объективно существующий спиральный цикл создания сложных систем. Она позволяет переходить на следующий этап, не дожидаясь полного завершения работы на текущем, и решить главную задачу - 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 вводятся временные ограничения на каждый из этапов *жизненного цикла*, и переход осуществляется в соответствии с планом, даже если не вся запланированная работа закончена. Планирование производится на основе статистических данных, полученных в предыдущих проектах, и личного опыта разработчиков.

Несмотря на настойчивые рекомендации компаний - вендоров и экспертов в области проектирования и разработки ИС, многие компании продолжают использовать каскадную модель вместо какого-либо варианта итерационной модели. Основные причины, по которым каскадная модель сохраняет свою популярность, следующие:

1. Привычка - многие ИТ-специалисты получали образование в то время, когда изучалась только каскадная модель, поэтому она используется ими и в наши дни.

2. Иллюзия снижения рисков участников проекта (заказчика и исполнителя). Каскадная модель предполагает разработку законченных продуктов на каждом этапе: технического задания, технического проекта, программного продукта и пользовательской документации. Разработанная документация позволяет не только определить требования к продукту следующего этапа, но и определить обязанности сторон, объем работ и сроки, при этом окончательная оценка сроков и стоимости проекта производится на начальных этапах, после завершения обследования. Очевидно, что если требования к информационной системе меняются в ходе реализации проекта, а качество документов оказывается

невысоким (требования неполны и/или противоречивы), то в действительности использование каскадной модели создает лишь иллюзию определенности и на деле увеличивает риски, уменьшая лишь ответственность участников проекта. При формальном подходе менеджер проекта реализует только те требования, которые содержатся в спецификации, опирается на документ, а не н а реальные потребности бизнеса. Есть два основных типа контрактов на разработку ПО. Первый тип предполагает выполнение определенного объема работ за определенную сумму в определенные сроки (fixed price). Второй тип предполагает повременную оплату работы (time work). Выбор того или иного типа контракта зависит от степени определенности задачи. Каскадная модель с определенными этапами и их результатами лучше приспособлена для заключения контракта с оплатой по результатам работы, а именно этот тип контрактов позволяет получить полную оценку стоимости проекта до его завершения. Более вероятно заключение контракта с повременной оплатой на небольшую систему, с относительно небольшим весом в структуре затрат предприятия. Разработка и внедрение интегрированной информационной системы требует существенных финансовых затрат, поэтому используются контракты с фиксированной ценой, и, следовательно, каскадная модель разработки и внедрения. *Спиральная модель* чаще применяется при разработке информационной системы силами собственного отдела ИТ предприятия.

3. Проблемы внедрения при использовании итерационной модели. В некоторых областях спиральная модель не может применяться, поскольку невозможно использование/тестирование продукта, обладающего неполной функциональностью (например, военные разработки, атомная энергетика и т.д.). Поэтапное итерационное внедрение информационной системы для бизнеса возможно, но сопряжено с организационными сложностями (перенос данных, интеграция систем, изменение бизнес-процессов, учетной политики, обучение пользователей). Трудозатраты при поэтапном итерационном внедрении оказываются значительно выше, а управление проектом требует настоящего искусства. Предвидя указанные сложности, заказчики выбирают каскадную модель, чтобы "внедрять систему один раз".

1.5. Стандартизация процессов проектирования[2]

Каждая из стадий создания системы предусматривает выполнение определенного объема работ, которые представляются в виде *процессов ЖЦ*. *Процесс* определяется как совокупность взаимосвязанных действий, преобразующих входные данные в выходные. Описание каждого процесса включает в себя перечень решаемых задач, исходных данных и результатов.

[2] Практическое занятие 1. Стандарты в области инфокоммуникационных систем

Существует целый ряд стандартов, регламентирующих ЖЦ ПО, а в некоторых случаях и процессы разработки.

Значительный вклад в теорию проектирования и разработки информационных систем внесла компания IBM, предложив еще в середине 1970-х годов методологию BSP (Business System Planning - методология организационного планирования). Метод структурирования информации с использованием матриц пересечения бизнес-процессов, функциональных подразделений, функций систем обработки данных (информационных систем), информационных объектов, документов и баз данных, предложенный в BSP, используется сегодня не только в ИТ-проектах, но и проектах по реинжинирингу бизнес-процессов, изменению организационной структуры. Важнейшие шаги процесса BSP, их последовательность (получить поддержку высшего руководства, определить процессы предприятия, определить классы данных, провести интервью, обработать и организовать данные интервью) можно встретить практически во всех формальных методиках, а также в проектах, реализуемых на практике.

Среди наиболее известных стандартов можно выделить следующие:

ГОСТ 34.601-90 - распространяется на автоматизированные системы и устанавливает стадии и этапы их создания. Кроме того, в стандарте содержится описание содержания работ на каждом этапе. Стадии и этапы работы, закрепленные в стандарте, в большей степени соответствуют *каскадной модели* жизненного цикла.

ISO/IEC 12207:1995 - стандарт на процессы и организацию *жизненного цикла*. Распространяется на все виды заказного ПО. Стандарт не содержит описания фаз, стадий и этапов.

Custom Development Method (методика Oracle) по разработке прикладных информационных систем - технологический материал, детализированный до уровня заготовок проектных документов, рассчитанных на использование в проектах с применением Oracle. Применяется CDM для классической модели ЖЦ (предусмотрены все работы/задачи и этапы), а также для технологий "быстрой разработки" (Fast Track) или "облегченного подхода", рекомендуемых в случае малых проектов.

Rational Unified Process (RUP) предлагает итеративную модель разработки, включающую четыре фазы: начало, исследование, построение и внедрение. Каждая фаза может быть разбита на этапы (итерации), в результате которых выпускается версия для внутреннего или внешнего использования. Прохождение через четыре основные фазы называется циклом разработки, каждый цикл завершается генерацией версии системы. Если после этого работа над проектом не прекращается, то полученный продукт продолжает развиваться и снова минует те же фазы. Суть работы в рамках RUP - это создание и сопровождение моделей на базе UML.

Microsoft Solution Framework (MSF) сходна с RUP, так же включает четыре фазы: анализ, проектирование, разработка, стабилизация, является итерационной, предполагает использование объектно-ориентированного моделирования. MSF в сравнении с RUP в большей степени ориентирована на разработку бизнес-приложений.

Extreme Programming (XP). Экстремальное программирование (самая новая среди рассматриваемых методологий) сформировалось в 1996 году. В основе методологии командная работа, эффективная коммуникация между заказчиком и исполнителем в течение всего проекта по разработке ИС, а разработка ведется с использованием последовательно дорабатываемых прототипов.

В соответствии с базовым международным стандартом ISO/IEC 12207 все *процессы ЖЦ ПО* делятся на три группы:

1. Основные процессы:
   - приобретение;
   - поставка;
   - разработка;
   - эксплуатация;
   - сопровождение.
2. Вспомогательные процессы:
   - документирование;
   - управление конфигурацией;
   - обеспечение качества;
   - разрешение проблем;
   - аудит;
   - аттестация;
   - совместная оценка;
   - верификация.
3. Организационные процессы:
   - создание инфраструктуры;
   - управление;
   - обучение;
   - усовершенствование.

Для поддержки практического применения стандарта ISO/IEC 12207 разработан ряд технологических документов: Руководство для ISO/IEC 12207 (ISO/IEC TR 15271:1998 Information technology - Guide for ISO/IEC 12207) и Руководство по применению ISO/IEC 12207 к управлению проектами (ISO/IEC TR 16326:1999 Software engineering - Guide for the application of ISO/IEC 12207 to project management).

Позднее был разработан и в 2002 г. опубликован стандарт на процессы *жизненного цикла* систем (ISO/IEC 15288 System life cycle processes). К разработке стандарта были привлечены специалисты различных областей: системной

инженерии, программирования, управления качеством, человеческими ресурсами, безопасностью и пр. Был учтен практический опыт создания систем в правительственных, коммерческих, военных и академических организациях. Стандарт применим для широкого класса систем, но его основное предназначение - поддержка создания компьютеризированных систем.

Согласно стандарту ISO/IEC серии 15288 в структуру ЖЦ следует включать следующие группы процессов:

1. Договорные процессы:
   - приобретение (внутренние решения или решения внешнего поставщика);
   - поставка (внутренние решения или решения внешнего поставщика).
2. Процессы предприятия:
   - управление окружающей средой предприятия;
   - инвестиционное управление;
   - управление ЖЦ ИС;
   - управление ресурсами;
   - управление качеством.
3. Проектные процессы:
   - планирование проекта;
   - оценка проекта;
   - контроль проекта;
   - управление рисками;
   - управление конфигурацией;
   - управление информационными потоками;
   - принятие решений.
4. Технические процессы:
   - определение требований;
   - анализ требований;
   - разработка архитектуры;
   - внедрение;
   - интеграция;
   - верификация;
   - переход;
   - аттестация;
   - эксплуатация;
   - сопровождение;
   - утилизация.
5. Специальные процессы:
   - определение и установка взаимосвязей исходя из задач и целей.

Стадии создания системы, предусмотренные в стандарте ISO/IEC 15288, несколько отличаются от рассмотренных выше.

Перечень стадий и основные результаты, которые должны быть достигнуты к моменту их завершения следующие:

1. Формирование концепции - анализ потребностей, выбор концепции и проектных решений.
2. Разработка – проектирование системы.
3. Реализация – изготовление системы.
4. Эксплуатация – ввод в эксплуатацию и использование системы
5. Поддержка – обеспечение функционирования системы
6. Снятие с эксплуатации – прекращение использования, демонтаж, архивирование системы

2. Методология проектирования инфокоммуникационных систем[3]

2.1 Методологические подходы к проектированию инфокоммуникационных систем

Microsoft Solutions Framework (MSF)

Microsoft Solutions Framework (MSF) — это методология разработки программного обеспечения от Microsoft.

MSF опирается на практический опыт корпорации Майкрософт и описывает управление людьми и рабочими процессами в процессе разработки решения.

MSF состоит из двух моделей и трех дисциплин.

Они подробно описаны в 5 whitepapers (белых книгах).

MSF содержит:

- модели:
  - модель проектной группы
  - модель процессов
- дисциплины:
  - дисциплина управление проектами
  - дисциплина управление рисками
  - дисциплина управление подготовкой

Модель проектной группы MSF (MSF Team Model) описывает подход Майкрософт к организации работающего над проектом персонала и его деятельности в целях максимизации успешности проекта.

Данная **модель определяет** ролевые кластеры, их области компетенции и зоны ответственности, а также рекомендации членам проектной группы, позволяющие им успешно осуществить свою миссию по воплощению проекта в жизнь.

[3] Домашнее задание. Создание проекта простейшей инфокоммуникационной системы однопользовательского масштаба

Модель проектной группы MSF разрабатывалась в течение нескольких лет и возникла в результате осмысления недостатков пирамидальной, иерархической структуры традиционных проектных групп.

В соответствии с моделью MSF проектные группы строятся как небольшие многопрофильные команды, члены которых распределяют между собой ответственность и дополняют области компетенций друг друга. Это дает возможность четко сфокусировать внимание на нуждах проекта.

Проектную группу объединяет единое видение проекта, стремление к воплощению его в жизнь, высокие требования к качеству работы и желание самосовершенствоваться.

MSF включает в себя ряд основных принципов, которые имеют отношение к успешной работе команды:

- Распределение ответственности при фиксации отчетности
- Наделение членов команды полномочиями

Проект (project) – это ограниченная временными рамками деятельность, цель которой состоит в создании уникального продукта или услуги.

Управление проектами (project management) – это область знаний, навыков, инструментария и приемов, используемых для достижения целей проектов в рамках согласованных параметров качества, бюджета, сроков и прочих ограничений/

Эта модель сочетает в себе свойства двух стандартных производственных моделей: **каскадной** (waterfall) и **спиральной** (spiral).

Процесс MSF ориентирован на "**вехи**" (milestones) – ключевые точки проекта, характеризующие достижение в его рамках какого-либо существенного (промежуточного либо конечного) результата. Этот результат может быть оценен и проанализирован, что подразумевает ответы на вопросы: "Пришла ли проектная группа к однозначному пониманию целей и рамок проекта?", "В достаточной ли степени готов план действий?", "Соответствует ли продукт утвержденной спецификации?", "Удовлетворяет ли решение нужды заказчика?" и т.д.

Модель процессов MSF **учитывает постоянные изменения** проектных требований. Она исходит из того, что разработка решения должна состоять из коротких циклов, создающих поступательное движение от простейших версий решения к его окончательному виду.

Что есть решение?

В повседневном смысле решение – это просто стратегия или метод, позволяющие решить проблему. На жаргоне IT-индустрии "решениями" все чаще называют программные продукты. Поэтому время от времени возникает недопонимание или даже скептицизм в отношении того, что в действительности понимается под решением.

В MSF термин "решение" (solution) имеет очень специфическое значение. Это *скоординированная поставка* набора элементов (таких как программно-

технические средства, документация, обучение и сопровождение), необходимых для удовлетворения некоторой бизнес-потребности *конкретного заказчика*. Хотя MSF и используется при разработке коммерческих продуктов для массового потребительского рынка, он концентрируется главным образом на поставке решений, предназначенных для определенного заказчика.

Решение может включать в себя один или несколько программных продуктов, тем не менее, нужно четко разграничивать продукты и решения. Их различия суммируются в вышеприведенной таблице.

На рис. 2.1 представлены основные элементы успешного решения.

Рис. 2.1. Элементы решения

Проекты могут отличаться по уровню сложности разработки и внедрения.

Рамки проекта

Рамки (scope) – это сумма всех составляющих проекта, которые должны стать результатом работы над ним, а также все предоставляемые услуги, имеющие отношение к проекту. Рамки проекта определяют, что должно быть сделано для реализации единого видения. Они являются результатом компромисса между сформулированными целями и условиями реальности и отражают приоритезацию заказчиком имеющихся требований к создаваемому решению. Частью процесса определения рамок проекта является вынесение менее важной функциональности из текущего проекта в планы на будущее.

Четкое очерчивание рамок предоставляет возможность:

- Разбиения долговременных планов на достижимые составляющие.
- Определения функциональности, добавляемой в каждую из выпускаемых версий решения.
- Гибкости в планировании и реализации решения.

- Создания базиса для выработки компромиссов.

Необходимо определить рамки как для выполняемой работы и набора предоставляемых услуг, так и для функциональности создаваемого решения.

Термин “рамки” имеет два аспекта: рамки решения и рамки проекта. Несмотря на то, что между ними имеется тесная взаимосвязь, они не тождественны друг другу. Понимание различий между ними помогает эффективному управлению календарным графиком и стоимостью проекта.

Рамки решения (solution scope) определяют функциональность решения и его возможности (включая те, что не относятся к программному обеспечению). Возможность (функциональность, составляющая, feature) – это требуемый или желаемый аспект программного или аппаратного обеспечения. Например, предварительный просмотр перед печатью может быть возможностью текстового процессора; шифрование почтовых сообщений – возможностью почтовой программы. Сопроводительные руководства пользователей, интерактивные файлы помощи, операционные руководства и обучение также могут быть составляющими (features) решения.

Рамки проекта (project scope) определяют объем работ, который должен быть выполнен проектной группой для поставки заказчику каждого из элементов, определенного рамками решения. Некоторые организации называют рамки проекта “описанием работы” (statement of work - SOW).

Управление компромиссами

Управление рамками проекта критично для его успеха. Неудачи многих IT-проектов, включая затягивание сроков и перерасход средств, обусловлены плохой организацией управления их рамками. Эта деятельность должна включать в себя раннее определение рамок проекта, хорошо организованные мониторинг его хода и управление изменениями.

В силу свойственной IT-проектам неопределенности и рискованности, одним из ключевых факторов их успеха являются эффективные компромиссные решения (trade-offs).

Треугольник компромиссов

Хорошо известна взаимозависимость между ресурсами проекта (людскими и финансовыми), его календарным графиком (временем) и реализуемыми возможностями (рамками). Эти три переменные образуют треугольник, показанный на рис. 2.2.

После достижения равновесия в этом треугольнике изменение на любой из его сторон для поддержания баланса требует модификаций на другой (двух других) сторонах и/или на изначально измененной стороне.

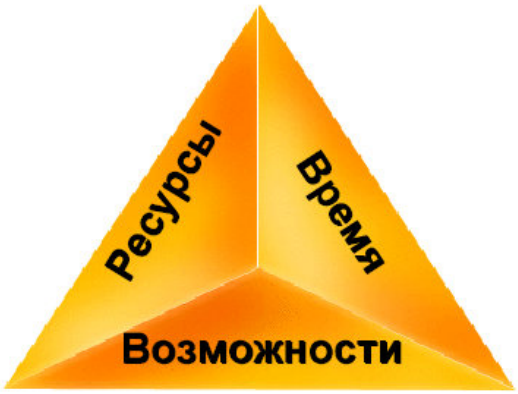

Рис. 2.2 Треугольник компромиссов

Нахождение верного баланса между ресурсами, временем разработки и возможностями – ключевой момент в построении решения, должным образом отвечающего нуждам заказчика.

Иногда заказчики не хотят, чтобы в жертву приносились определенные (не необходимые) возможности продукта. Изображенный выше треугольник помогает проиллюстрировать им суть имеющихся ограничений и служит инструментом поиска компромиссных решений.

Реализуемая функциональность должна иметь качество не ниже определенного уровня. Параметр качества может быть изображен в виде четвертого измерения, превращающего треугольник в тетраэдр (треугольную пирамиду). Хотя снижение порога качества (quality bar) дает возможность сократить затрачиваемые ресурсы/время и увеличить функциональность решения, это ведет, безусловно, к неизбежно плохому результату.

Матрица компромиссов проекта

Другое весьма полезное средство для управления проектными компромиссами – матрица компромиссов проекта (project tradeoff matrix), показанная на рис. 2.3.

Она отражает достигнутое на ранних этапах проекта соглашение между проектной группой и заказчиком о выборе приоритетов в возможных в будущем компромиссных решениях. В определенных случаях из этой приоритезации могут делаться исключения, но в целом следование ей облегчает достижение соглашений по спорным вопросам.

Рис. 2.3 показывает матрицу компромиссов проекта, используемую обычно проектными группами Майкрософт.

Она помогает обозначить проектное ограничение, воздействие на которое практически невозможно (колонка “Фиксируется”),

фактор, являющийся в проекте приоритетным (колонка “Согласовывается”), и третий параметр, значение которого должно быть принято в соответствии с установленными значениями первых двух величин (колонка “Принимается”).

Нельзя произвольно сокращать функциональность решения. Как проектная группа, так и заказчик должны тщательно проанализировать все имеющиеся в проекте ограничения и быть готовыми идти на обусловленные ими уступки.

Для иллюстрации использования матрицы компромиссов рассмотрим следующее предложение (вместо пропущенных слов могут быть вставлены “календарный график”, “ресурсы” и “функциональность”):

Зафиксировав _____, мы согласовываем _____ и принимаем результирующий _____.

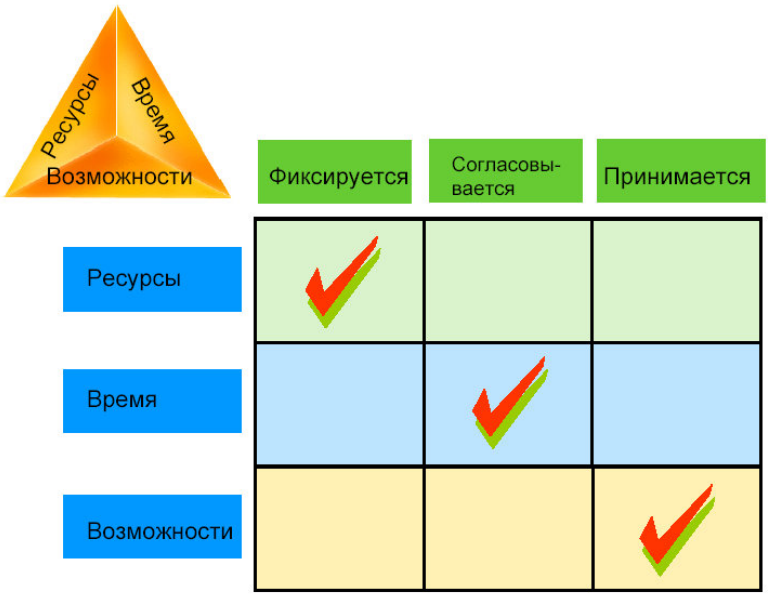

Рис. 2.3 Матрица компромиссов

Возможны такие логические взаимосвязи:

- Зафиксировав ресурсы, мы согласовываем календарный график и принимаем результирующий объем функциональности решения.
- Зафиксировав ресурсы, мы согласовываем функциональность решения и принимаем результирующие сроки.
- Зафиксировав объем функциональности решения, мы согласовываем затрачиваемые ресурсы и принимаем результирующие сроки.
- Зафиксировав объем функциональности решения, мы согласовываем календарный график и принимаем результирующие затраты ресурсов.
- Зафиксировав календарный график, мы согласовываем затраты ресурсов и принимаем результирующую функциональность решения.
- Зафиксировав сроки, мы согласовываем объем функциональности решения и принимаем результирующие затраты ресурсов.

Принципиально важно наличие у проектной группы и заказчика единого, однозначного взгляда на матрицу компромиссов проекта.

Характеристики модели процессов MSF

Тремя особенностями модели процессов MSF являются:

- Подход, основанный на фазах и вехах.
- Итеративный подход.
- Интегрированный подход к созданию и внедрению решений.

Подход, основанный на вехах

Занимая центральное место в методологии MSF, вехи используются как опорные точки для планирования и мониторинга хода проекта.

MSF вводит два типа вех: главные (major) и промежуточные (interim).

Главные вехи служат точками перехода от одной фазы к другой. Они также определяют изменения в текущих задачах ролевых кластеров.

В MSF используются обобщенные главные вехи, большинство из которых применимо к любому типу IT-проектов.

Промежуточные вехи показывают достижение в ходе проекта определенного прогресса и расчленяют большие сегменты работы на меньшие, обозримые участки.

Промежуточные вехи могут варьироваться от проекта к проекту. MSF рекомендует использовать определенный набор промежуточных вех, но на практике проектная группа может сама устанавливать их в соответствии с особенностями своей работы.

Итеративный подход

Итеративный подход к процессу разработки широко используется в MSF. Программный код, документация, дизайн, планы и другие рабочие материалы создаются, как правило, итеративными методами.

MSF рекомендует начинать разработку решения с построения, тестирования и внедрения его базовой функциональности. Затем к решению добавляются все новые и новые возможности. Такая стратегия именуется стратегией версионирования. Несмотря на то, что для малых проектов может быть достаточным выпуск одной версии, рекомендуется не упускать возможности создания для одного решения ряда версий. Рис. 2.4 показывает, как с созданием новых версий эволюционирует функциональность решения.

Версии решения не обязательно следуют одна за другой. Зрелые программные продукты обычно развиваются по нескольким направлениям параллельно. Временные интервалы между выпусками версий зависят как от размера и типа проекта, так и от нужд и стратегии заказчика.

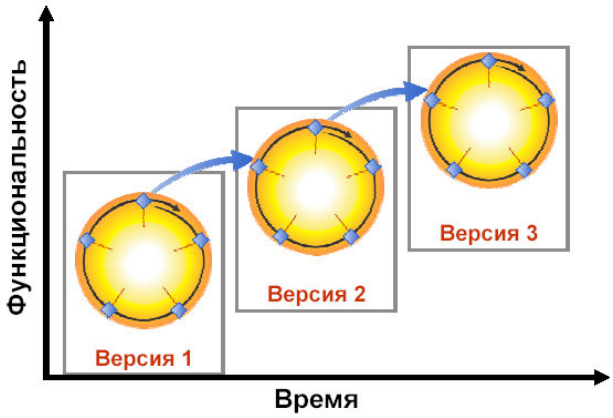

Рис. 2.4 Версионирование

Фазы и вехи модели процессов MSF

MSF версии 3.0 интегрирует в себе две ранние модели процессов: модель разработки приложений (application development - AD) и модель внедрения инфраструктуры (infrastructure deployment - ID).

Новая единая модель покрывает процесс создания решения с самого его начала и до момента окончательного внедрения.

Таким образом, использовавшаяся ранее четырехфазная схема расширена до пяти фаз. Каждая фаза заканчивается главной вехой, результаты которой становятся видимыми за пределами проектной команды. Рис. 8 изображает фазы и вехи модели процессов MSF. Хотя этот рисунок может удивить некоторых MSF-практиков, произошедшие изменения не столь значительны, как кажется. Фактически, не потерян ни один из принципиальных элементов двух исходных моделей. Все лучшее от каждой из них было соединено вместе в единый цикл. В Приложении А приводится обоснование этих, произведенных в MSF версии 3.0, изменений.

Рис. 2.5 Фазы и вехи модели процессов MSF

Точка конвергенции

В точке конвергенции (bug convergence) становится заметен существенный прогресс в устранении ошибок, то есть скорость устранения ошибок начинает

превосходить скорость их обнаружения. Рис. 2.6 иллюстрирует суть точки конвергенции.

Поскольку количество найденных, но не устраненных ошибок может колебаться даже после того, как оно начало убывать, конвергенция может рассматриваться скорее как тенденция, нежели как фиксированный момент во времени. Вслед за этой вехой количество активных ошибок должно продолжать убывать, вплоть до точки достижения нуля. Точка конвергенции дает проектной группе возможность понять, что процесс тестирования близится к концу.

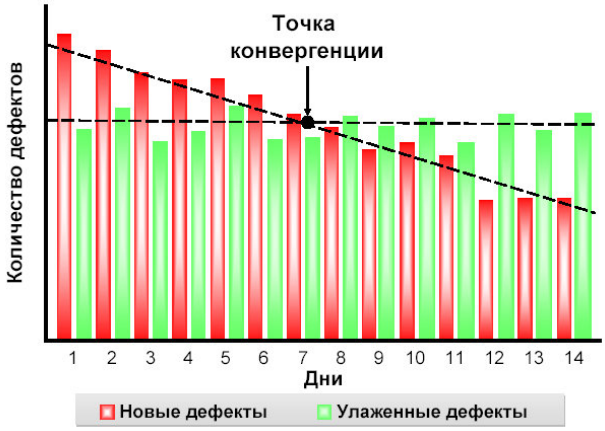

Рис. 2.6 Точка конвергенции

Точка достижения нуля

Точка достижения нуля (zero-bug bounce) – это момент, когда впервые все выявленные ошибки оказываются устраненными. Рис. 11 иллюстрирует эту точку. Вслед за ней пики количества активных ошибок должны становиться все меньше, вплоть до полного угасания в момент, когда решение уже достаточно стабильно для выпуска первой версии-кандидата.

Существенную роль играет тщательная приоритезаия ошибок, поскольку устранение всякой из них содержит риск внесения новых ошибок.

Точка достижения нуля ясно показывает, что проектная группа приближается к созданию стабильной версии-кандидата (release candidate).

Заметим, что новые ошибки после достижения этой вехи наверняка будут найдены. Однако точка достижения нуля – это первый момент в работе над проектом, когда команда может честно отчитаться об отсутствии активных ошибок и сфокусироваться на сохранении этого состояния.

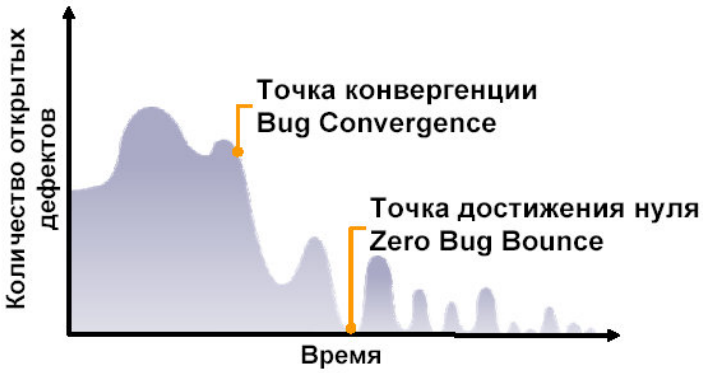

Рис. 2.7 Точка достижения нуля

MSF 5.0 для гибкой разработки программного обеспечения

Используя шаблон процесса MSF для гибкой разработки программного обеспечения версии 5.0 совместно с системой Visual Studio Application Lifecycle Management (ALM), команды могут упростить внедрение гибких методов разработки.

Scrum. Scrum — это платформа для управления разработкой сложных продуктов и систем, характеризующаяся гибкими принципами и характеристиками. Ее процессы помогают команде быстрее предоставлять клиентам дополнительные преимущества.

Рекомендации по проектированию. Для проектирования, разработки, тестирования и поставки кода команда может также использовать проверенные методы. Эти рекомендации помогают увеличить скорость, с которой команда предоставляет желаемые результаты клиентам.

Артефакты. Использование артефактов из платформы MSF для гибкой разработки программного обеспечения версии 5.0 позволяет команде затрачивать меньше усилий на применение процессов Scrum. Каждый артефакт служит для реализации определенной функции и предоставляет возможности для уточнения процессов с течением временем. Например, в книге по отставанию продукта можно описать потребности и цели клиентов.

Роли. В процессе Scrum определены три роли. Большинство участников процесса выполняют роль команды, которая отвечает за создание и доводку программного обеспечения. Кроме того, владелец продукта представляет клиентов, а координатор помогает команде и владельцу продукта соблюдать процессы Scrum.

Собрания. При использовании командой платформы Scrum будет проводиться ряд собраний. Каждое собрание имеет конкретную цель, проводится с определенной периодичностью и ограничено по времени. Например, на собрании по планированию спринта команда определяет, какие пользовательские описания функциональности она реализует в спринте. Такое собрание проводится в начале спринта и должно длиться от двух до четырех часов в зависимости от продолжительности спринта.

Scrum — это платформа для запуска проектов, основанная на гибких принципах и показателях.

Она определяет набор действий, которые помогут команде быстрее достигать поставленных заказчиками целей. Эти действия обеспечивают заказчикам возможность проверять промежуточные результаты, руководить ходом работ и влиять на них иным образом. При этом подходе не предпринимается попытка полностью определить проект в самом его начале.

Работа команды разбивается на короткие итерации (также называемые **спринтами**), и план действий уточняется по мере выполнения поставленных задач.

Платформа MSF для гибкой разработки программного обеспечения версии 5.0 основана на методологии Scrum.

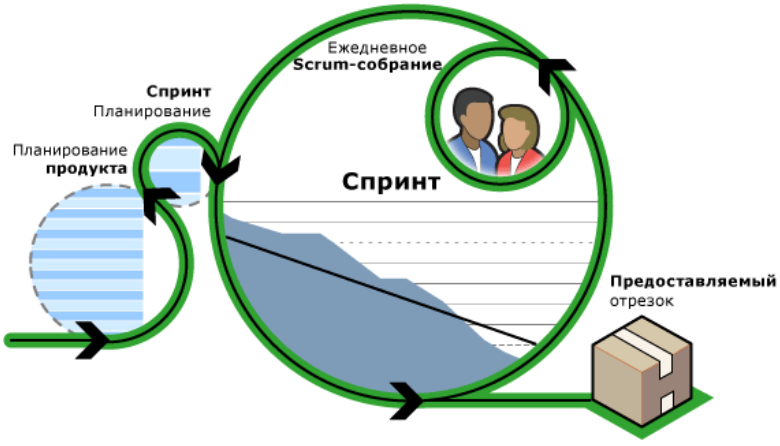

Рис. 2.8 Спринт

В Scrum итерация (цикл выпуска продукта) называется Sprint. Ее длительность имеет фиксированную длительность, обычно от двух до восьми недель, в течение которых выполняются все действия по разработке.

Результатом Sprint является готовый продукт (build), который можно потенциально передавать заказчику (по крайней мере, система должна быть готова к показу заказчику). После нескольких спринтов, обычно от 3 до 10, продукт соднржит достаточную ценность, чтобы его можно было разворачивать.

Короткие спринты обеспечивают быстрый feedback проектной команде от заказчика. Заказчик получает возможность гибко управлять scope системы, оценивая результат спринта и предлагая улучшения к созданной функциональности. Такие улучшения попадают в **Product Backlog**, приоритезируются наравне с прочими требованиями и могут быть запланированы на следующий (или на один из следующих) спринтов.

Product Backlog - это приоритезированный список имеющихся на данный момент бизнес-требований и технических требований к системе.

Каждый спринт представляет собой маленький "водопад". В течение спринта делаются все работы по сбору требований, дизайну, кодированию и тестированию продукта.

Рамки спринта должен быть фиксированным. Это позволяет команде давать обязательства на тот объем работ, который должен быть сделан в спринте. Это означает, что Sprint Backlog не может быть изменен никем, кроме команды.

Каждый спринт начинается с собрания по его планированию и заканчивается собранием, где поводятся итоги спринта.

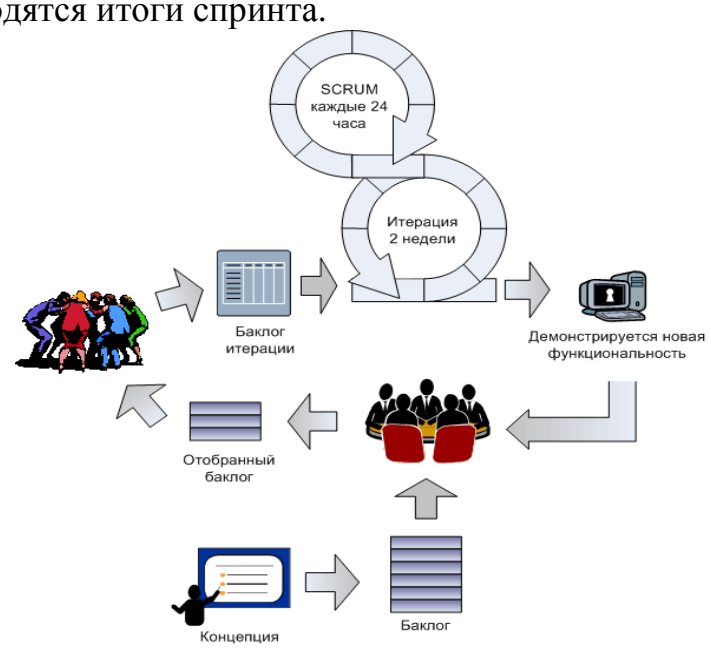

Рис. 2.9 Организация спринта

Роли

В методологии **Scrum** три роли:

- Scrum Master
- Product Owner
- Team

Скрам Мастер (Scrum Master) - координатор команды

Скрам Мастер (Scrum Master) - самая важная роль в методологии.

Мастер (Координатор) отвечает за успех использования **Scrum** в проекте. По сути, Мастер является интерфейсом между менеджментом и командой. Координатор несет ответственность за обеспечение соблюдения Scrum-процессов командой и владельцем продукта.

Как правило, эту роль в проекте играет менеджер проекта. Важно подчеркнуть, что Мастер не раздает задачи членам команды. В Agile команда является самоорганизующейся и самоуправляемой.

Основные обязанности Мастера:

- Создает атмосферу доверия.

- Участвует в митингах в качестве фасилитатора (человек, занимающийся организацией и ведением групповых форм работы с целью повышения их эффективности. Задача фасилитатора следить за регламентом и способствовать комфортной атмосфере, сплочению группы и плодотворному обсуждению).
- Устраняет препятствия.
- Делает проблемы и открытые вопросы видимыми.
- Отвечает за соблюдение практик и процесса в команде.

Роль координатора команды

Координатор команды отвечает за создание эффективной команды и организацию ее работы в соответствии со спецификой Scrum-процессов.

Он выполняет функции агента по внесению изменений, который помогает команде преодолевать возникающие сложности.

Каким должен быть координатор

Необходимо обладать отличными навыками общения, обсуждения и разрешения конфликтов. Эти навыки требуются ежедневно для обеспечения работы команды. Необходимо быть активным слушателем. Вы должны уметь не только слышать слова, но и видеть, как говорящий относится к тому, что говорит, и что именно хочет сказать (обращайте внимание на язык тела и невербальные аспекты коммуникации).Задавайте вопросы для выяснения скрытых проблем и повторяйте услышанное, перефразировав, чтобы подтвердить понимание. Активно используйте навыки слушателя, чтобы выявить потенциальные трудности в команде и предупредить их перерастание в серьезные проблемы. Устранение неполадок на ранних этапах непосредственно после обнаружения требует меньших затрат, так и в команде — любую проблему проще решить, когда она еще назревает. Ваша задача обеспечить комфорт для команды, владельца продукта и предприятия, тем самым повышая производительность. Собирайте, анализируйте и представляйте данные так, чтобы сразу были видны рост и развитие команды. Например, если команда существенно повысила добавленную стоимость и число ошибок становится все меньше и меньше, это улучшение не должно остаться незамеченным для предприятия.

Основные сферы ответственности

Эти способности и навыки используются для выполнения множества функций. Основная обязанность — обеспечение соблюдения Scrum-процессов командой и владельцем продукта. Вы обучаете команду для более эффективной реализации Scrum-процессов. Например, нельзя допускать, чтобы ежедневное Scrum-собрание перерастало в 45-минутную открытую дискуссию. Вы должны гарантировать, чтобы владелец продукта не просил команду добавить работу в уже выполняемый спринт. Не позволяйте участникам команды представлять на

собраниях по анализу спринтов недоделанные пользовательские описания функциональности.

Вы должны помогать устранять блокирующие проблемы, с которыми может столкнуться команда. Эти проблемы могут потребовать выполнения меньших задач, подобных утверждению закупки нового компьютера для построения. Проблемы могут потребовать и большего участия, такого как разрешение конфликтов между членами команды. Когда в команде возникают внутренние напряжения, вы должны способствовать возрождению командного духа.

Мастер ведет Daily Scrum Meeting и отслеживает прогресс команды при помощи Sprint Backlog, отмечая статус всех задач в спринте.

ScrumMaster может также помогать Product Owner создавать Backlog для команды.

Product Owner

Product Owner – это человек, отвечающий за разработку продукта, он определяет какие функции реализуются в продукте.

Как правило, это product manager для продуктовой разработки, менеджер проекта для внутренней разработки и представитель заказчика для заказной разработки.

Product Owner - это единая точка принятия окончательных решений для команды в проекте, именно поэтому это всегда один человек, а не группа или комитет.

Роль владельца продукта

В качестве владельца продукта ваша основная функция — действовать как интерфейс между клиентами и командой разработчиков. Это очень интенсивная роль; вы будете задействованы во множестве направлений между разными клиентами и заинтересованными лицами.

Правила успеха владельца продукта

Необходимо анализировать потребности клиентов и формулировать их в виде пользовательских описаний функциональности. У вас должны быть хорошие переговорные навыки, чтобы вы могли помочь клиентам понять компромисс между запрошенными функциями и влиянием, которое они оказывают на расписание. Следует работать с клиентами, чтобы уделять первостепенное внимание невыполненной работе по продукту. Это позволит команде создать наиболее полезный продукт или систему, постепенно наращивая функциональность на каждом этапе. Очень важно иметь опыт в той области или отрасли бизнеса, для которой создается система. Например, если команда создает систему здравоохранения для больницы, необходимы базовые знания в области здравоохранения и медицинского страхования. Без таких знаний невозможно эффективно назначить приоритет невыполненной работе по продукту и объяснить команде рабочие элементы такой невыполненной работы. Также будут полезны базовые финансовые навыки, такие как способность понимания периода

окупаемости системы, амортизации бюджетов, а также понимание бюджетирования капитала и затрат. Ваше понимание процессов Scrum и приверженность им также очень важны для успеха команды.

Основные сферы ответственности

Владелец продукта представляет клиентов и заинтересованных лиц. Его основные обязанности — представлять потребности клиентов команде, содействовать работе команды и отвечать на вопросы ее участников. Необходимо поддерживать невыполненную работу по продукту в актуальном состоянии и в порядке приоритета. Для поддержания невыполненной работы по продукту вы регулярно общаетесь с клиентами, заинтересованными лицами и командой разработчиков. С заинтересованными лицами следует встречаться как минимум каждые две недели. Порядок, в котором реализуются описания функциональности пользователей, повлияет на период окупаемости и объем работы, которую должна выполнить команда. Команда поможет вам понять, как порядок описания функциональности пользователей влияет на работу. Необходимо помочь заинтересованным лицам понять подобное влияние решений по ранжированию. Также снижается потребность в подробных спецификациях, поскольку вы доступны команде, когда у нее возникают вопросы о том, как реализовать ту или иную функциональность. Чтобы команда продвигалась, у вас должны быть ответы на эти вопросы или возможность найти их очень быстро (за несколько часов). Недоступность владельца продукта неблагоприятно повлияет на результаты работы команды.

Примечание. Уровень детализации спецификаций зависит от множества факторов, к которым относится тип приложения, разрабатываемого вашей командой. Для многих приложений самые эффективные спецификации — это тщательно продуманные описания функциональности пользователей и открытые каналы коммуникации. Однако приложение, которое обрабатывает лабораторные тесты, может требовать очень подробных спецификаций, чтобы команда смогла детально проанализировать влияние изменений, которые она вносит со временем.

Также вы будете работать как с командой, так и с заинтересованными лицами над построением приемочных тестов, что позволит команде знать, когда то или иное описание функциональности пользователя завершено и готово к спринт-рассмотрению. Необходимо понять, идентифицировать и сформулировать риск, как для команды, так и для заинтересованных лиц.

В итоге вашу функцию можно описать как обеспечение того, чтобы у вашей команды была информация, необходимая ей для удовлетворения клиентов путем построения для них наилучшего программного обеспечения.

Product Owner ставит задачи команде, но он не вправе ставить задачи конкретному члену проектной команды в течение спринта.

Команда (Team)

В методологии Scrum команда является самоорганизующейся и самоуправляемой. Команда берет на себя обязательства по выполнению объема работ на спринт перед Product Owner. Работа команды оценивается как работа единой группы. В Scrum вклад отдельных членов проектной команды не оценивается, так как это разваливает самоорганизацию команды.

Размер команды ограничивается размером группы людей, способных эффективно взаимодействовать лицом к лицу. Типичные размер команды - 7 плюс минус 2.

Команда в Scrum кроссфункциональна. В нее входят люди с различными навыками - разработчики, аналитики, тестировщики. Нет заранее определенных и поделенных ролей в команде, ограничивающих область действий членов команды. Команда состоит из инженеров, которые вносят свой вклад в общий успех проекта в соответствии со своими способностями и проектной необходимостью. Команда самоорганизуется для выполнения конкретных задач в проекте, что позволяет ей гибко реагировать на любые возможные задачи.

Для облегчения коммуникаций команда должна находиться в одном месте (colocated). Предпочтительно размещать команду не в кубиках, а в одной общей комнате для того, чтобы уменьшить препятствия для свободного общения. Команде необходимо предоставить все необходимое для комфортной работы, обеспечить досками и флипчартами, предоставить все необходимые инструменты и среду для работы.

Командная роль

Команда состоит из лиц, ответственных за создание программного обеспечения.Команда выбирает пользовательские описания функциональности в начале спринта, проводит совместную работу для реализации и тестирования пользовательских описаний функциональности в течение спринта и представляет готовое программное обеспечение при анализе спринта.Команда является самоуправляемой и управляет своей работой.Команда является универсальной в том смысле, что в нее входят специалисты с практическими навыками, необходимыми для реализации пользовательских описаний функциональности в книге "Отставание продукта".Платформа MSF для гибкой разработки программного обеспечения версии 5.0 не предусматривает разделения ролей в команде по таким признакам, как инженерная дисциплина или область специализации.

Какой должна быть команда

Собрать хорошую команду разработчиков нелегко, плюс нужно время, чтобы команда "сработалась".Примеры команд вокруг нас — футбольная команда, бригада хирургов, пастух и его собаки.У каждого участника команды может быть своя специализация, однако команда работает как единое целое и добивается успеха или терпит неудачу как единое целое.

Команды разработчиков также состоят из отдельных участников, которые вместе стремятся к достижению общей цели.Не следует рассматривать команду разработчиков как совокупность специалистов, каждый из которых поочередно выполняет те задачи, на которых специализируется.Вместо этого команда должна рассматриваться как группа людей, чьи совокупные навыки и опыт перевешивают любой отдельный навык, которым может обладать участник команды.Только благодаря сотрудничеству, общению, доверию и открытости эта группа людей сможет результативно работать и "перерасти" способности отдельных ее участников, чтобы стать высокопроизводительной командой.

Хорошая команда располагает людьми и навыками, необходимыми для выполнения обязательства по реализации концепции и поставки работоспособного программного обеспечения.Это означает, что работающие с полной занятостью участники команды должны обладать большинством навыков или всеми навыками, необходимыми для реализации проекта.Участники команды с узкой специализацией — дизайнеры, менеджеры, архитекторы или знатоки определенной технологии — могут работать в режиме частичной занятости.Команда может на кратковременной основе привлекать внешних специалистов.Однако навыки участников команды с полной занятостью должны охватывать большую часть спектра, необходимого для выполнения работ.

Основные сферы ответственности команды

Команда отвечает за поставку программного обеспечения, начиная с собрания по планированию спринта, на котором описания функциональности пользователей разбиваются на задачи.

Цикл продолжается, охватывая разработку и тестирование, до представления работающего программного обеспечения владельцу продукта (и, возможно, клиентам) на собрании по анализу спринта. Команда гарантирует, что результаты этих этапов работы удовлетворят ее собственным ожиданиям и ожиданиям ее клиентов в отношении готового программного обеспечения. Первоочередной ответственностью команды являются работы по спринту, написание и тестирование программного обеспечения.

Кроме того, команда отвечает за свою собственную эффективность. Команда самостоятельно управляет определением и исполнением выбранной ею работы, а также взаимодействием между участниками команды для оптимизации эффективности работы.

Собрания

При использовании командой платформы **Scrum** будет проводиться серия собраний, каждое с определенной целью и частотой.

Координатор должен следить, чтобы на каждом собрании была достигнута запланированная цель собрания, придерживаясь следующих рекомендаций.

- Повестка собрания должна быть четко определена.

- Если участники команды начинают обсуждать вопросы, не связанные с целью собрания, такие вопросы должны быть отложены для их решения в будущем. Координатору следует определить необходимость перевода дискуссии в автономный режим и объявить об этом участникам команды.
- На всех собраниях должна быть соблюдена основная структура проведения, описанная для каждого собрания.
- Собрания должны начинаться вовремя, даже если некоторые участники команды опаздывают.
- Участники собрания должны своевременно подключаться к собранию, за исключением редких обстоятельств, которых невозможно избежать. Если рабочий график не позволяет вам своевременно посещать собрания, этот конфликт должен быть решен в максимально приемлемые сроки. При необходимости для устранения конфликта времени координатору следует изменить время проведения собрания, при условии, что такое изменение не доставит необоснованных неудобств другим участникам команды.
- Каждый участник команды должен приходить на собрание подготовленным.
- Собрание должно заканчиваться в установленное время. В большинстве случаев продолжительность собрания определяется продолжительностью спринта. Например, если продолжительность спринта составляет неделю, на собрание по планированию спринта отводится два часа, если две недели — четыре часа.
- **Scrum** реализует данную структуру на таком уровне, который может вызывать у некоторых людей недовольство. Подобная реакция обусловлена вынуждением своевременного присутствия, индивидуальной ответственностью, связанной с принятием и выполнением обязательств, и прозрачностью, необходимой для активного участия.

Унифицированный процесс разработки

В рамках описания унифицированного процесса разработки сложных информационных систем будем считать процессом частично упорядоченное множество шагов, направленных на достижение некоторой цели. В контексте проектирования инфокоммуникационных систем целью является разработка и внедрение в предсказуемые сроки системы, удовлетворяющей потребностям бизнеса.

Выше было отмечено, что UML практически не зависит от процесса, в том смысле, что его можно использовать в различных процессах разработки. Описываемый в этом параграфе унифицированный процесс RUP (Rational Unified Process) является одним из таких подходов к организации жизненного цикла сложных систем. Отличительной особенностью именно RUP является то, что он особенно хорошо приспособлен к UML.

Цель RUP – обеспечить создание системы высокого качества, соответствующей потребностям заказчика, в заданные сроки и в пределах заранее составленной сметы.

RUP включает в себя лучшие рекомендации, которые есть в существующих методиках разработки (Best Practices).

Характеристики процесса

RUP итеративен. Если речь идет о простых системах, не представляет особого труда последовательно определить задачу, спроектировать ее целостное решение, написать программу и протестировать конечный продукт. Но, учитывая сложность и разветвленность современных систем, такой линейный подход к разработке оказывается нереалистичным. Итеративный подход предполагает постепенное проникновение в суть проблемы путем последовательных уточнений и построение все более емкого решения на протяжении нескольких циклов. Итеративному подходу присуща внутренняя гибкость, позволяющая включать в бизнес-цели новые требования или тактические изменения. Его использование оставляет возможность выявить и устранить риски, связанные с проектом, на возможно более ранних этапах разработки.

Суть работы в рамках RUP - это создание и сопровождение моделей, а не бумажных документов. Модели, особенно выраженные на языке UML, дают семантически насыщенное представление разрабатываемого программного комплекса. На них можно смотреть с разных точек зрения, а представленную в них информацию допустимо мгновенно извлечь и проконтролировать электронным способом. Рациональный Унифицированный Процесс обращен, прежде всего, на модели, а не на бумажные документы; причина состоит в том, чтобы свести к минимуму накладные расходы, связанные с созданием и сопровождением документов, и повысить степень информационного наполнения проектирования.

В центре разработки в рамках RUP лежит архитектура. Основное внимание уделяется раннему определению архитектуры программного комплекса и формулированию основных ее особенностей. Наличие прочной архитектуры позволяет "распараллелить" разработку, сводит к минимуму переделки, увеличивает вероятность того, что компоненты можно будет использовать повторно, и, в конечном счете, делает систему более удобной для последующего сопровождения. Подобный архитектурный чертеж - это фундамент, на базе которого можно планировать процесс разработки компонентного программного обеспечения и управлять им.

Разработка в рамках RUP сосредоточена на **прецедентах**. В основу построения системы положено исчерпывающее представление о том, каково ее назначение. Концепции прецедентов и сценариев используются на всех стадиях процесса, от формулирования требований до тестирования; они помогают проследить все действия от начала разработки до поставки готовой системы.

RUP поддерживает объектно-ориентированные методики. Каждая модель объектно-ориентирована. Модели, применяемые в рамках RUP, основаны на понятиях объектов и классов и отношений между ними, а в качестве общей нотации используют нотацию UML.

RUP поддается конфигурированию. Хотя ни один отдельно взятый процесс не способен удовлетворить требованиям всех организаций, занимающихся разработкой программного обеспечения, RUP поддается настройке и масштабируется для использования как в совсем небольших коллективах, так и в гигантских компаниях. Он базируется на простой и ясной архитектуре, которая обеспечивает концептуальное единство во множестве всех процессов разработки, но при этом адаптируется к различным ситуациям. Составной частью RUP являются рекомендации по его конфигурированию для нужд конкретной организации.

RUP поощряет объективный контроль качества и управление рисками на всех стадиях воплощения проекта. Контроль качества является неотъемлемой частью процесса, охватывает все виды работ и всех участников. При этом применяются объективные критерии и методы оценки. Контроль качества не рассматривается как особый род деятельности, которой можно заняться по завершении разработки. Управление рисками также встроено в процесс, так что возможные препятствия на пути успешного завершения проекта выявляются и устраняются на ранних этапах разработки, когда для реагирования еще остается время.

Фазы и итерации

Фаза (Phase) - это промежуток времени между двумя важными опорными точками процесса, в которых должны быть достигнуты четко определенные цели, подготовлены те или иные артефакты и принято решение о том, следует ли переходить к следующей фазе. Как видно из рис. 2.10, RUP состоит из следующих четырех фаз:

1. Начало (Inception) - определение бизнес-целей проекта.
2. Исследование (Elaboration) - разработка плана и архитектуры проекта.
3. Построение (Construction) - постепенное создание системы.
4. Внедрение (Transition) - поставка системы конечным пользователям.

Фазы начала и исследования охватывают проектные стадии жизненного цикла процесса разработки; фазы построения и внедрения относятся к производству.

Внутри каждой фазы происходит несколько итераций. Итерация (Iteration) представляет полный цикл разработки, от выработки требований во время анализа до реализации и тестирования. Конечным результатом является выпуск готового продукта.

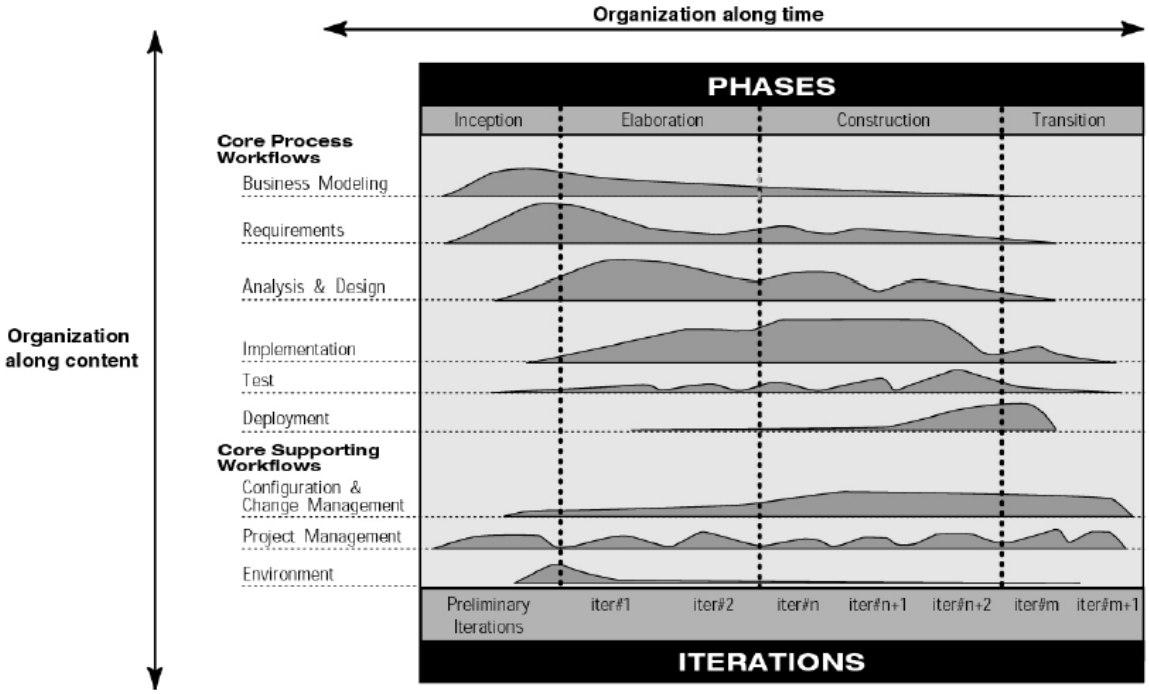

Рис.2.10 Структура RUP

Все фазы и итерации подразумевают определенные затраты усилий на снижение рисков. В конце каждой фазы находится четко определенная опорная точка, где оценивается, в какой мере достигнуты намеченные цели и не следует ли внести в процесс изменения, прежде чем двигаться дальше.

Фазы

Начало. На этой стадии определяются цели системы и устанавливаются рамки проекта. Анализ целей включает выработку критерия успешности, оценку рисков, необходимых ресурсов и составление плана, в котором отражены основные опорные точки. Нередко создается исполняемый прототип, демонстрирующий реалистичность концепции.

В конце начальной фазы еще раз подвергается внимательному изучению весь жизненный цикл проекта и принимается решение, стоит ли начинать полномасштабную разработку.

Исследование. На данном этапе стоит задача проанализировать предметную область, выработать прочные архитектурные основы, составить план проекта и устранить наиболее опасные риски. Архитектурные решения должны приниматься тогда, когда стала ясна структура системы в целом, то есть большая часть требований уже сформулирована. Для подтверждения правильности выбора архитектуры создается система, демонстрирующая выбранные принципы в действии и реализующая некоторые наиболее важные прецеденты.

В конце фазы исследования изучаются детально расписанные цели проекта, его рамки, выбор архитектуры и методы управления основными рисками, а затем принимается решение о том, надо ли приступать к построению.

Построение. В фазе построения постепенно и итеративно разрабатывается продукт, готовый к внедрению. На этом этапе описываются оставшиеся требования и критерии приемки, проект "обрастает плотью", завершается разработка и тестирование программного комплекса.

В конце фазы построения принимается решение о готовности программ, эксплуатационных площадок и пользователей к внедрению.

Внедрение. В фазе внедрения программное обеспечение передается пользователям. После этого часто возникают требующие дополнительной проработки вопросы по настройке системы, исправлению ошибок, ранее оставшихся незамеченными, и окончательному оформлению ряда функций, реализация которых была отложена. Обычно эта стадия воплощения проекта начинается с выпуска бета-версии системы, которая затем замещается коммерческой версией.

В конце фазы внедрения делается заключение о том, достигнуты ли цели проекта и надо ли начинать новый цикл разработки. Подводятся итоги работы над проектом и извлекаются уроки, которые помогут улучшить процесс разработки в ходе работы над новым проектом.

Итерации

Каждая фаза RUP может быть разбита на итерации.

Итерация - это завершенный этап, в результате которого выпускается версия (для внутреннего или внешнего использования) исполняемого продукта, реализующая часть запланированных функций. Затем эта версия от итерации к итерации наращивается до получения готовой системы. Во время каждой итерации выполняются особые рабочие процессы, хотя в разных фазах основной упор делается на разных работах. В начальной фазе главней задачей является выработка требований, в фазе исследования - анализ и проектирование, в фазе построения - реализация, а в фазе внедрения - развертывание.

Циклы разработки

Прохождение через четыре основные фазы называется циклом разработки. Каждый цикл завершается генерацией версии системы. Первый проход через все четыре фазы называют начальным циклом разработки. Если после этого работа над проектом не прекращается, то полученный продукт продолжает развиваться и снова минует те же фазы: начальную, исследования, построения и внедрения. Система эволюционирует, поэтому все циклы, следующие за начальным, называются эволюционными.

Рабочие процессы

RUP состоит из девяти рабочих процессов:

- **Business modeling** (**бизнес-анализ**) - предполагает анализ требований на данной итерации жизненного цикла, определение желаемых параметров

системы и нужд пользователей. При моделировании бизнес-процессов описывается структура и динамика организации;

- **Requirements** (**требования**) - формализация образа системы. Предполагает сбор требований и управление требованиями, перевод требований в функциональные спецификации. Здесь начинается анализ прецедентов и построение use cases (формальное отображение требований пользователя в UML). При разработке требований описывается основанный на прецедентах метод постановки задач.
- **Analysis and design** (**анализ и проектирование**) - предполагает перевод собранных требований в формализованную программную модель. Результатом является описание системы на фазе реализации (технический проект) - это документы уровня разработчиков системы. Язык формализации - UML, о котором речь пойдет ниже. В процессе итеративной разработки эволюционировать будет продукт именно этого потока - модель проекта. Все изменения привязываются в RUP непосредственно к моделям, а средства автоматизации и довольно гибкий язык моделирования позволяют управлять данным процессом более или менее безболезненно в плане затрат времени и ресурсов. При анализе и проектировании описываются различные виды архитектуры системы.
- **Implementation** (**реализация**) - предполагает собственно написание кода. Элементы кода в RUP уже созданы на этапе анализа и дизайна, так как средства реализации UML позволяют создавать элементы кода на нескольких языках программирования. При реализации выполняется собственно разработка программ, автономное тестирование и интеграция.
- **Test** (**тестирование**) - предполагает тестирование продукта на данной итерации. При тестировании описываются тестовые сценарии, процедуры и метрики для измерения числа ошибок.
- **Deployment** (**внедрение**) - предполагает установку продукта на полигоне заказчика, подготовку персонала, запуск системы плюс приемо-сдаточные испытания, подготовка стандартов упаковки и распространения продукта, передача материалов отделу продаж (действия опциональны в зависимости от специфики продукта). Развертывание охватывает конфигурирование поставляемой системы.
- **Configuration management** (**управление конфигурацией и изменениями**) - мощный слой административных действий, направленных на управление версиями продукта, что предполагает контроль исходного кода (модели, исполняемых модулей, тестов, документации), контроль версий продукта, корпоративные стандарты разработки кода и документации, отслеживание изменений и ошибок.
- **Management** (**управление проектом**) - предполагает набор административных действий управления проектом согласно идеологии RUP,

используются средства управления проектом. Управление проектом описывает разные стратегии работы с итеративным процессом.

- **Environment** (**окружение**) - предполагает создание и поддержку средств анализа, проектирования, разработки, тестирования (как программное, так и аппаратное обеспечение). При анализе среды окружения рассматриваются вопросы инфраструктуры, необходимой для разработки системы.

Приведенные выше элементы не являются новыми в плане жизненного цикла разработки ПО, поскольку имеют место практически в любой методологии.

Внутри каждого рабочего процесса сосредоточены связанные между собой артефакты и деятельности.

Артефакт (Artifact) - это некоторый документ, отчет или исполняемая программа, которые производятся, а впоследствии преобразуются или потребляются.

Термином *деятельность* (Activity) описываются задачи - обдумывание, выполнение, анализ проекта - которые решаются сотрудниками с целью создания или модификации артефактов, а также способы и рекомендации по решению этих задач. В число таких способов могут входить и инструментальные средства, позволяющие автоматизировать решение части задач.

С некоторыми из рабочих процессов ассоциированы важные связи между артефактами. Например, модель прецедентов, созданная в ходе выработки требований, конкретизируется в виде проектной модели, являющейся результатом процесса анализа и проектирования, воплощается в модели реализации, которая получена в процессе реализации, и верифицируется моделью тестирования из процесса тестирования.

2.2 CALS-технологии

CALS – (*Continuous Acquisition and Lifecycle Support* — непрерывная информационная поддержка поставок и жизненного цикла изделий) — современный подход к проектированию и производству высокотехнологичной и наукоёмкой продукции, заключающийся в использовании компьютерной техники и современных информационных технологий на всех стадиях жизненного цикла изделия.

За счет непрерывной информационной поддержки обеспечиваются единообразные способы управления процессами и взаимодействия всех участников этого цикла: заказчиков продукции, поставщиков/производителей продукции, эксплуатационного и ремонтного персонала.

Информационная поддержка реализуется в соответствии с требованиями системы международных стандартов, регламентирующих правила указанного взаимодействия преимущественно посредством электронного обмена данными.

ИПИ (информационная поддержка процессов жизненного цикла изделий) — русскоязычный аналог понятия CALS.

Применение CALS-технологий позволяет существенно сократить объёмы проектных работ, так как описания многих составных частей оборудования, машин и систем, проектировавшихся ранее, хранятся в унифицированных форматах данных сетевых серверов, доступных любому пользователю технологий CALS.

Существенно облегчается решение проблем ремонтопригодности, интеграции продукции в различного рода системы и среды, адаптации к меняющимся условиям эксплуатации, специализации проектных организаций и т. п. Предполагается, что успех на рынке сложной технической продукции будет немыслим вне технологий CALS.

Развитие CALS-технологий должно привести к появлению так называемых *виртуальных производств*, в которых процесс создания спецификаций с информацией для программно управляемого технологического оборудования, достаточной для изготовления изделия, может быть распределён во времени и пространстве между многими организационно-автономными проектными студиями. Среди несомненных достижений CALS-технологий следует отметить лёгкость распространения передовых проектных решений, возможность многократного воспроизведения частей проекта в новых разработках и др.

Построение открытых распределённых автоматизированных систем для проектирования и управления в промышленности составляет основу современных CALS-технологий. Главная проблема их построения — обеспечение единообразного описания и интерпретации данных, независимо от места и времени их получения в общей системе, имеющей масштабы вплоть до глобальных. Структура проектной, технологической и эксплуатационной документации, языки её представления должны быть стандартизированными. Тогда становится реальной успешная работа над общим проектом разных коллективов, разделённых во времени и пространстве и использующих разные CAD/CAM/CAE-системы. Одна и та же конструкторская документация может быть использована многократно в разных проектах, а одна и та же технологическая документация — адаптирована к разным производственным условиям, что позволяет существенно сократить и удешевить общий цикл проектирования и производства. Кроме того, упрощается эксплуатация систем.

Для обеспечения информационной интеграции CALS использует стандарты IGES и STEP в качестве форматов данных. В CALS входят также стандарты электронного обмена данными, электронной технической документации и руководства для усовершенствования процессов. В последние годы работа по созданию национальных CALS-стандартов проводится в России под эгидой ФСТЭК РФ. С этой целью создан Технический Комитет ТК431 «CALS-технологии», силами которого разработан ряд стандартов серии ГОСТ Р ИСО 10303, являющихся аутентичными переводами соответствующих международных стандартов (STEP).

В ряде источников данную аббревиатуру представляют, как **Computer Aided Acquisition and Logistic Support**. В 1985 году Министерство обороны США объявило планы создания глобальной автоматизированной системы электронного описания всех этапов проектирования, производства и эксплуатации продуктов военного назначения. За прошедшие годы CALS-технология получила широкое развитие в оборонной промышленности и военно-технической инфраструктуре Министерства обороны США. По имеющимся данным это позволило ускорить выполнение НИОКР на 30—40%, уменьшить затраты на закупку военной продукции на 30%, сократить сроки закупки ЗИП на 22%, а также в 9 раз сократить время на корректировку проектов.

В отличие от автоматизированной системы управления производством АСУП и от ИАСУ CALS-технологии охватывают все стадии жизненного цикла продукции.

Предмет CALS – технологии совместного использования и обмена информацией в процессах, выполняемых в течение жизненного цикла продукта.

Суть концепции CALS (ИПИ) состоит в применении принципов и технологий информационной поддержки на всех стадиях ЖЦ продукции, основанного на использовании ИИС, обеспечивающей единообразные способы управления процессами и взаимодействия всех участников этого цикла: заказчиков продукции (включая государственные учреждения и ведомства), поставщиков (производителей) продукции, эксплуатационного и ремонтного персонала. Эти принципы и технологии реализуются в соответствии с требованиями международных стандартов, регламентирующих правила управления и взаимодействия преимущественно посредством электронного обмена данными.

ИИС - основа, ядро CALS - представляет собой распределенное хранилище данных, существующее в сетевой компьютерной системе, охватывающей (в идеале) все службы и подразделения предприятия, связанные с процессами ЖЦ изделий. В ИИС действует единая система правил представления, хранения и обмена информацией. В соответствии с этими правилами в ИИС протекают информационные процессы, сопровождающие и поддерживающие ЖЦ изделия на всех его этапах. Здесь реализуется главный принцип CALS: информация, однажды возникшая на каком-либо этапе ЖЦ, сохраняется в ИИС и становится доступной всем участникам этого и других этапов (в соответствии с имеющимися у них правами пользования этой информацией). Это позволяет избежать дублирования, перекодировки и несанкционированных изменений данных, а также ошибок, связанных с этими процедурами, и сократить затраты труда, времени и финансовых ресурсов.

Основное содержание CALS, принципиально отличающее эту концепцию от других, составляют базовые принципы и технологии, которые реализуются (полностью или частично) в течение ЖЦ любого изделия, независимо от его назначения и физического воплощения.

Базовыми принципами CALS являются:

1. Безбумажный обмен информацией.

2. Анализ и реинжиниринг бизнес-процессов.

3. Параллельный инжиниринг.

4. Системная организация постпроизводственных процессов жизненного цикла изделия.

Нормативную базу применения CALS технологий составляют различные международные стандарты (например, ИСО 10303 – Система автоматизации производства и их интеграция).

Преимущества использования CALS технологий:

1. Расширяются области деятельности предприятий за счет кооперации с другими предприятиями, обеспечиваемой стандартизации предоставления информации на разных стадиях и этапах жизненного цикла.

2. Повышается эффективность бизнес-процессов.

3. Повышается конкурентоспособность продукции.

4. Сокращаются затраты и трудоемкость процессов технической подготовки и освоения производства новых изделий.

5. Сокращаются календарные сроки вывода новых видов продукции на рынок.

6. Сокращается доля брака и затрат, связанных с внесением изменений в конструкцию.

7. Сокращаются затраты на эксплуатацию, обслуживание и ремонты изделий.

Для определения организационного механизма функционирования при создании глобальной информационной индустриальной инфраструктуры организовано международное CALS-сообщество, в котором Россия принимает участие. Госстандартом России разработана программа стандартизации в области CALS-технологий.

Стандарты CALS

Воплощением CALS-идеологии являются CALS-стандарты, регламентирующие правила представления информации и информационного взаимодействия участников жизненного цикла изделия.

Стандарты CALS определяют набор правил и регламентов в соответствии с которыми строится взаимодействие участников процессов ЖЦ. Стандарты являются основным строительным блоком CALS .

Стандарты CALS покрывают весь спектр потребностей пользователей, обеспечивая единое представление текста, графики, информационных структур и данных о проекте, сопровождении и производстве, включая звук, видео, мультимедийные средства, передачу данных, хранение данных, документацию и многое другое для всех приложений. Каждый вид информации представляется через соответствующий стандарт.

Стандартизации в области CALS-технологий охватывает все процессы ЖЦ изделий: замысел (заказ), разработка (проектирование), производство (предоставление услуг), поставка (реализация), эксплуатация (техническое обслуживание), сопровождение (совершенствование, модернизация), утилизация.

Стандарты и методические материалы в области CALS-технологий в основном определяют общий подход, способ представления и интерфейсы доступа к данным различного типа, вопросы защиты информации и ее электронной авторизации (цифровой подписи).

Фундаментом CALS-технологий является система единых международных стандартов, включающая в себя:

функциональные - стандарты, определяющие описание данных об изделиях, процессы и методы формализации На сегодня в качестве функциональных стандартов в CALS рассматриваются стандарты, определяющие функциональные требования для ввода изделий в эксплуатацию и их поддержки в течение всего ЖЦ.

информационные - стандарты по описанию данных о продуктах и процессах;

технического обмена - стандарты, контролирующие носители информации и процессы обмена данными между передающими и принимающими системами.

Область действия рассматриваемых стандартов включает также информацию, необходимую для работы организаций заказчика и поставщика, а также для обмена данными между ними..

К международным стандартам представление информации о продукте относятся : ISO/IEC 10303 Standard for the Exchange of Product Model Data (STEP) и ISO 13584 Industrial Automation -- Parts Library.

К стандартам представление текстовой и графической информации относятся : ISO 8879 Information Processing -- Text and Office System - Standard Generalised Markup Language (SGML); ISO/IEC 10179 Document Style Semantics and Specification Language (DSSSL); ISO/IEC IS 10744 Information Technology -- Hypermedia/Time Based Document Structuring Language (HyTime); ISO/IEC 8632 Information Processing Systems -- Computer Graphics - Metafile; ISO/IEC 10918 Coding of Digital Continuous Tone Still Picture Images (JPEG); ISO 11172 MPEG2 Motion Picture Experts Group (MPEG); Coding of Motion Pictures and associated Audio for Digital Storage Media и ISO/IECS 13522 Information Technology -- Coding of Multimedia and Hypermedia Information (MHEG).

Третья группу стандартов общего назначения включены : ISO 11179 Information Technology -- Basic Data Element Attributes; ISO 3166 Information Processing -- Country Name Representations; ISO 31 Information Processing Representation of Quantities and Units; ISO 4217 Information Processing -- Currencies and Funds; ISO 639 Information Processing Coded Representation of Names of Languages и ISO 8601 Information Processing -- Date/Time Representations.

Основные положения и принципы CALS

Исторически по ряду объективных и субъективных причин многие подсистемы САПР и АСУ создавались как автономные системы, не ориентированные на взаимодействие с другими АС. При этом каждая из АС успешно решает определенный круг задач отдельного этапа проектирования изделий или помогает принимать решения по отдельным бизнес-процедурам этапов ЖЦИ. Но задача взаимодействия АС разных производителей и их подсистем зачастую не ставилась и не рассматривалась. Языки и форматы представления данных в разных программах не были согласованными, например, данные конструкторского проектирования не отвечали требованиям к входным данным для программ проектирования технологических процессов.

Негативные последствия несогласованности лингвистического и информационного обеспечений разных АС наиболее выпукло проявляются при росте сложности систем, в проектировании которых задействовано несколько предприятий. Показательным примером является попытка в 80-е годы создания в США системы стратегической оборонной инициативы. Стало очевидным, что без информационного взаимодействия разных АС и их подсистем эффективность автоматизации оказывается низкой, а создание многих современных сложных технических изделий – неразрешимой проблемой.

Таким образом, дальнейший прогресс в области техники и промышленных технологий оказался в зависимости от решения проблем интеграции АС путем создания единого информационного пространства управления, проектирования, производства и эксплуатации изделий. Ответом на возникшие проблемы стало создание методологии компьютерного сопровождения и информационной поддержки промышленных изделий на всех этапах их жизненного цикла. Эта методология получила название CALS.

К основным целям CALS относится прежде всего создание принципиальной возможности дальнейшего технического прогресса по пути разработки и производства усложняющихся промышленных изделий. Но CALS позволяет повысить эффективность разработки и изготовления также большинства традиционных изделий, что выражается в повышении качества, в сокращении материальных и временных затрат как на проектирование и производство, так и на эксплуатацию изделий.

Первоначально CALS создавалась как совокупность методов и средств решения логистических задач и аббревиатура CALS расшифровывалась как Computer Aided Logistics Systems. В дальнейшем сфера применения CALS расширилась и охватила все стороны информационной поддержки промышленных изделий, включая проектирование, управление предприятиями и технологическими процессами. Соответственно CALS получила новую интерпретацию и стала рассматриваться как Continuous Acquisition and Lifecycle

Support. В качестве русскоязычного эквивалента CALS принято сокращение ИПИ – информационная поддержка изделий.

Что же такое CALS в современном понимании?

Существует и используется несколько толкований.

В широком смысле слова CALS = это методология создания единого информационного пространства промышленной продукции, обеспечивающего взаимодействие всех промышленных автоматизированных систем. В этом смысле предметом CALS являются методы и средства как взаимодействия разных АС и их подсистем, так и сами АС с учетом всех видов их обеспечения. Практически синонимом CALS в этом смысле становится термин PLM (Product Lifecycle Management), широко используемый в последнее время ведущими производителями АС.

В узком смысле слова CALS – это технология интеграции различных АС со своими лингвистическим, информационным, программным, математическим, методическим, техническим и организационным видами обеспечения.

К лингвистическому обеспечению CALS относятся языки и форматы данных о промышленных изделиях и процессах, используемые для представления и обмена информацией между АС и их подсистемами на различных этапах ЖЦИ.

Информационное обеспечение составляют базы данных, включающие сведения о промышленных изделиях, используемые разными системами в процессе проектирования, производства, эксплуатации и утилизации продукции. В состав информационного обеспечения входят также серии международных и национальных CALS стандартов и спецификаций.

Программное обеспечение CALS представлено программными комплексами, предназначенными для поддержки единого информационного пространства этапов ЖЦИ. Это, прежде всего системы управления документами и документооборотом, системы PDM, средства разработки интерактивных электронных технических руководств и некоторые другие.

Математическое обеспечение CALS включает методы и алгоритмы создания и использования моделей взаимодействия различных систем в CALS-технологиях. Среди этих методов, в первую очередь, следует назвать методы имитационного моделирования сложных систем, методы планирования процессов и распределения ресурсов.

Методическое обеспечение CALS представлено методиками выполнения таких процессов, как параллельное (совмещенное) проектирование и производство, структурирование сложных объектов, их функциональное и информационное моделирование, объектно-ориентированное проектирование, создание онтологий приложений.

К техническому обеспечению CALS относят аппаратные средства получения, хранения, обработки, визуализации данных при информационном сопровождении изделий. Взаимодействие разных частей виртуальных предприятий и систем,

поддерживающих разные этапы ЖЦИ, происходит через линии передачи данных и сетевое коммутирующее оборудование. При этом широко используются возможности Internet и Web-технологий. Однако используемые технические средства не являются специфическими для CALS.

Организационное обеспечение CALS представлено различного рода документами, совокупностью соглашений и инструкций, регламентирующих роли и обязанности участников жизненного цикла промышленных изделий.

При реализации целей и задач CALS необходимо соблюдать следующие основные принципы:

• информационная поддержка всех этапов ЖЦИ;

• единство представления и интерпретации данных в процессах информационного обмена между АС и их подсистемами, что обусловливает разработку онтологий приложений и соответствующих языков представления данных;

• доступность информации для всех участников ЖЦИ в любое время и в любом месте, что обусловливает применение современных телекоммуникационных технологий;

• унификация и стандартизация средств взаимодействия АС и их подсистем;

• поддержка процедур совмещенного (параллельного) проектирования изделий.

2.3 Методология структурного анализа и проектирования систем

Схематично применение структурного подхода изображено на рис. 5.1.

Рис. 2.11 Алгоритм структурного подхода

В начале разрабатывается функциональная модель, с помощью которой определяются, анализируются и фиксируются требования к составу и структуре функций системы, т. е. определяется, для каких целей разрабатывается система, какие функции она будет выполнять. На этой же модели указываются исходная информация, промежуточные и итоговые результаты работы системы. На основе информационных потоков определяется состав и структура необходимых данных, хранимых в системе (строится информационная модель). Далее, с учетом

разработанных моделей, создаются процедуры реализации функции, т. е. алгоритмы обработки данных и поведения элементов системы. На заключительной стадии устанавливается распределение функций по подсистемам (компонентам), необходимое техническое обеспечение и строится модель их распределения по узлам системы.

Показанная на рис.2.11 схема не означает, что построение моделей должно строго соответствовать указанному порядку – одна за другой. Как правило, разработка каждой последующей модели начинается еще до полного завершения разработки предыдущей, а иногда – параллельно.

На стадии проектирования модели расширяются, уточняются и дополняются диаграммами, отражающими технологию использования системы, ее архитектуру, экранные формы и т. п.

В структурном анализе и проектировании используются различные модели, описывающие:

1. Функциональную структуру системы;
2. Последовательность выполняемых действий;
3. Передачу информации между функциональными процессами;
4. Отношения между данными.

Наиболее распространенными моделями первых трех групп являются:

- функциональная модель SADT (Structured Analysis and Design Technique);
- модель IDEF3;
- DFD (Data Flow Diagrams) - диаграммы потоков данных.

Метод SADT представляет собой совокупность правил и процедур, предназначенных для построения функциональной модели объекта какой-либо предметной области. Функциональная модель SADT отображает функциональную структуру объекта, т.е. производимые им действия и связи между этими действиями. Метод SADT разработан Дугласом Россом (SoftTech, Inc.) в 1969 г. для моделирования искусственных систем средней сложности. Данный метод успешно использовался в военных, промышленных и коммерческих организациях США для решения широкого круга задач, таких, как долгосрочное и стратегическое планирование, автоматизированное производство и проектирование, разработка ПО для оборонных систем, управление финансами и материально-техническим снабжением и др. Метод SADT поддерживается Министерством обороны США, которое было инициатором разработки семейства стандартов IDEF (Icam DEFinition), являющегося основной частью программы ICAM (интегрированная компьютеризация производства), проводимой по инициативе ВВС США. Метод SADT реализован в одном стандартов этого семейства - IDEF0, который был утвержден в качестве федерального стандарта США в 1993 г., его подробные спецификации можно найти на сайте http://www.idef.com.

Модели SADT (IDEF0) традиционно используются для моделирования организационных систем (бизнес-процессов). Следует отметить, что метод SADT успешно работает только при описании хорошо специфицированных и стандартизованных бизнес-процессов в зарубежных корпорациях, поэтому он и принят в США в качестве типового. Достоинствами применения моделей SADT для описания бизнес-процессов являются:

- полнота описания бизнес-процесса (управление, информационные и материальные потоки, обратные связи);
- жесткие требования метода, обеспечивающих получение моделей стандартного вида;
- соответствие подхода к описанию процессов стандартам ISO 9000.

В большинстве российских организаций бизнес-процессы начали формироваться и развиваться сравнительно недавно, они слабо типизированы, поэтому разумнее ориентироваться на менее жесткие модели.

Метод моделирования IDEF3, являющийся частью семейства стандартов IDEF, был разработан в конце 1980-х годов для закрытого проекта ВВС США. Этот метод предназначен для таких моделей процессов, в которых важно понять последовательность выполнения действий и взаимозависимости между ними. Хотя IDEF3 и не достиг статуса федерального стандарта США, он приобрел широкое распространение среди системных аналитиков как дополнение к методу функционального моделирования IDEF0 (модели IDEF3 могут использоваться для детализации функциональных блоков IDEF0, не имеющих диаграмм декомпозиции). Основой модели IDEF3 служит так называемый сценарий процесса, который выделяет последовательность действий и подпроцессов анализируемой системы.

Диаграммы потоков данных (Data Flow Diagrams - DFD) представляют собой иерархию функциональных процессов, связанных потоками данных. Цель такого представления - продемонстрировать, как каждый процесс преобразует свои входные данные в выходные, а также выявить отношения между этими процессами.

Для построения DFD традиционно используются две различные нотации, соответствующие методам Йордона-ДеМарко и Гейна-Сэрсона. Эти нотации незначительно отличаются друг от друга графическим изображением символов. В соответствии с данными методами модель системы определяется как иерархия диаграмм потоков данных, описывающих асинхронный процесс преобразования информации от ее ввода в систему до выдачи потребителю. Практически любой класс систем успешно моделируется при помощи DFD-ориентированных методов. Они с самого начала создавались как средство проектирования информационных систем (тогда как SADT - как средство моделирования систем вообще) и имеют более богатый набор элементов, адекватно отражающих специфику таких систем (например, хранилища данных являются прообразами файлов или баз данных,

внешние сущности отражают взаимодействие моделируемой системы с внешним миром).

С другой стороны, эти разновидности средств структурного анализа примерно одинаковы с точки зрения возможностей изобразительных средств моделирования. При этом одним из основных критериев выбора того или иного метода является степень владения им со стороны консультанта или аналитика, грамотность выражения своих мыслей на языке моделирования. В противном случае в моделях, построенных с использованием любого метода, будет невозможно разобраться.

Наиболее распространенным средством моделирования данных (предметной области) является **модель "сущность-связь"** (Entity-Relationship Model - ERM). Она была впервые введена Питером Ченом в 1976 г. Эта модель традиционно используется в структурном анализе и проектировании, однако, по существу, представляет собой подмножество объектной модели предметной области. Одна из разновидностей модели "сущность-связь" используется в методе IDEF1X, входящем в семейство стандартов IDEF и реализованном в ряде распространенных CASE-средств (в частности, AllFusion ERwin Data Modeler).

Структурный анализ является методологической разновидностью системного анализа. Концептуально в основе структурного анализа лежит выявление структуры в системе – как относительно устойчивой совокупности отношений, признания методологического примата отношений над элементами в системе.

Вспомним знаменитую фразу «Глокая куздра штеко будланула бокра и курдячит бокрёнка». Каждый ее элемент непонятен, но смысл фразы достаточно ясен – некое одушевленное существо женского рода (причем весьма неприятное) сделало какой-то агрессивный выпад (действие быстрое, сильное) в отношении существа мужского рода и в настоящий момент делает что-то нехорошее-длительное с детенышем существа мужского рода. Причем мы имеем все основания предположить, что бокр без сознания или физически не может защитить своего ребенка. Перед нами забавный и поучительный пример того, что связи в системе могут быть важнее (в определенном отношении), чем ее элементы. Синтаксис, семантика и прагматика языка, по правилам которых составлено предложение, задают структуру (связи его элементов), а также позволяют придать смысл всему предложению.

Структурным анализом называется метод исследования системы (сопровождаемый построением серии моделей), который начинается с ее общего обзора и затем детализируется, приобретая нисходящую иерархическую структуру с все большим числом уровней *. Для таких методов характерно:

- разбиение на уровни абстракции с ограничением числа элементов на каждом уровне (от трех до семи);
- ограниченный контекст, включающий лишь существенные на каждом уровне детали;

• использование строгих формальных правил записи;

• последовательное приближение к конечному результату.

Цель структурного анализа заключается в преобразовании общих, расплывчатых знаний о предметной области в точные модели, описывающие различные аспекты рассмотрения моделируемого объекта или явления. Все методологии структурного анализа базируются на ряде общих принципов, составляющих основы инженерии программного обеспечения. Многие из этих принципов являются базовыми правилами моделирования и в других областях знания.

Первым является принцип «*разделяй и властвуй*» – трудная проблема разбивается на множество меньших, независимых задач (частей), легких для понимания и решения. Части – это те самые черные ящики, о которых говорилось выше. Пользователю неважно знать, КАК работает черный ящик. Важно знать, ЧТО он делает, его функцию.

Вторым является принцип *иерархического упорядочивания*. Он декларирует, что внутреннее устройство частей также существенно для понимания и построения системы. Спускаясь по уровням иерархии, следует все более подробно описывать систему. Сколько должно быть таких уровней? Это определяется точкой зрения модели и теми вопросами, на которые должна ответить модель. Ведь в когнитивном аспекте любая модель является инструментом ответа на вопросы – и для научных исследований, и для конструирования чего-либо. Сами вопросы, если они корректно и грамотно заданы, определяют те понятия и степень подробности, в которых должна быть представлена моделируемая система.

В инженерии программного обеспечения не менее важны следующие принципы, которые определяют методологии моделирования и проектирования систем как в структурном, так и в объектно-ориентированном подходе

- Принцип абстрагирования заключается в выделении существенных с некоторых позиций аспектов системы и отвлечении от несущественных с целью представления проблемы в простом общем виде.
- Принцип формализации заключается в необходимости строгого методического подхода к решению проблемы.
- Принцип упрятывания заключается в утаивании несущественной на конкретном этапе информации, каждая часть «знает» только необходимую ей информацию.
- Принцип концептуальной целостности заключается в единообразном (даже унифицированном) представлении объекта моделирования, подчиненном общей идее. Как верно отмечает Дж. Фокс [14], концептуальную целостность лучше пояснять «от противного» – на отрицательных примерах, когда эта целостность нарушается.
- Принцип полноты заключается в контроле на присутствие лишних элементов.

- Принцип непротиворечивости заключается в обоснованности и согласованности элементов.
- Принцип логической независимости заключается в том, что в описании системы на уровне логики (строения, поведения) не должны использоваться и приниматься в расчет понятия/элементы, которые будут реализовывать эту логику на уровне физического проектирования – независимость логической модели от модели, реализующей логику.
- Принцип независимости данных – модели данных должны быть проанализированы и спроектированы независимо от процессов их логической обработки, а также от их физической структуры и распределения.
- Принцип структурирования данных – данные должны быть структурированы и иерархически организованы.
- Принцип доступа конечного пользователя – пользователь должен иметь средства доступа к данным (к БД), которые он может использовать непосредственно, без программирования.

Важнейшей характеристикой структурной методологии является порядок построения модели. Есть функционально-ориентированные и информационно-ориентированные методологии.

Методология SADT (*Structured Analysis and Design Technique*) – одна из самых известных методологий анализа и проектирования систем, введенная в 1973 г. Д. Россом (D. Ross). Она успешно использовалась в военных и коммерческих организациях для решения широкого круга задач, таких как программное обеспечение телефонных сетей, системная поддержка и диагностика, долгосрочное и стратегическое планирование, автоматизированное производство и проектирование, конфигурация компьютерных систем, управление финансами и материально-техническим снабжением и др. Данная методология широко поддерживается министерством обороны США, которое было инициатором разработки стандарта IDEF0 как подмножества SADT .

С точки зрения SADT модель может основываться либо на функциях системы, либо на ее предметах (планах, данных, оборудования, информации и т. д.). Традиционный функционально-ориентированный подход регламентирует первичность проектирования функциональных компонентов по отношению к проектированию структур данных – требования к данным раскрываются через функциональные требования. При информационно-ориентированном подходе вход и выход являются наиболее важными – структуры данных определяются первыми, а процедурные компоненты являются производными от данных.

При моделировании бизнеса, производства, управления производством исторически использовался функциональный подход. Во времена первых АСУ (автоматизированных систем управления) объектно-ориентированный подход еще не появился. В наши дни предпочтительное использование функционального

моделирования задач предприятия связано с тем, что современное предприятие рассматривается в процессной парадигме как совокупность взаимодействующих бизнес-процессов и бизнес-правил.

Неспециалисты в программировании – руководители производства, технологи, плановики и пр. – интуитивно лучше понимают процессные (функциональные) модели предприятия, поскольку руководство воспринимает деятельность предприятия в понятиях технологий, должностных инструкций и бизнес-процессов. Информационная модель предприятия, как правило, содержит несколько сотен объектов, а функциональная иерархическая модель может содержать десятки тысяч объектов. Подтверждением первичности функциональной модели является тот факт, что на Западе, где различные методики реорганизации применяются уже длительное время, большинство методологий являются функционально-ориентированными.

В методологии SADT система описывается в трех измерениях.

Уровень детализации. Описывает структуру системы в виде иерархии элементов с определением целей системы и точки зрения (аспекта, ракурса) ее рассмотрения для задач моделирования. Это может быть деление большой задачи на подзадачи, большой системы на подсистемы, большого процесса на подпроцессы. Это может быть организационная структура предприятия – иерархия его подразделений. Важно, что структура задает связи, не зависящие от времени, – статические связи между элементами. Они определяют относительную устойчивость системы, поскольку в процессе функционирования структура может меняться.

Отвечает на вопросы – из чего состоит система с данной точки зрения и какая у нее цель.

• **Уровень функционирования**. Описывает логику поведения системы. Для человеко-машинных систем, определяющих работу предприятия, для бизнес-моделирования, при описании функций используется четыре вида отношений:

1) вхождение – описывает отношение между функциями (один к одному, один ко многим, многие ко многим);

2) предметное отношение описывает связи по передаче материалов, энергии, информации, финансов и пр. между функциями. Обобщенно это можно назвать описание материальных потоков между элементами системы;

3) причинное отношение описывает причинно-следственные связи между функциями, логику их связей во времени. Обобщенно это можно назвать описание потоков управления между элементами системы;

4) приоритетное отношение – описывает приоритеты между функциями при распределении ресурсов, требуемых для исполнения функции.

Подчеркнем, что этот уровень описывает только логику функционирования – что должна делать система без привязки к реализации этой логики.

Отвечает на вопрос – ЧТО должна делать система, какие действия/функции должна выполнять.

Технологический, или технический уровень. Описывает эрготехнические средства, обеспечивающие практическую реализацию функций, логики поведения системы, описанной на втором уровне. К типам эрготехнических средств относятся:

1) аппаратное обеспечение – станки, оборудование, приборы, вычислительные машины и сети и пр.;

2) программное обеспечение – программы и программные комплексы, специализированные информационные системы управления предприятием;

3) руководящие документы – инструкции, стандарты, нормативы, приказы руководителей, технологии и пр.;

4) персонал – все работники/сотрудники, действующие в соответствии с руководящими документами.

Отвечает на вопрос – КАК, какими средствами и силами будет практически осуществляться функционирование системы.

Воплощение идей структурного анализа реализуется в конкретных технологиях и инструментах – IDEF0, DFD, IDEF3, Erwin, BPwin.

2.4 Объектно-ориентированный подход к анализу и проектированию систем[4]

Концептуальной основой объектно-ориентированного анализа и проектирования ПО (ООАП) является объектная модель. Ее основные принципы (абстрагирование, инкапсуляция, модульность и иерархия) и понятия (объект, класс, атрибут, операция, интерфейс и др.) наиболее четко сформулированы Гради Бучем.

Большинство современных методов ООАП основаны на использовании языка UML. Унифицированный язык моделирования UML (Unified Modeling Language) представляет собой язык для определения, представления, проектирования и документирования программных систем, организационно-экономических систем, технических систем и других систем различной природы. UML содержит стандартный набор диаграмм и нотаций самых разнообразных видов.

UML - это преемник того поколения методов ООАП, которые появились в конце 1980-х и начале 1990-х годов. Создание UML фактически началось в конце 1994 г., когда Гради Буч и Джеймс Рамбо начали работу по объединению их методов Booch и OMT (Object Modeling Technique) под эгидой компании Rational Software. К концу 1995 г. они создали первую спецификацию объединенного метода, названного ими Unified Method, версия 0.8. Тогда же в 1995 г. к ним присоединился создатель метода OOSE (Object-Oriented Software Engineering)

[4] Практическое занятие 3. Реализация объектно-ориентированного подхода к анализу и проектированию систем

Ивар Якобсон. Таким образом, UML является прямым объединением и унификацией методов Буча, Рамбо и Якобсона, однако дополняет их новыми возможностями. Главными в разработке UML были следующие цели:

- предоставить пользователям готовый к использованию выразительный язык визуального моделирования, позволяющий им разрабатывать осмысленные модели и обмениваться ими;
- предусмотреть механизмы расширяемости и специализации для расширения базовых концепций;
- обеспечить независимость от конкретных языков программирования и процессов разработки.
- обеспечить формальную основу для понимания этого языка моделирования (язык должен быть одновременно точным и доступным для понимания, без лишнего формализма);
- стимулировать рост рынка объектно-ориентированных инструментальных средств;
- интегрировать лучший практический опыт.

UML находится в процессе стандартизации, проводимом OMG (Object Management Group) - организацией по стандартизации в области объектно-ориентированных методов и технологий, в настоящее время принят в качестве стандартного языка моделирования и получил широкую поддержку в индустрии ПО. UML принят на вооружение практически всеми крупнейшими компаниями - производителями ПО (Microsoft, Oracle, IBM, Hewlett-Packard, Sybase и др.). Кроме того, практически все мировые производители CASE-средств, помимо IBM Rational Software, поддерживают UML в своих продуктах (Oracle Designer, Together Control Center (Borland), AllFusion Component Modeler (Computer Associates), Microsoft Visual Modeler и др.). Полное описание UML можно найти на сайтах http://www.omg.org и http://www.rational.com.

Диаграммы UML[5]

Стандарт UML версии 1.1, принятый OMG в 1997 г., содержит следующий набор диаграмм:

- Структурные (structural) модели:
  - диаграммы классов (class diagrams) - для моделирования статической структуры классов системы и связей между ними;
  - диаграммы компонентов (component diagrams) - для моделирования иерархии компонентов (подсистем) системы;
  - диаграммы размещения (deployment diagrams) - для моделирования физической архитектуры системы.
- Модели поведения (behavioral):

[5] Лабораторная работа 1. Исследование методов построения графических моделей средствами Visual Studio
Лабораторная работа 2. Создание диаграмм в Visual Studio

- диаграммы вариантов использования (use case diagrams) - для моделирования функциональных требований к системе (в виде сценариев взаимодействия пользователей с системой);
- диаграммы взаимодействия (interaction diagrams):
  - диаграммы последовательности (sequence diagrams) и кооперативные диаграммы (collaboration diagrams) - для моделирования процесса обмена сообщениями между объектами;
- диаграммы состояний (statechart diagrams) - для моделирования поведения объектов системы при переходе из одного состояния в другое;
- диаграммы деятельности (activity diagrams) - для моделирования поведения системы в рамках различных вариантов использования, или потоков управления.

Диаграммы вариантов использования показывают взаимодействия между вариантами использования и действующими лицами, отражая функциональные требования к системе с точки зрения пользователя. Цель построения диаграмм вариантов использования - это документирование функциональных требований в самом общем виде, поэтому они должны быть предельно простыми.

Вариант использования представляет собой последовательность действий (транзакций), выполняемых системой в ответ на событие, инициируемое некоторым внешним объектом (действующим лицом). Вариант использования описывает типичное взаимодействие между пользователем и системой и отражает представление о поведении системы с точки зрения пользователя. В простейшем случае вариант использования определяется в процессе обсуждения с пользователем тех функций, которые он хотел бы реализовать, или целей, которые он преследует по отношению к разрабатываемой системе.

Диаграмма вариантов использования является самым общим представлением функциональных требований к системе. Для последующего проектирования системы требуются более конкретные детали, которые описываются в документе, называемом "сценарием варианта использования" или "потоком событий" (flow of events). Сценарий подробно документирует процесс взаимодействия действующего лица с системой, реализуемого в рамках варианта использования. Основной поток событий описывает нормальный ход событий (при отсутствии ошибок). Альтернативные потоки описывают отклонения от нормального хода событий (ошибочные ситуации) и их обработку.

Достоинства модели вариантов использования заключаются в том, что она:

- определяет пользователей и границы системы;
- определяет системный интерфейс;
- удобна для общения пользователей с разработчиками;

- используется для написания тестов;
- является основой для написания пользовательской документации;
- хорошо вписывается в любые методы проектирования (как объектно-ориентированные, так и структурные).

Диаграммы взаимодействия описывают поведение взаимодействующих групп объектов (в рамках варианта использования или некоторой операции класса). Как правило, диаграмма взаимодействия охватывает поведение объектов в рамках только одного потока событий варианта использования. На такой диаграмме отображается ряд объектов и те сообщения, которыми они обмениваются между собой. Существует два вида диаграмм взаимодействия: диаграммы последовательности и кооперативные диаграммы.

Диаграммы последовательности отражают временную последовательность событий, происходящих в рамках варианта использования, а кооперативные диаграммы концентрируют внимание на связях между объектами.

Диаграмма классов определяет типы классов системы и различного рода статические связи, которые существуют между ними. На диаграммах классов изображаются также атрибуты классов, операции классов и ограничения, которые накладываются на связи между классами. Вид и интерпретация диаграммы классов существенно зависит от точки зрения (уровня абстракции): классы могут представлять сущности предметной области (в процессе анализа) или элементы программной системы (в процессах проектирования и реализации).

Диаграммы состояний определяют все возможные состояния, в которых может находиться конкретный объект, а также процесс смены состояний объекта в результате наступления некоторых событий. Диаграммы состояний не надо создавать для каждого класса, они применяются только в сложных случаях. Если объект класса может существовать в нескольких состояниях и в каждом из них ведет себя по-разному, для него может потребоваться такая диаграмма.

Диаграммы деятельности, в отличие от большинства других средств UML, заимствуют идеи из нескольких различных методов, в частности, метода моделирования состояний SDL и сетей Петри. Эти диаграммы особенно полезны в описании поведения, включающего большое количество параллельных процессов. Диаграммы деятельности являются также полезными при параллельном программировании, поскольку можно графически изобразить все ветви и определить, когда их необходимо синхронизировать.

Диаграммы деятельности можно применять для описания потоков событий в вариантах использования. С помощью текстового описания можно достаточно подробно рассказать о потоке событий, но в сложных и запутанных потоках с множеством альтернативных ветвей будет трудно понять логику событий. Диаграммы деятельности предоставляют ту же информацию, что и текстовое описание потока событий, но в наглядной графической форме.

Диаграммы компонентов моделируют физический уровень системы. На них изображаются компоненты ПО и связи между ними. На такой диаграмме обычно выделяют два типа компонентов: исполняемые компоненты и библиотеки кода.

Каждый класс модели (или подсистема) преобразуется в компонент исходного кода. Между отдельными компонентами изображают зависимости, соответствующие зависимостям на этапе компиляции или выполнения программы.

Диаграммы компонентов применяются теми участниками проекта, кто отвечает за компиляцию и сборку системы. Они нужны там, где начинается генерация кода.

Диаграмма размещения отражает физические взаимосвязи между программными и аппаратными компонентами системы. Она является хорошим средством для того, чтобы показать размещение объектов и компонентов в распределенной системе.

Диаграмма размещения показывает физическое расположение сети и местонахождение в ней различных компонентов. Ее основными элементами являются узел (вычислительный ресурс) и соединение - канал взаимодействия узлов (сеть).

Диаграмма размещения используется менеджером проекта, пользователями, архитектором системы и эксплуатационным персоналом, чтобы понять физическое размещение системы и расположение ее отдельных подсистем.

UML обладает механизмами расширения, предназначенными для того, чтобы разработчики могли адаптировать язык моделирования к своим конкретным нуждам, не меняя при этом его метамодель. Наличие механизмов расширения принципиально отличает UML от таких средств моделирования, как IDEF0, IDEF1X, IDEF3, DFD и ERM. Перечисленные языки моделирования можно определить как сильно типизированные (по аналогии с языками программирования), поскольку они не допускают произвольной интерпретации семантики элементов моделей. UML, допуская такую интерпретацию (в основном за счет стереотипов), является слабо типизированным языком. К его механизмам расширения относятся:

- стереотипы;
- тегированные (именованные) значения;
- ограничения.

Стереотип - это новый тип элемента модели, который определяется на основе уже существующего элемента. Стереотипы расширяют нотацию модели и могут применяться к любым элементам модели. Стереотипы классов - это механизм, позволяющий разделять классы на категории. Разработчики ПО могут создавать свои собственные наборы стереотипов, формируя тем самым специализированные подмножества UML (например, для описания бизнес-

процессов, Web-приложений, баз данных и т.д.). Такие подмножества (наборы стереотипов) в стандарте языка UML носят название профилей языка.

Именованное значение - это пара строк "тег = значение", или "имя = содержимое", в которых хранится дополнительная информация о каком-либо элементе системы, например, время создания, статус разработки или тестирования, время окончания работы над ним и т.п.

Ограничение - это семантическое ограничение, имеющее вид текстового выражения на естественном или формальном языке (OCL - Object Constraint Language), которое невозможно выразить с помощью нотации UML.

Сопоставление и взаимосвязь структурного и объектно-ориентированного подходов

Гради Буч сформулировал главное достоинство объектно-ориентированного подхода (ООП) следующим образом: объектно-ориентированные системы более открыты и легче поддаются внесению изменений, поскольку их конструкция базируется на устойчивых формах. Это дает возможность системе развиваться постепенно и не приводит к полной ее переработке даже в случае существенных изменений исходных требований.

Буч отметил также ряд следующих преимуществ ООП:

- объектная декомпозиция дает возможность создавать программные системы меньшего размера путем использования общих механизмов, обеспечивающих необходимую экономию выразительных средств. Использование ООП существенно повышает уровень унификации разработки и пригодность для повторного использования не только ПО, но и проектов, что в конце концов ведет к сборочному созданию ПО. Системы зачастую получаются более компактными, чем их не объектно-ориентированные эквиваленты, что означает не только уменьшение объема программного кода, но и удешевление проекта за счет использования предыдущих разработок;
- объектная декомпозиция уменьшает риск создания сложных систем ПО, так как она предполагает эволюционный путь развития системы на базе относительно небольших подсистем. Процесс интеграции системы растягивается на все время разработки, а не превращается в единовременное событие;
- объектная модель вполне естественна, поскольку в первую очередь ориентирована на человеческое восприятие мира, а не на компьютерную реализацию;
- объектная модель позволяет в полной мере использовать выразительные возможности объектных и объектно-ориентированных языков программирования.

К недостаткам ООП относятся некоторое снижение производительности функционирования ПО (которое, однако, по мере роста производительности компьютеров становится все менее заметным) и высокие начальные затраты. Объектная декомпозиция существенно отличается от функциональной, поэтому переход на новую технологию связан как с преодолением психологических

трудностей, так и дополнительными финансовыми затратами. При переходе от структурного подхода к объектному, как при всякой смене технологии, необходимо вкладывать деньги в приобретение новых инструментальных средств. Здесь следует учесть расходы на обучение методу, инструментальным средствам и языку программирования. Для некоторых организаций эти обстоятельства могут стать серьезными препятствиями.

Объектно-ориентированный подход не дает немедленной отдачи. Эффект от его применения начинает сказываться после разработки двух-трех проектов и накопления повторно используемых компонентов, отражающих типовые проектные решения в данной области. Переход организации на объектно-ориентированную технологию - это смена мировоззрения, а не просто изучение новых CASE-средств и языков программирования.

Таким образом, структурный подход по-прежнему сохраняет свою значимость и достаточно широко используется на практике. На примере языка UML хорошо видно, что его авторы заимствовали то рациональное, что можно было взять из структурного подхода: элементы функциональной декомпозиции в диаграммах вариантов использования, диаграммы состояний, диаграммы деятельности и др. Очевидно, что в конкретном проекте сложной системы невозможно обойтись только одним способом декомпозиции. Можно начать декомпозицию каким-либо одним способом, а затем, используя полученные результаты, попытаться рассмотреть систему с другой точки зрения.

Основой взаимосвязи между структурным и объектно-ориентированным подходами является общность ряда категорий и понятий обоих подходов (процесс и вариант использования, сущность и класс и др.). Эта взаимосвязь может проявляться в различных формах. Так, одним из возможных вариантов является использование структурного анализа как основы для объектно-ориентированного проектирования. При этом структурный анализ следует прекращать, как только структурные модели начнут отражать не только деятельность организации (бизнес-процессы), а и систему ПО. После выполнения структурного анализа можно различными способами приступить к определению классов и объектов. Так, если взять какую-либо отдельную диаграмму потоков данных, то кандидатами в классы могут быть элементы структур данных.

Другой формой проявления взаимосвязи можно считать интеграцию объектной и реляционной технологий. Реляционные СУБД являются на сегодняшний день основным средством реализации крупномасштабных баз данных и хранилищ данных. Причины этого достаточно очевидны: реляционная технология используется достаточно долго, освоена огромным количеством пользователей и разработчиков, стала промышленным стандартом, в нее вложены значительные средства и создано множество корпоративных БД в самых различных отраслях, реляционная модель проста и имеет строгое математическое основание; существует большое разнообразие промышленных средств

проектирования, реализации и эксплуатации реляционных БД. Вследствие этого реляционные БД в основном используются для хранения и поиска объектов в так называемых объектно-реляционных системах.

Взаимосвязь между структурным и объектно-ориентированным подходами достаточно четко просматривается в различных ТС ПО.

Визуальное моделирование как способ познания

В процессе моделирования естественно разработчик упрощает реальность, чтобы лучше понять проектируемую систему. С помощью UML, вы можете строить модели из базовых блоков, таких как классы, интерфейсы, кооперации, компоненты, узлы, зависимости, обобщения и ассоциации. Диаграммы позволяют обозревать эти строительные блоки в удобной для понимания форме.

Диаграмма - это графическое представление совокупности элементов, чаще всего изображаемое в виде связного графа, состоящего из вершин (сущностей) и ребер (отношений). С помощью диаграмм можно визуализировать систему с различных точек зрения. Поскольку сложное целое нельзя понять, глядя на него лишь с одной стороны, в UML определено много разных диаграмм, которые позволяют сосредоточиться на различных аспектах моделируемой системы.

Хорошие диаграммы облегчают понимание модели. Продуманный выбор диаграмм при моделировании системы позволяет задавать правильные вопросы о ней, помогает грамотной постановке задачи и проясняет последствия принимаемых решений.

При моделировании системы с различных точек зрения вы фактически конструируете ее сразу в нескольких измерениях. Правильный выбор совокупности видов или представлений позволит задать нужные вопросы, касающиеся системы, и выявить риск, который необходимо учесть. Если же виды выбраны плохо или вы сосредоточились только на одном из них в ущерб остальным, то возрастает опасность не заметить или отложить на будущее такие вопросы, пренебрежение которыми рано или поздно поставит под угрозу весь проект.

Для моделирования системы с использованием различных представлений рекомендуется выполнить следующее:

1. Решите, какие именно виды лучше всего отражают архитектуру систем и возможный технический риск, связанный с проектом. При этом стоит начать с описанных выше пяти взглядов на архитектуру.
2. В отношении каждого из выбранных видов определите, какие артефакты необходимо создать для отражения его наиболее существенных деталей. Эти артефакты по большей части будут состоять из различных диаграмм UML.
3. В ходе планирования процесса решите, какие из диаграмм удобнее все превратить в инструмент контроля (формального или неформального) разработкой системы. Эти диаграммы вы будете периодически корректировать и сохранять в составе проектной документации.

4. На всякий случай сохраните место для диаграмм, которые пока не задействованы.

Допустим, если вы моделируете клиентское приложение, выполняемое на одном компьютере, могут потребоваться только нижеперечисленные диаграммы:

- вид с точки зрения вариантов использования - диаграммы прецедентов;
- вид с точки зрения проектирования - диаграммы классов (для структурного моделирования) и диаграммы взаимодействия (для моделирования поведения);
- вид с точки зрения процессов - не требуется;
- вид с точки зрения реализации - не требуется; вид с точки зрения развертывания - также не требуется.

Если же разрабатываемая система относится к управлению рабочим процессом, то для моделирования ее поведения понадобятся соответственно диаграммы состояний и действий.

Если система построена на архитектуре "клиент/сервер", например, автоматизирующее работу инфокоммуникационной системы учебного центра то стоит включать в работу диаграммы компонентов и развертывания для моделирования конкретных физических деталей реализации.

Наконец, моделируя сложную распределенную систему, используйте все имеющиеся в UML диаграммы. Они позволят выразить ее архитектуру и связанный с проектом технический риск. Вам потребуется следующее:

- вид с точки зрения прецедентов - диаграммы прецедентов и диаграммы действий (для моделирования поведения);
- вид с точки зрения проектирования - диаграммы классов (структурное моделирование), диаграммы взаимодействия (моделирование поведения), диаграммы состояния (моделирование поведения);
- вид с точки зрения процессов - снова диаграммы классов (структурное моделирование) и диаграммы взаимодействия (моделирование поведения);
- вид с точки зрения реализации - диаграммы компонентов;
- вид с точки зрения развертывания - диаграммы развертывания.

Диаграмма Use Case (Вариантов использования)

Функциональные требования к системе удобно определять и документировать с помощью модели прецедентов (вариантов использования). Данная модель показывает функции системы (собственно варианты использования), их окружение (актеры) и связи (отношения) между прецедентами и актерами.

Для общего представления функциональности моделируемой системы предназначены диаграммы вариантов использования, которые на концептуальном уровне описывают поведение системы в целом.

Диаграмма вариантов использования является исходным концептуальным представлением или концептуальной моделью системы в процессе ее проектирования и разработки.

Разработка диаграммы вариантов использования преследует следующие цели:

- Определить общие границы и контекст моделируемой предметной области на начальных этапах проектирования системы.
- Сформулировать общие требования к функциональному поведению проектируемой системы.
- Разработать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей.
- Подготовить исходную документацию для взаимодействия разработчиков системы с ее заказчиками и пользователями.

Прецеденты и актеры

Прецедент (Use case) (часто применяют термин вариант использования) специфицирует поведение системы или ее части и представляет собой описание множества последовательностей действий (именно последовательности, сведение к атомарному действию является ошибкой при определении прецедента), выполняемых системой для того, чтобы субъект (актер) мог получить определенный результат.

С помощью прецедентов можно описать поведение разрабатываемой системы, не определяя ее реализацию. Таким образом, они позволяют достичь взаимопонимания между разработчиками, экспертами и конечными пользователями. Кроме того, прецеденты помогают проверить архитектуру системы в процессе ее разработки. Реализуются они кооперациями.

Хорошо структурированные прецеденты описывают только существенное поведение системы или подсистемы и не являются не слишком общими, не слишком специфическими.

Важнейшая особенность разработки прецедентов состоит в том, что вы не специфицируете конкретный способ их реализации. Например, поведение банкомата можно описать с помощью вариантов взаимодействия с ним пользователей, но вам не обязательно знать, как он устроен. Прецеденты специфицируют желаемое поведение, но ничего не говорят о том, как его достичь. И, что очень важно, это позволяет вам как эксперту или конечному пользователю общаться с разработчиками, конструирующими систему в соответствии с вашими требованиями, не углубляясь в детали реализации. Подробности будут рассмотрены потом, а на данном этапе вы можете сконцентрироваться на наиболее существенных проблемах.

В UML поведение моделируется посредством прецедентов, специфируемых независимо от реализации.

Формулировка понятия прецедента как описание множества последовательностей действий, которые выполняются системой для того, чтобы

актер получил результат, имеющий для него определенное значение, включает в себя несколько важных моментов.

Первый момент. Прецедент описывает множество последовательностей, каждая из которых представляет взаимодействие сущностей, находящихся вне системы (ее актеров), с системой как таковой и ее ключевыми абстракциями.

Такие взаимодействия являются в действительности функциями уровня системы, которыми вы пользуетесь для визуализации, специфицирования, конструирования и документирования ее желаемого поведения на этапах сбора и анализа требований. Прецедент представляет функциональные требования к системе в целом. Например, основным прецедентом в работе банка является обработка займов.

Второй момент. Прецеденты предполагают взаимодействие актеров и системы.

Актер представляет собой логически связанное множество ролей, которые играют пользователи прецедентов во время взаимодействия с ними. Актерами могут быть как люди, так и автоматизированные системы. Например, при моделировании работы банка процесс обработки займов включает в себя, помимо всего прочего, взаимодействие между клиентом и сотрудником кредитного отдела.

Третий момент. Варианты использования могут иметь разновидности.

В любой хорошо продуманной системе существуют прецеденты, которые либо являются специализированными версиями других, более общих, либо входят в состав прочих прецедентов, либо расширяют их поведение. Общее поведение множества прецедентов, допускающее повторное применение, можно выделить, организуя их в соответствии с тремя описанными видами отношений. В частности, при моделировании работы банка базовый прецедент, описывающий обработку займов, подразделяется на несколько вариаций, от оформления крупной закладной до выдачи маленькой деловой ссуды. Однако все эти прецеденты имеют нечто общее в особенностях поведения, например оценку платежеспособности клиента.

Четвертый момент. Всякий прецедент должен выполнять некоторый объем работы.

С точки зрения данного актера, прецедент делает нечто представляющее для него определенную ценность, например, вычисляет результат, создает новый объект или изменяет состояние другого объекта. В примере с работой банка процесс обработки заявки на ссуду приводит к подписанию расходного документа и материализуется в виде некоторой суммы денег, вручаемой клиенту.

Пятый момент. Прецеденты могут быть применены ко всей системе или к ее части, в том числе к подсистемам или даже к отдельным классам и интерфейсам.

В любом случае прецеденты не только представляют желаемое поведение этих элементов, но могут быть использованы как основа для их тестирования на различных этапах разработки. Прецеденты в применении к подсистемам - это

прекрасный источник регрессионных тестов, а в применении к системе в целом - источник комплексных и системных тестов.

Графическое изображение прецедента было показано на рис. 2.2. Актер изображается человечком из палочек, но в некоторых инструментальных средствах рисуется почти настоящий человек. Например, средства визуализации UML диаграмм Visual Studio 2010 Ultimate показывают как на рис. 2.10.

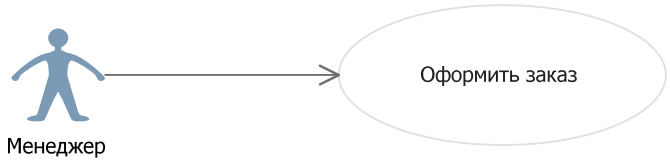

Рис. 2.12 Актер и прецедент

Любой прецедент должен иметь имя, отличающее его от других прецедентов. Оно должно быть уникально внутри объемлющего пакета. Имя прецедента представляет собой текстовую строку. Взятое само по себе, оно называется простым именем. К составному имени спереди добавлено имя пакета, в котором он находится. Обычно при изображении прецедента указывают только его имя, как показано на рис. 2.12.

Имя прецедента может состоять из любого числа букв, цифр и некоторых знаков препинания (за исключением таких, как двоеточия, которые применяются для отделения имени прецедента от имени объемлющего пакета). Имя может занимать несколько строк. На практике для именования прецедентов используют короткие глагольные фразы в активной форме, обозначающие некоторое поведение и взятые из словаря моделируемой системы.

Актер представляет собой связное множество ролей, которые пользователи прецедентов исполняют во время взаимодействия с ними. Обычно актер представляет роль, которую в данной системе играет человек, аппаратное устройство или даже другая система. Например, если вы работаете в банке, то можете играть роль *Сотрудник кредитного отдела*. Если в этом банке у вас имеется счет, вы играете роль *Клиента*. Таким образом, экземпляр актера представляет собой конкретную личность, взаимодействующую с системой определенным образом. Хотя вы и используете актеров в своих моделях, они не являются частью системы, так как существуют вне ее.

Как показано на рис. 2.13, актеров изображают в виде человеческих фигурок. Можно определить общие типы актеров (например, *Клиент*) и затем специализировать их (например, создав разновидность *Коммерческий Клиент*) с помощью отношений обобщения.

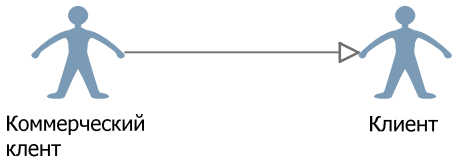

Рис. 2.13 Актеры

Актеров можно связывать с прецедентами только отношениями ассоциации (как на рис. 2.12). Ассоциация между актером и прецедентом показывает, что они общаются друг с другом, возможно, посылая или принимая сообщения.

Прецеденты и поток событий

Прецедент описывает, что делает система (подсистема, класс или интерфейс), но не определяет, каким образом она это делает. В процессе моделирования всегда важно разделять внешнее и внутреннее представления.

Можно специфицировать поведение прецедента путем описания потока событий в текстовой форме - в виде, понятном для постороннего читателя. В описание необходимо включить указание на то, как и когда прецедент начинается и заканчивается, когда он взаимодействует с актерами и какими объектами они обмениваются.

Важно обозначить также основной и альтернативный потоки поведения системы. Например, в контексте банкомата можно было бы следующим образом описать прецедент **ValidateUser** (Проверить Пользователя):

- Основной поток событий: прецедент начинается, когда система запрашивает у клиента его персональный идентификационный номер (PIN). Клиент (**Customer**) может ввести его с клавиатуры. Завершается ввод нажатием клавиши Enter. После этого система проверяет введенный PIN и, если он правильный, подтверждает ввод. На этом прецедент заканчивается.
- Исключительный поток событий: клиент может прекратить транзакцию в любой момент, нажав клавишу Cancel. Это действие начинает прецедент заново. Никаких изменений на счету клиента не производится.
- Исключительный поток событий: клиент может в любой момент до нажатия клавиши Enter стереть свой PIN и ввести новый.
- Исключительный поток событий: если клиент ввел неправильный PIN, прецедент запускается сначала. Если это происходит три раза подряд, система отменяет всю транзакцию и не позволяет данному клиенту снова начать работу с банкоматом в течение 60 секунд.

Прецеденты и сценарии

Как правило, в начале работы потоки событий прецедента описывают в текстовой форме. По мере уточнения требований к системе будет удобнее перейти к графическому изображению потоков на диаграммах взаимодействия (о них

будет рассказано ниже). Обычно для описания основного потока прецедента используют диаграмму последовательности, а для дополнительных – ее вариации.

Желательно отделять главный поток от альтернативных, поскольку прецедент описывает не одну, а множество последовательностей, и выразить все детали интересующего вас прецедента с помощью одной последовательности невозможно. Например, в системе управления человеческими ресурсами присутствует прецедент Hire employee (Нанять работника). У этой общей бизнес-функции существует множество вариаций. Вы можете переманить сотрудника из другой компании (так бывает чаще всего), перевести человека из одного подразделения в другое (эта практика распространена в транснациональных компаниях) или нанять иностранца (особый случай, регулируемый специальными правилами). Каждый вариант описывается своей последовательностью.

Таким образом, один прецедент Hire employee описывает несколько последовательностей, или сценариев, каждый из которых представляет одну из возможных вариаций данного потока событий.

Сценарий (Scenario) - это некоторая последовательность действий, иллюстрирующая поведение системы. Сценарии находятся в таком же отношении к прецедентам, как экземпляры к классам, то есть сценарий - это экземпляр прецедента.

Выбор актеров

Актеры не являются частью системы, они представляют что угодно, или кого угодно, что взаимодействует с системой. Правильный актер может только вводить в систему информацию и получать ее от системы. Обычно актеры выявляются на основе изучения предметной области по результатам общения с заказчиком и экспертами. Для определения актеров в системе рекомендуется использовать следующе вопросы:

- Кто заинтересован в данных требованиях?
- Где будет применяться данная система?
- Кто выигрывает от использования системы?
- Кто обеспечивает систему информацией, применяет и удаляет её?
- Кто занимается поддержкой системы?
- Использует ли система внешние ресурсы?
- Выполняет ли один человек несколько ролей?
- Взаимодействует ли система с другими системами?

К процессу определения актеров рекомендуется подходить итеративно, и первый вариант списка актеров редко совпадает с конечным. Можно начинать с создания актеров для каждой роли, которые может играть человек. Например, в системе регистрации курсов обучения определены актеры «Студент» и «Профессор», но также выявлена роль ассистент преподавателя. Возникает вопрос: может быть ассистент актером или нет. Из анализа предметной области

выяснилось, что ассистент преподавателя может посещать одни курсы и преподавать на других. Необходимая ему возможность выбирать курсы для посещения повторяет функциональность, заложенную в актере «Студент», а возможность для преподавания – в актере «Профессор». Получается, что необходимость в актере для ассистента отпадает. Таким образом, выявляя идентичных актеров и документируя их взаимодействие с системой, можно постепенно получить оптимальный набор актеров для системы.

Выбор прецедентов

Выше было отмечено, что набор прецедентов должен охватывать все способы употребления системы. Для определения прецедентов в системе рекомендуется ответить на следующие вопросы:

- Какую задачу выполняет каждый из актеров?
- Будет ли кто-то из актеров создавать, сохранять, изменять, удалять информацию из системы?
- Где будет создаваться, сохраняться, изменяться, удаляться информация?
- Потребуется ли актеру уведомлять систему о внешних изменениях?
- Необходимо ли оповещать актеров о наступлении каких-либо событий в системе?
- Могут ли все функциональные требования быть выполнены с помощью прецедентов?

Одной из проблем при создании набора прецедентов является определение уровня детализации прецедентов. Универсального правила для выбора степени обобщенности или конкретизации не существует, но на практике можно использовать простое правило: прецедент обычно представляет большой кусок функциональности от начала и до конца, а также он должен предоставлять актеру явную пользу. Например, в результате анализа функционирования системы регистрации курсов установлено, что студент должен выбирать курсы на семестр, студенты должны быть добавлены в список обучаемых по выбранному ими курсу и студентам должны быть выписаны счета для оплаты обучения. Возникает естественный вопрос: это три прецедента или один общий? Очевидно, что выгода для центра обучения будет тогда, когда все три действия будут выполнены. Какой будет толк от системы, если студент не будет зачислен на курс обучения или ему не будет выписан счет на оплату? В результате все действия следует поместить в один прецедент.

Еще одной проблемой является объединение различной функциональности, когда она кажется связанной. Например, в той же системе обучения актер «Секретарь» может добавлять курсы, удалять их и изменять. Опять вопрос: это три прецедента или один? В данном случае есть смысл объединить эти действия в один прецедент, и назвать его, к примеру, «Поддержка учебного плана». Это

можно объяснить тем, что описанная функциональность завязана на одного актера и работает он с одной сущностью в системе (учебный план).

Одной из наиболее распространенных ошибок при создании прецедентов является попадание в ловушку функциональной декомпозиции. Чтобы избежать этого рекомендуется при трактовке прецедентов придерживаться следующих рекомендаций:

- Определите, представляет ли прецедент некую законченную функциональность, которую инициирует один актер.
- Избегайте мелких прецедентов, представляющих кусочки функциональности.
- Избегайте множества прецедентов, редко для реальных систем может быть их больше полусотни.

Наиболее распространенным примером реализации диаграммы прецедентов является моделирование банковского автомата:

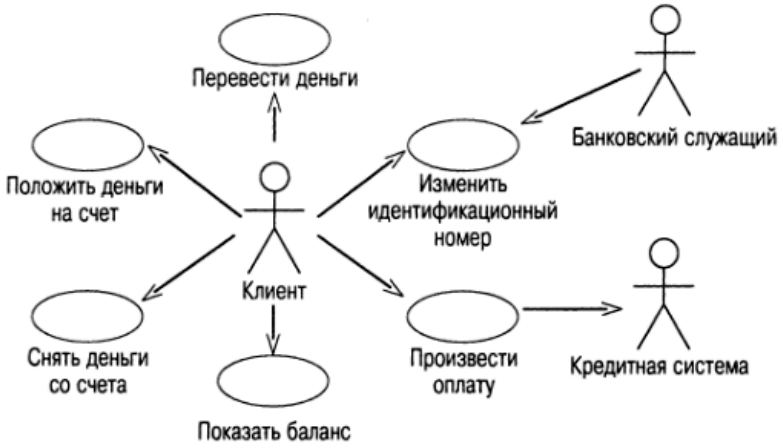

Рис. 2.14 Диаграмма прецедентов банкомата

Диаграммы взаимодействия

В хорошо спроектированной системе объекты не являются обособленными, они взаимодействуют друг с другом, постоянно обмениваясь сообщениями.

Взаимодействием (Interactions) называется поведение, которое предполагает обмен сообщениями между объектами данной совокупности в данном контексте, в результате чего достигается определенная цель.

Взаимодействия применяют для моделирования динамических аспектов кооперации (Collaborations), которые представляют собой сообщества играющих определенные роли объектов, совместное поведение которых более значимо, чем сумма его элементов. Эти роли представляют экземпляры классов, интерфейсов, компонентов, узлов и прецедентов, динамические аспекты которых визуализируются, специфицируются, конструируются и документируются в виде потоков управления. Эти потоки управления могут быть представлены в системе

простыми последовательными потоками или более сложными, включающие ветвления, циклы, рекурсию и параллелизм.

Каждое взаимодействие можно моделировать двумя способами: акцентируя внимание на временной упорядоченности сообщений или на их последовательности в контексте некоей структурной организации объектов.

Хорошо структурированные взаимодействия похожи на хорошо структурированные алгоритмы, они эффективны, просты, понятны и пригодны для адаптации к различным задачам.

Диаграммы последовательностей и кооперации (и те, и другие называются диаграммами взаимодействий) относятся к числу пяти видов диаграмм, применяемых в UML для моделирования динамических аспектов системы. На диаграммах взаимодействий показывают связи, включающие множество объектов и отношений между ними, в том числе сообщения, которыми объекты обмениваются. При этом диаграмма последовательностей акцентирует внимание на временной упорядоченности сообщений, а диаграмма кооперации - на структурной организации посылающих и принимающих сообщения объектов.

Диаграммы взаимодействий могут существовать автономно и служить для визуализации, специфицирования, конструирования и документирования динамики конкретного сообщества объектов, а могут использоваться для моделирования отдельного потока управления в составе прецедента.

Диаграммы взаимодействий важны не только для моделирования динамических аспектов системы, но и для создания исполняемых систем посредством прямого и обратного проектирования.

Диаграммой последовательностей (Sequence diagram) называется диаграмма взаимодействий, акцентирующая внимание на временной упорядоченности сообщений. Графически такая диаграмма представляет собой таблицу, *объекты* в которой располагаются вдоль оси X, а *сообщения* в порядке возрастания времени - вдоль оси Y.

Диаграммы последовательности отражают поток событий, происходящих в рамках варианта использования и демонстрирует взаимодействие объектов, упорядоченное по времени.

Объект – это представление сущности либо из реального мира, либо концептуальной. Объект может представлять нечто конкретное, например, грузовик или компьютер, или концепцию, например химический процесс, банковскую транзакцию, чек на покупку.

Класс описывает группу объектов с общими свойствами (атрибутами), общим поведением (операциями), общими связями с другими объектами и общей семантикой (шаблон для создания объекта).

На диаграмме последовательности:

- изображаются исключительно те объекты, которые непосредственно участвуют во взаимодействии,

- не показываются возможные статические ассоциации с другими объектами.

Организация отображения объектов

Для диаграммы последовательности ключевым моментом является именно динамика взаимодействия объектов во времени.

При этом диаграмма последовательности имеет как бы два измерения. Одно - *слева направо* в виде вертикальных линий, каждая из которых изображает линию жизни отдельного объекта, участвующего во взаимодействии.

Графически каждый объект изображается прямоугольником и располагается в верхней части своей линии жизни. Внутри прямоугольника записываются имя объекта и имя класса, разделенные двоеточием. При этом вся запись подчеркивается, что является признаком объекта, который, как известно, представляет собой экземпляр класса.

Крайним слева на диаграмме изображается объект, который является инициатором взаимодействия (объект 1).

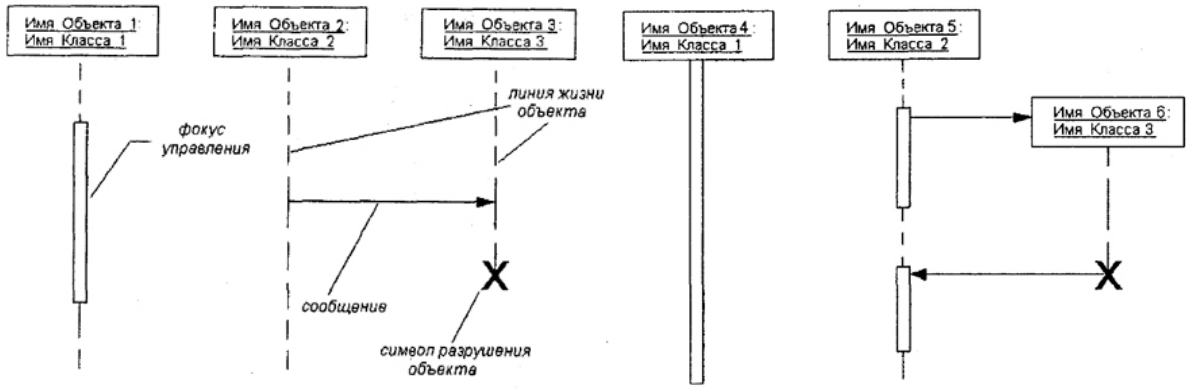

Рис. 2.15 Отображение объектов на диаграмме

Правее изображается другой объект, который непосредственно взаимодействует с первым. Таким образом, все объекты на диаграмме последовательности образуют некоторый порядок, определяемый степенью активности этих объектов при взаимодействии друг с другом.

Второе измерение диаграммы последовательности - *вертикальная временная* ось, направленная сверху вниз. Начальному моменту времени соответствует самая верхняя часть диаграммы. При этом взаимодействия объектов реализуются посредством *сообщений*, которые посылаются одними объектами другим.

Сообщения изображаются в виде горизонтальных стрелок с именем сообщения и также образуют порядок по времени своего возникновения. Другими словами, сообщения, расположенные на диаграмме последовательности выше, инициируются раньше тех, которые расположены ниже. При этом масштаб на оси времени не указывается, поскольку диаграмма последовательности моделирует лишь временную упорядоченность взаимодействий типа "раньше-позже".

Линия жизни

Линия жизни объекта изображается пунктирной вертикальной линией, ассоциированной с единственным объектом на диаграмме последовательности.

Линия жизни служит для обозначения периода времени, в течение которого объект существует в системе и, следовательно, может потенциально участвовать во всех ее взаимодействиях. Если объект существует в системе постоянно, то и его линия жизни должна продолжаться по всей плоскости диаграммы последовательности от самой верхней ее части до самой нижней (объекты 1 и 2 на рис. 2.15).

Отдельные объекты, выполнив свою роль в системе, могут быть уничтожены (разрушены), чтобы освободить занимаемые ими ресурсы. Для таких объектов линия жизни обрывается в момент его уничтожения. Для обозначения момента уничтожения объекта используется специальный символ в форме латинской буквы "X" (объект 3). Ниже этого символа пунктирная линия не изображается, поскольку соответствующего объекта в системе уже нет, и этот объект должен быть исключен из всех последующих взаимодействий.

Вовсе не обязательно создавать все объекты в начальный момент времени. Отдельные объекты в системе могут создаваться по мере необходимости, существенно экономя ресурсы системы и повышая ее производительность. В этом случае прямоугольник такого объекта изображается не в верхней части диаграммы последовательности, а в той ее части, которая соответствует моменту создания объекта (объект 6). При этом прямоугольник объекта вертикально располагается в том месте диаграммы, которое по оси времени совпадает с моментом его возникновения в системе. Очевидно, объект обязательно создается со своей линией жизни и, возможно, с фокусом управления.

Фокус управления

В процессе функционирования объектно-ориентированных систем одни объекты могут находиться в активном состоянии, непосредственно выполняя определенные действия или в состоянии пассивного ожидания сообщений от других объектов.

Чтобы явно выделить подобную активность объектов, в языке UML применяется специальное понятие, получившее название *фокуса управления*.

Фокус управления изображается в форме вытянутого узкого прямоугольника (см. объект 1 на рис. 2.15), верхняя сторона которого обозначает начало получения фокуса управления объекта (начало активности), а ее нижняя сторона - окончание фокуса управления (окончание активности). Этот прямоугольник располагается ниже обозначения соответствующего объекта и может заменять его линию жизни (объект 4), если на всем ее протяжении он является активным.

С другой стороны, периоды активности объекта могут чередоваться с периодами его пассивности или ожидания. В этом случае у такого объекта имеются несколько фокусов управления (объект 5).

Важно понимать, что получить фокус управления может только существующий объект, у которого в этот момент имеется линия жизни. Если же некоторый объект был уничтожен, то вновь возникнуть в системе он уже не может. Вместо него лишь может быть создан другой экземпляр этого же класса, который, строго говоря, будет являться другим объектом.

В отдельных случаях инициатором взаимодействия в системе может быть актер (следующий пример). В этом случае актер изображается на диаграмме последовательности самым первым объектом слева со своим фокусом управления.

Чаще всего актер и его фокус управления будут существовать в системе постоянно, отмечая характерную для пользователя активность в инициировании взаимодействий с системой. При этом сам актер может иметь собственное имя либо оставаться анонимным.

Ветвление потока управления

Для изображения ветвления рисуются две или более стрелки, выходящие из одной точки фокуса управления объекта (фокус управления объекта 1 на рис. 2.16). При этом соответствующие условия должны быть явно указаны рядом с каждой из стрелок в форме сторожевого условия.

Если условие записано в форме булевского выражения, то ветвление будет содержать только две ветви. В любом случае условия должны взаимно исключать одновременную передачу альтернативных сообщений.

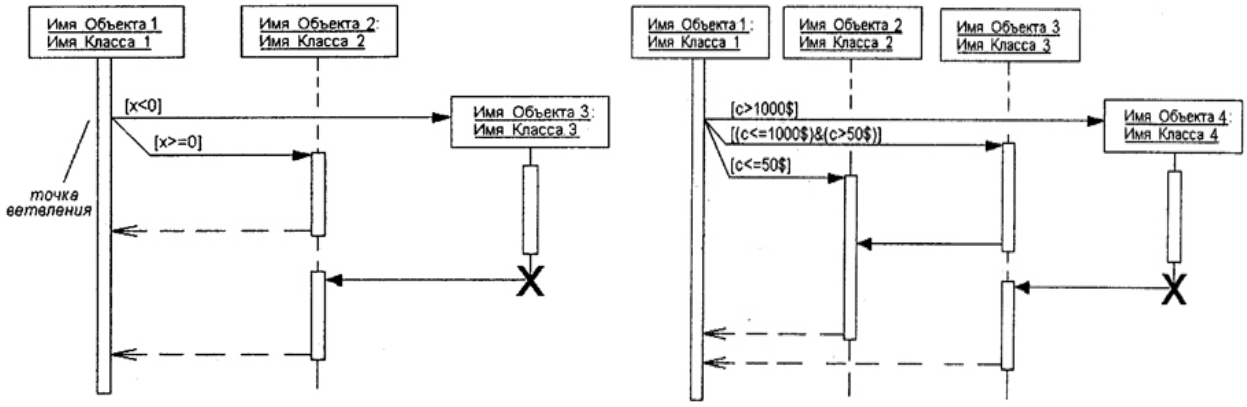

Рис. 2.16 Реализация ветвлений потока управления

С помощью ветвления можно изобразить и более сложную логику взаимодействия объектов между собой (фокус управления объекта 1 на рис. справа).

Если условий более двух, то для каждого из них необходимо предусмотреть ситуацию единственного выполнения. Рассматриваемый пример относится к моделированию взаимодействия программной системы обслуживания клиентов в банке. На этом примере диаграммы последовательности объект 1 передает управление одному из трех других объектов.

Например, прецедент «Снять деньги» предусматривает несколько возможных последовательностей: снятие денег, попытка снять деньги при отсутствии их достаточного количества на счету и другие.

Нормальный сценарий снятия 20$ со счета (при отсутствии таких проблем, как неправильный пин-код или недостаток денег на счету) показан на рис. 2.15.

В верхней части показаны все действующие лица и объекты, требуемые для выполнения ВИ «Снять деньги».

Стрелки соответствуют сообщениям, передаваемым между действующим лицом и объектом или между объектами для выполнения требуемых функций.

Показаны именно объекты, а не классы.

Прецедент начинается, когда клиент вставляет карточку в устройство чтения (объект), он считывает номер карточки и т.д. по номерам.

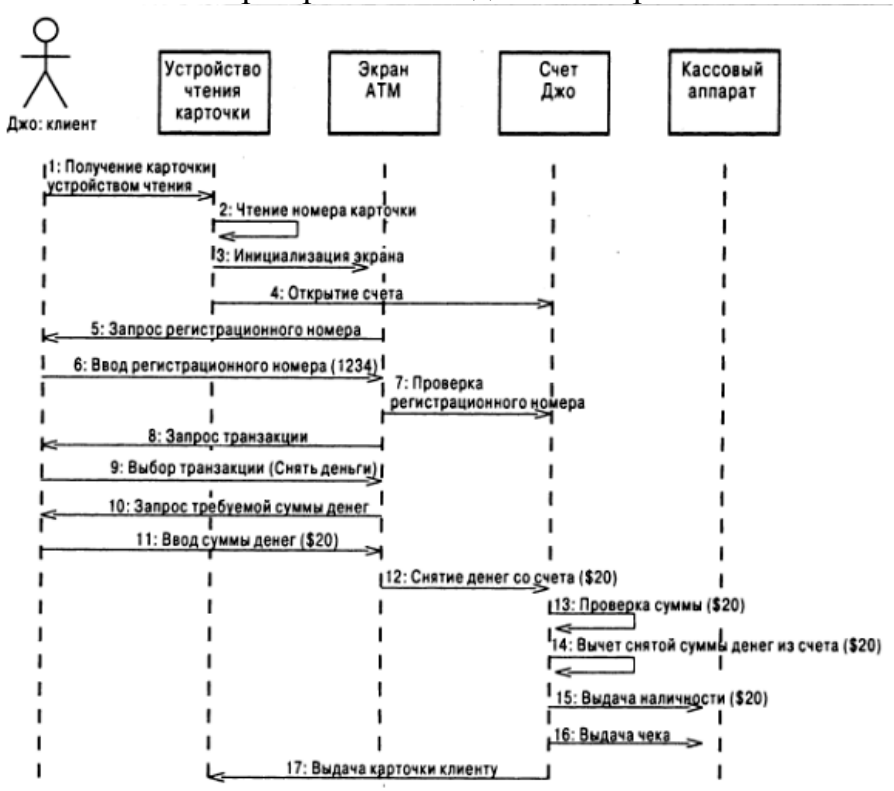

Рис. 2.17 Диаграмма последовательности для прецедента «Снять деньги

Таким образом, диаграмма последовательности иллюстрирует последовательность действий, реализующий вариант использования «Снять деньги со счета» на конкретном примере.

Изучая эту диаграмму, пользователи знакомятся со спецификой своей работы. Аналитики видят последовательность (поток) действий, разработчики – объекты, которые надо создать и их операции, специалисты по контролю качества смогут разработать тесты для проверки.

Отдельная диаграмма последовательности может показать только один поток управления (хотя с помощью нотации UML для итераций и ветвлений можно проиллюстрировать простые вариации). Поэтому, как правило, создают несколько диаграмм взаимодействий, одни из которых считаются основными, а другие описывают альтернативные пути и исключительные условия. Такой набор диаграмм последовательностей можно организовать в пакет, дав каждой диаграмме подходящее имя, отличающее ее от остальных.

Диаграмма классов

Диаграмма классов (class diagram) служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования.

Диаграмма классов может отражать, в частности, различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывает их внутреннюю структуру и типы отношений.

На данной диаграмме не указывается информация о временных аспектах функционирования системы. С этой точки зрения диаграмма классов является дальнейшим развитием концептуальной модели проектируемой системы.

Диаграмма классов представляет собой некоторый граф, вершинами которого являются элементы типа "классификатор", которые связаны различными типами структурных отношений. Следует заметить, что диаграмма классов может также содержать интерфейсы, пакеты, отношения и даже отдельные экземпляры, такие как объекты и связи. Когда говорят о данной диаграмме, имеют в виду статическую структурную модель проектируемой системы.

Поэтому диаграмму классов принято считать графическим представленном таких структурных взаимосвязей логической модели системы, которые не зависят или инвариантны от времени.

На диаграмме классов *класс* создается для каждого типа объектов из диаграмм последовательности.

Графически класс изображается в виде прямоугольника, который дополнительно может быть разделен горизонтальными линиями на разделы или секции. В этих разделах могут указываться имя класса, атрибуты (переменные) и операции (методы).

Обязательным элементов обозначения класса является его имя. На начальных этапах разработки диаграммы для классов можно указывать только имя соответствующего класса. По мере проработки отдельных компонентов классы дополняются атрибутами и операциями.

Диаграмма классов для варианта использования «Снять деньги» показана на рис. 2.18.

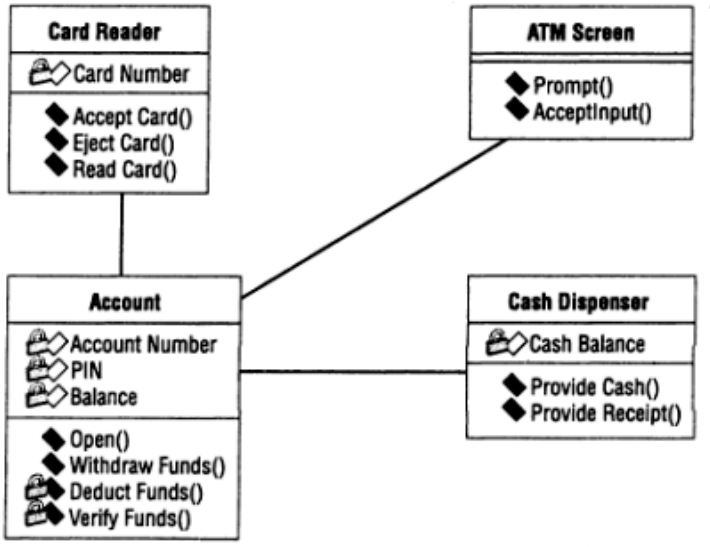

Рис. 2.18 Диаграмма классов для прецедента «Снять деньги»

В этом процессе задействованы четыре класса: Card Reader (устройство для чтения карточек), Account (счет), ATM Screen (экран) и Cash Dispenser (кассовый аппарат).

Например, класс Account (счет) имеет три атрибута: *номер счета*, *идентификационный номер* и *баланс*. В последней части указаны операции, отражающие его поведение: *Открыть*, *Снять деньги*, *Вычесть сумму из счета*, *Проверить наличие денег.*

Применение диаграмм классов

Выше было отмечено, что диаграммы классов применяют для моделирования статического вида системы с точки зрения проектирования. В этом представлении удобнее всего описывать функциональные требования к системе, т.е. услуги, которые она предоставляет конечному пользователю.

Обычно диаграммы классов используются в следующих целях:

- для *моделирования словаря* системы. Моделирование словаря системы предполагает принятие решения о том, какие абстракции являются частью системы, а какие - нет. С помощью диаграмм классов вы можете определить эти абстракции и их обязанности;
- для *моделирования простых кооперaций*. Кооперация - это сообщество классов, интерфейсов и других элементов, работающих совместно для обеспечения некоторого кооперативного поведения, более значимого, чем сумма составляющих его элементов. Например, моделируя семантику транзакций в распределенной системе, вы не сможете понять происходящие процессы, глядя на один-единственный класс, поскольку соответствующая семантика обеспечивается несколькими совместно работающими классами. С помощью диаграмм классов удается визуализировать и специфицировать эти классы и отношения между ними;

- для *моделирования логической схемы базы данных*. Логическую схему можно представлять себе как чертеж концептуального проекта базы данных. Во многих сферах деятельности требуется хранить устойчивую информацию в реляционной или объектно-ориентированной базе данных. Моделировать схемы также можно с помощью диаграмм классов.

Простые кооперации

Классы не существуют сами по себе. Любой класс функционирует совместно с другими, реализуя семантику, выходящую за границы каждого отдельного элемента. Таким образом, кроме определения словаря системы нужно уделить внимание визуализации, специфицированию, конструированию и документированию различных способов совместной работы вошедших в словарь сущностей. Для моделирования такого "сотрудничества" применяются диаграммы классов.

Создавая диаграмму классов, вы просто моделируете часть сущностей и отношений, описываемых в представлении системы с точки зрения проектирования. Желательно, чтобы каждая диаграмма описывала только одну кооперацию.

Моделирование кооперации осуществляется следующим образом:

1. Идентифицируйте механизмы, которые вы собираетесь моделировать. Механизм - это некоторая функция или поведение части моделируемой системы, появляющееся в результате взаимодействия сообщества классов, интерфейсов и других сущностей. (Механизмы, как правило, тесно связаны с прецедентами, см. главу 16.)
2. Для каждого механизма идентифицируйте классы, интерфейсы и другие кооперации, принимающие участие в рассматриваемой кооперации. Идентифицируйте также отношения между этими сущностями.
3. Проверьте работоспособность системы с помощью прецедентов (см. главу 15). По ходу дела вы, возможно, обнаружите, что некоторые ее части оказались пропущены, а некоторые - семантически неправильны.
4. Заполните идентифицированные элементы содержанием. Что касается классов, начните с правильного распределения обязанностей; позже можно будет превратить их в конкретные атрибуты и операции.

Диаграммы состояний и деятельности

Понятие автомата и деятельности

С помощью взаимодействий (они были описаны ранее) можно моделировать поведение сообщества совместно работающих объектов. Для моделирования поведения отдельного объекта применяется автомат.

Автомат (State machine) описывает поведение в терминах последовательности состояний, через которые проходит объект в течение своей жизни, отвечая на события, а также его реакций на эти события.

Автоматы используются для моделирования динамических аспектов системы. В основном под этим понимается описание жизни экземпляров класса, прецедентов и системы в целом. Экземпляры могут реагировать на такие события, как сигналы, операции или истечение промежутка времени.

Диаграмма, которая описывает все возможные состояния, в которых может находиться объект в процессе работы системы называется диаграммой состояний. Она указывает, какие события могут влиять на объект, и в каких ситуациях он меняет свое состояние.

Действие (Action) - это атомарное вычисление, которое приводит к изменению состояния модели или возврату значения.

Событие - это спецификация существенного факта, имеющего место в пространстве и во времени. В контексте автоматов событие - это некий стимул, инициирующий переход из одного состояния в другое.

Состояние объекта - это ситуация в его жизни, на протяжении которой он удовлетворяет некоторому условию, осуществляет определенную деятельность или ожидает какого-то события.

Переход - это отношение между двумя состояниями, показывающее, что объект, находящийся в первом состоянии, должен выполнить определенные действия и перейти во второе состояние, как только произойдет указанное событие, и будут удовлетворены определенные условия.

Графически состояние изображается в виде прямоугольника с закругленными углами. Переход обозначается линией со стрелкой.

Когда происходит событие, в зависимости от текущего состояния объекта имеет место та или иная деятельность.

Деятельность (Activity) -это занимающее некоторое время неатомарное вычисление внутри автомата. Результатом деятельности является некоторое действие, составленное из атомарных вычислений, которое приводит к изменению состояния модели или возврату значения. Последовательность деятельностей составляют диаграмму деятельности (активности).

Визуализировать автомат можно двумя способами: выделяя передачу потока управления от одной деятельности к другой (с помощью диаграммы деятельности) или выделяя потенциальные состояния объектов и переходы между ними (с помощью диаграммы состояний).

Хорошо структурированные автоматы подобны хорошо структурированным алгоритмам, они эффективны, просты, адаптируемы к разным ситуациям и просты для понимания.

Как работает автомат

У каждого объекта есть некоторое время жизни. Объект рождается, когда его создают, и перестает существовать после уничтожения. В промежутке он может воздействовать на другие объекты (посылая им сообщения, а также сам подвергаться воздействиям (являясь получателем сообщения). Во многих случаях

сообщения могут быть простыми синхронными вызовами. Например, экземпляр класса **Клиент** может инициировать операцию *получитьБалансСчета* на экземпляре класса **БанковскийСчет**. Чтобы специфицировать поведение такого рода объектов, автомат не нужен, поскольку в любой момент времени их поведение не зависит от прошлого.

В системах иного типа встречаются объекты, которые должны реагировать на сигналы, асинхронные стимулы, которыми обмениваются экземпляры. Например, сотовый телефон должен отвечать на вызовы от других телефонов, которые происходят в случайные моменты времени, на события нажатия клавиши владельцем, набирающим номер, и на события сети (когда телефон перемещается из одной ячейки сотовой связи в другую). Бывают также объекты, текущее поведение которых зависит от прошлого. Так, поведение системы наведения ракеты "воздух-воздух" зависит от ее текущего состояния, например НеЛетит (вряд ли стоит запускать ракету с самолета, находящегося на земле) или Поиск (нельзя ее запускать, пока нет ясного представления, какую цель надо поразить).

Если объекты должны реагировать на асинхронные стимулы или их текущее поведение зависит от их прошлого, лучше всего описывать это поведение с помощью автомата. К этой категории относятся экземпляры классов, которые могут получать сигналы, включая и многие активные объекты. Фактически объект, который получает сигнал, но не имеет описывающего его автомата, попросту проигнорирует этот сигнал. Автоматы применяются также для моделирования систем в целом, в особенности реактивных, то есть таких, которые должны отвечать на сигналы от внешних актеров.

Состояния

Было отмечено, что состояние - это ситуация в жизни объекта, на протяжении которой он удовлетворяет некоторому условию, выполняет определенную деятельность или ожидает какого-то события. Объект остается в некотором состоянии в течение конечного отрезка времени. Например, Обогреватель в доме может находиться в одном из четырех состояний:

- Ожидание (ждет команды "начать обогревать дом").
- Активация (газ подается, но не достигнута нужная температура),.
- Активен (газ и вентилятор включены).
- Выключается (газ не подается, но вентилятор еще включен и удаляет остаточное тепло из системы).

Когда автомат объекта находится в данном состоянии, говорят, что и объект находится в этом состоянии. Например, экземпляр класса **Обогреватель** может находиться в состоянии *Ожидание* или *Выключается*.

При указании состояния определяют следующие элементы:

- *имя* - текстовая строка, которая отличает одно состояние от всех остальных. Имя состояния должно быть уникальным внутри объемлющего пакета;

- *действия при входе/выходе* - действия, выполняемые при входе в состояние и выходе из него;
- *внутренние переходы* - переходы, обрабатываемые без выхода из состояния;
- *подсостояния* - внутри состояния могут существовать подсостояния, как непересекающиеся (активизируемые последовательно), так и параллельные (активные одновременно);

Имя состояния может иметь произвольную длину и включать любые цифры и буквы, а также некоторые знаки препинания (за исключением имеющих специальный смысл, например двоеточия). Имя может занимать несколько строк. На практике оно чаще всего бывает коротким, составленным из одного или нескольких существительных, взятых из словаря моделируемой системы. Первую букву каждого слова в имени состояния принято делать заглавной, например *Ожидание* или *ИдетОтключение*.

Как показано на рис. 2.19, состояние изображается прямоугольником с закругленными углами.

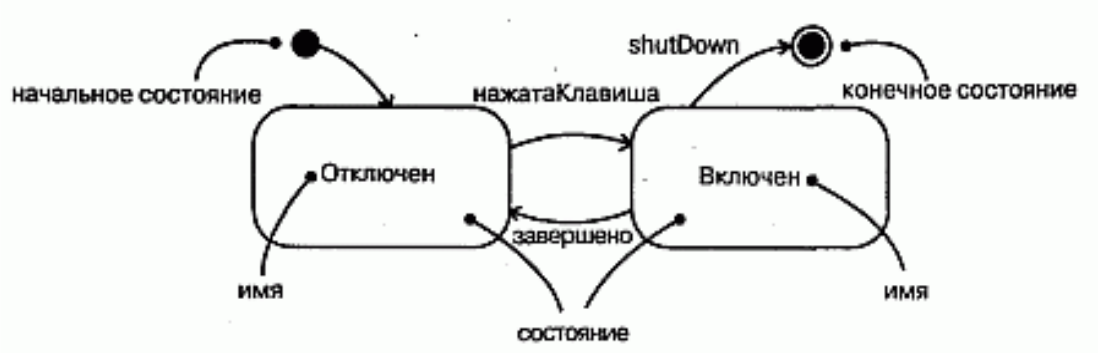

Рис. 2.19 Состояния

При описании состояний можно определить два специальных состояния: *начальное и конечное состояния*. В начальном состоянии автомат или подсостояние находятся по умолчанию в исходный момент времени. Изображается оно в виде закрашенного кружка. В конечном состоянии завершается выполнение автомата или объемлющего состояния. Оно представлено закрашенным кружком, заключенным в окружность.

Обычно начальное и конечное состояния - это псевдосостояния, они не имеют ни одной из характеристик нормального состояния, кроме имени. Однако переход из начального состояния в конечное может обладать всеми свойствами, включая сторожевое условие и действие (но не событие-триггер).

Переходы

Переход - это отношение между двумя состояниями, показывающее, что объект, находящийся в первом состоянии, должен выполнить определенные действия и перейти во второе состояние, как только произойдет указанное

событие, и будут удовлетворены указанные условия. Говорят, что при таком изменении состояния переход *срабатывает*. Пока переход не сработал, объект находится в исходном состоянии; после срабатывания он находится в целевом состоянии. Например, **Обогреватель** может перейти из состояния *Ожидание* в состояние *Активация* при возникновении события **tooCold** (*слишкомХолодно*) с параметром **desiredTemp** (*желаемаяТемпература*).

Переход определяют пять элементов:

- *исходное состояние* - состояние, из которого происходит переход. Если объект находится в исходном состоянии, то исходящий переход может сработать, когда объект получит событие-триггер, инициирующее этот переход, причем должно быть выполнено сторожевое условие (если оно задано);
- *событие-триггер* - событие при получении которого объектом, находящимся в исходном состоянии, может сработать переход (при этом должно быть выполнено сторожевое условие);
- *сторожевое условие* - булевское выражение, которое вычисляется при получении события-триггера. Если значение истинно, то переходу разрешено сработать, если ложно - переход не срабатывает. Если при этом не задано никакого другого перехода, инициируемого тем же самым событием, то событие теряется;
- *действие* - атомарное вычисление, которое может непосредственно воздействовать на объект, владеющий автоматом, или оказать косвенное воздействие на другие объекты, находящиеся в области видимости;
- *целевое состояние* - состояние, которое становится активным после завершения перехода.

На рис. 2.20 показаны переходы в виде линии со стрелкой, направленной от исходного состояния к целевому состоянию. Исходное и целевое состояния перехода в себя совпадают.

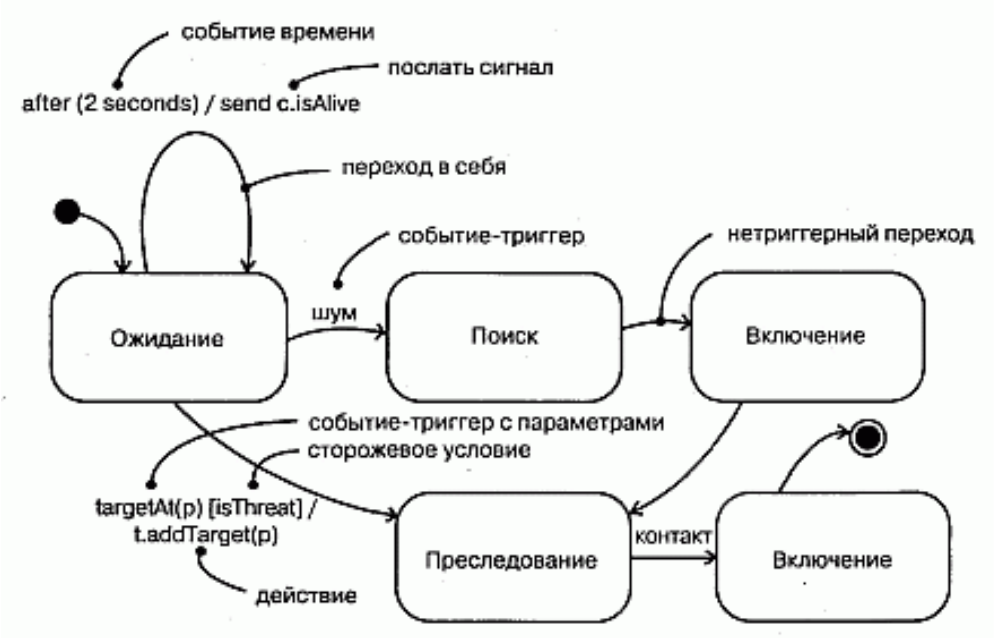

Рис. 2.20 Состояния и переходы

Возможны случаи, когда у перехода может быть несколько исходных состояний (в этом случае он представляет собой слияние нескольких параллельных состояний), а также несколько целевых состояний (в этом случае он представляет собой разделение на несколько параллельных состояний).

Событие-триггер. Известно, что событие представляет собой спецификацию существенного факта, происходящего в пространстве и во времени. В контексте автоматов событием является неким стимулом, инициирующим переход из одного состояния в другое. Из рис. 2.20 видно, что в число событий включаются сигналы, вызовы, истечение промежутка времени или изменение состояния. У сигнала и вызова могут быть параметры, значения которых доступны переходу, в том числе при вычислении сторожевого условия и выполнении действия.

Существуют и нетриггерные переходы, для которых нет никакого события-триггера. Нетриггерный переход, называемый также переходом по завершении, инициируется неявно, когда работа в исходном состоянии закончится. Такие переходы характерны для диаграммы деятельности.

Сторожевое условие. На рис. 2.20 показано, что сторожевое условие изображается булевским выражением, заключенным в квадратные скобки, которое помещается после события-триггера. Сторожевое условие вычисляется только после возникновения события-триггера, инициирующего соответствующий переход. Поэтому из одного состояния может быть несколько переходов с одним и тем же событием-триггером в том случае, если никакие два сторожевых условия не становятся одновременно истинными.

Сторожевое условие вычисляется ровно один раз для каждого перехода в момент наступления события, но может быть вычислено снова, если переход инициируется повторно. В булевское выражение можно включать условия, в которых фигурирует состояние объекта.

Действие - это атомарное вычисление. К действиям относятся вызовы операций (объекта, который владеет автоматом, а также любых других видимых объектов), создание и уничтожение другого объекта или посылка объекту сигнала. Как видно из рис. 2.20, для обозначения посылки сигнала определена специальная нотация - имени сигнала предшествует ключевое слово **send**.

Действие (Action) всегда атомарно, то есть не может быть прервано другим событием и, следовательно, выполняется до полного завершения. Этим оно отличается от деятельности (Activity), выполнение которой может быть прервано другими событиями.

Подсостояния

Рассмотренные выше свойства состояний и переходов решают целый ряд типичных проблем моделирования автоматов. У автоматов, рассматриваемых в UML, есть свойство, которое позволяет еще больше упростить моделирование сложного поведения. Это возможность указать **подсостояние** (Substate) - состояние, являющееся частью другого состояния. Например, **Обогреватель** может находиться в состоянии *Обогрев*, внутри которого содержится еще одно состояние – *Активация*. В таком случае говорят, что объект находится одновременно в состояниях *Обогрев* и *Активация*.

Простым называется такое состояние, которое не имеет внутренней структуры.

Состояние, у которого есть подсостояния, то есть вложенные состояния, именуется **составным**.

Составное состояние может содержать как параллельные (независимые), так и последовательные (непересекающиеся) подсостояния.

В UML составное состояние изображается так же, как и простое, но имеет дополнительный графический раздел, в котором показан вложенный автомат. Глубина вложенности состояний не ограничена, причем вложенная структура составного состояния подобна композиции.

Пример. Рассмотрим уже известную нам задачу моделирования банкомата.

Анализ предметной области показал, что система может находиться в одном из трех основных состояний:

Ожидание (действий со стороны пользователя),

Активен (выполняет транзакцию, запрошенную пользователем)

Обслуживается (возможно, пополняется запас банкнот).

В состоянии *Активен* поведение банкомата описывается простой схемой: проверить счет пользователя, выбрать тип транзакции, выполнить транзакцию, напечатать квитанцию.

После печати банкомат возвращается в состояние *Ожидание*. Эти этапы можно представить как состояния *Проверка*, *Выбор*, *Обработка*, *Печать*. Стоило бы даже дать пользователю возможность выбрать и выполнить несколько транзакций после того, как проверка счета выполнена, но еще до печати окончательной квитанции.

Проблема в том, что на любом этапе пользователь может отменить транзакцию и вернуть банкомат в состояние *Ожидание*. С помощью простых автоматов реализовать это можно, но не очень удобно. Поскольку пользователь имеет право отменить транзакцию в любой точке, пришлось бы включать соответствующий переход из любого состояния в последовательности *Активен*. Это может привести к ошибкам, так как очень легко забыть включить все необходимые переходы, а наличие множества событий, прерывающих основной поток управления, приведет к появлению большого числа переходов из разных исходных состояний, которые оканчиваются в одном и том же целевом. При этом у каждого такого перехода будет одинаковое событие-триггер, сторожевое условие и действие.

Использование последовательных подсостояний позволяет упростить моделирование этой задачи, как показано на рис. 2.21.

Рис. 2.21 Последовательные подсостояния

У состояния *Активен* имеется внутренний автомат, в который входят подсостояния *Проверка*, *Выбор*, *Обработка*, *Печать*. Состояние банкомата изменяется с *Ожидание* на *Активен*, когда пользователь вставляет в прорезь кредитную карту. При входе в состояние *Активен* выполняется действие readCard (прочитатьКарту). Начав с исходного состояния внутреннего автомата, управление переходит в состояние *Проверка*, затем - в *Выбор*, и, наконец, в состояние *Обработка*. После выхода из состояния *Обработка* управление может вернуться в *Выбор* (если пользователь избрал другую транзакцию) или попасть в

Печать. Из состояния *Печать* предусмотрен нетриггерный переход назад в *Ожидание*. Обратите внимание, что у состояния *Активен* есть действие при выходе, которое обеспечивает выбрасывание кредитной карты (ejectCard).

Обратите внимание также на переход из состояния *Активен* в состояние *Ожидание*, инициируемый событием cancel. В любом подсостояний состояния *Активен* пользователь может отменить транзакцию, что вернет банкомат в состояние *Ожидание* (но лишь после возврата кредитной карты владельцу, то есть после выполнения действия при выходе из состояния *Активен*, что произойдет вне зависимости от того, чем вызван переход из этого состояния).

Если бы подсостояний не было, то пришлось бы вводить переход, инициируемый событием cancel из каждого состояния вложенного автомата.

Такие подсостояния, как *Проверка* и *Обработка*, называются последовательными или непересекающимися. Если в объемлющем составном состоянии имеется несколько непересекающихся подсостояний, то говорят, что объект одновременно находится в составном состоянии и ровно в одном из подсостояний.

Таким образом, последовательные подсостояния разбивают множество вариантов составного состояния на непересекающиеся (дизъюнктные) части.

Переход из состояния, находящегося вне объемлющего составного состояния, может вести как в само это составное состояние, так и в любое из его подсостояний. Если целевое состояние является составным, то вложенный автомат должен иметь некоторое начальное состояние, куда управление попадает при входе в составное состояние после выполнения ассоциированного с ним действия при входе (если таковое определено). Если же целевым является одно из вложенных состояний, то управление попадает в него, но после выполнения действий при входе в объемлющее составное состояние и в подсостояние (если таковые определены).

Для перехода, исходящего из составного состояния, исходным может быть как оно само, так и какое-либо из его подсостояний. В любом случае управление сначала покидает вложенное подсостояние (тогда выполняется его действие при выходе, если оно определено), а затем составное состояние (тогда также выполняется действие при выходе). Переход, для которого исходным является составное состояние, по существу прерывает работу вложенного автомата.

При моделировании жизненного цикла объекта особое внимание уделяется специфицированию следующих элементов: событий, на которые объект должен реагировать, реакций на такие события, а также влияния прошлого на поведение в текущий момент. Кроме того, необходимо решить, в каком порядке объект будет осмысленно реагировать на события, начиная с момента его создания и вплоть до уничтожения.

При моделировании жизненного цикла объекта можно придерживаться следующих рекомендаций:

1. Выберите контекст для автомата, будь то класс, прецедент или система в целом, при этом:
    - если контекстом является класс или прецедент, идентифицируйте соседние классы, включая предков, а также все классы, доступные из данного с помощью зависимостей или ассоциаций (такие соседи - кандидаты на роль целей для действий или на включение в сторожевые условия);
    - если контекстом является система в целом, то следует сосредоточиться на какой-либо одной стороне ее поведения (теоретически каждый объект в системе может принимать участие в модели ее жизненного цикла, но за исключением простейших случаев такую полную модель практически невозможно воспринять).
2. Установите для объекта начальное и конечное состояния. Если предполагается его использование в остальной части модели, возможно, стоит сформулировать пред- и постусловия для начального и конечного состояний.
3. Решите, на какие события объект может реагировать. Если интерфейсы объекта уже специфицированы, то в них и содержится описание таких событий; в противном случае необходимо рассмотреть, какие объекты могут взаимодействовать с объектом в данном контексте и какие события они могут посылать.
4. Изобразите все состояния верхнего уровня, от начального до конечного, в которых может находиться объект. Соедините эти состояния переходами, инициируемыми теми или иными событиями, а затем свяжите с этими переходами действия.
5. Идентифицируйте все действия при входе и выходе (особенно если обнаружится, что выражаемая с их помощью идиома используется в автомате).
6. Если это необходимо, разверните выявленные к этому моменту состояния, применяя аппарат подсостояний.
7. Убедитесь, что все события, встречающиеся в автомате, соответствуют тем, которые ожидает интерфейс объекта. Следует убедиться также в том, что все события, ожидаемые интерфейсом объекта, нашли отражение в автомате. Наконец, нужно выявить те места, где имеет смысл игнорировать события.
8. Убедитесь, что все действия, упомянутые в автомате, поддержаны отношениями, методами и операциями объемлющего объекта.
9. Следуя переходам автомата вручную или с помощью инструментальных средств, проверьте ожидаемые последовательности событий и реакций на них. Особое внимание нужно обратить на недостижимые состояния и состояния, в которых автомат может "застрять" (то есть тупиковые).

10. Изменив по результатам проверки структуру автомата, снова проверьте его на ожидаемые последовательности событий, чтобы убедиться, что семантика объекта не изменилась.

Диаграммы деятельности

Диаграммы деятельности - это один из пяти видов диаграмм, применяемых в UML для моделирования динамических аспектов поведения системы (применяются еще диаграммы последовательностей и кооперации, состояний, прецедентов).

Диаграмма деятельности представляет, по существу, блок-схему, которая показывает, как поток управления переходит от одной деятельности к другой.

Диаграммы деятельности, как правило, они применяются, чтобы промоделировать последовательные и при необходимости параллельные шаги вычислительного процесса. С помощью диаграмм деятельности можно также моделировать жизнь объекта, когда он переходит из одного состояния в другое в разных точках потока управления.

Диаграммы деятельности могут использоваться самостоятельно для визуализации, специфицирования, конструирования и документирования динамики совокупности объектов, но они пригодны также и для моделирования потока управления при выполнении некоторой операции. Если в диаграммах взаимодействий акцент делается на переходах потока управления от объекта к объекту, то диаграммы деятельности описывают переходы от одной деятельности к другой.

Деятельность (Activity) - это некоторый относительно продолжительный этап выполнения в автомате. Ключевое слово в этом определении – продолжительный. В этом деятельность принципиальным образом отличается от действия. В конечном итоге деятельность сводится все-таки к некоторому действию, но которое составлено из атомарных вычислений, приводящих к изменению состояния системы или возврату значения.

Таким образом, диаграммы деятельности важны не только для моделирования динамических аспектов поведения системы, но и для построения выполняемых систем посредством прямого и обратного проектирования.

При моделировании инфокоммуникацилнных систем вы сталкиваетесь с проблемой. Например, как лучше всего промоделировать рабочий процесс или функционирование системы? То и другое - аспекты ее динамики.

С одной стороны, можно построить несколько прецедентов, описывающих взаимодействие различных представляющих интерес объектов и сообщения, которыми они обмениваются. В UML такие прецеденты можно моделировать двумя способами: делая акцент на упорядочении сообщений по времени (с помощью диаграмм последовательностей) или на структурных отношениях между взаимодействующими объектами (с помощью диаграмм кооперации). Такого рода

диаграммы взаимодействия близки к Gantt-диаграммам, в фокусе которых находятся объекты (ресурсы), выполняющие некоторую работу во времени.

С другой стороны, динамику поведения можно моделировать с помощью диаграмм деятельности, в которых внимание сосредоточено, прежде всего, на содержании деятельности, в которой принимают участие объекты, как показано на рис. 2.22. С этой точки зрения диаграммы деятельности напоминают Pert-диаграммы.

Рассматриваемая на этом рисунке диаграмма деятельности позволяет выбрать отдельный заказ, с которым необходимо что-то сделать. Диаграмма просто устанавливает основные правила последовательности действий, которые необходимо соблюдать. В этом смысле диаграмма деятельности и есть своеобразная блок-схема, которая описывает последовательность выполнения операций во времени. Ее можно представлять себе как вывернутую наизнанку диаграмму взаимодействий. Диаграмма взаимодействий - это взгляд на объекты, которые передают друг другу сообщения, а диаграмма деятельности - взгляд на операции, которые передаются от одного объекта другому. Семантическое различие трудноуловимо, но в результате нам открываются два совершенно разных взгляда на мир.

Диаграмма деятельности (Activity diagram) показывает поток переходов от одной деятельности к другой. Деятельность (Activity) - это продолжающийся во времени неатомарный шаг вычислений в автомате. Деятельности в конечном счете приводят к выполнению некоего действия (Action, см. главу 15), составленного из выполняемых атомарных вычислений, каждое из которых либо изменяет состояние системы, либо возвращает какое-то значение. Действие может заключаться в вызове другой операции, посылке сигнала, создании или уничтожении объекта либо в простом вычислении - скажем, значения выражения. Графически диаграмма деятельности представляется в виде графа, имеющего вершины и ребра.

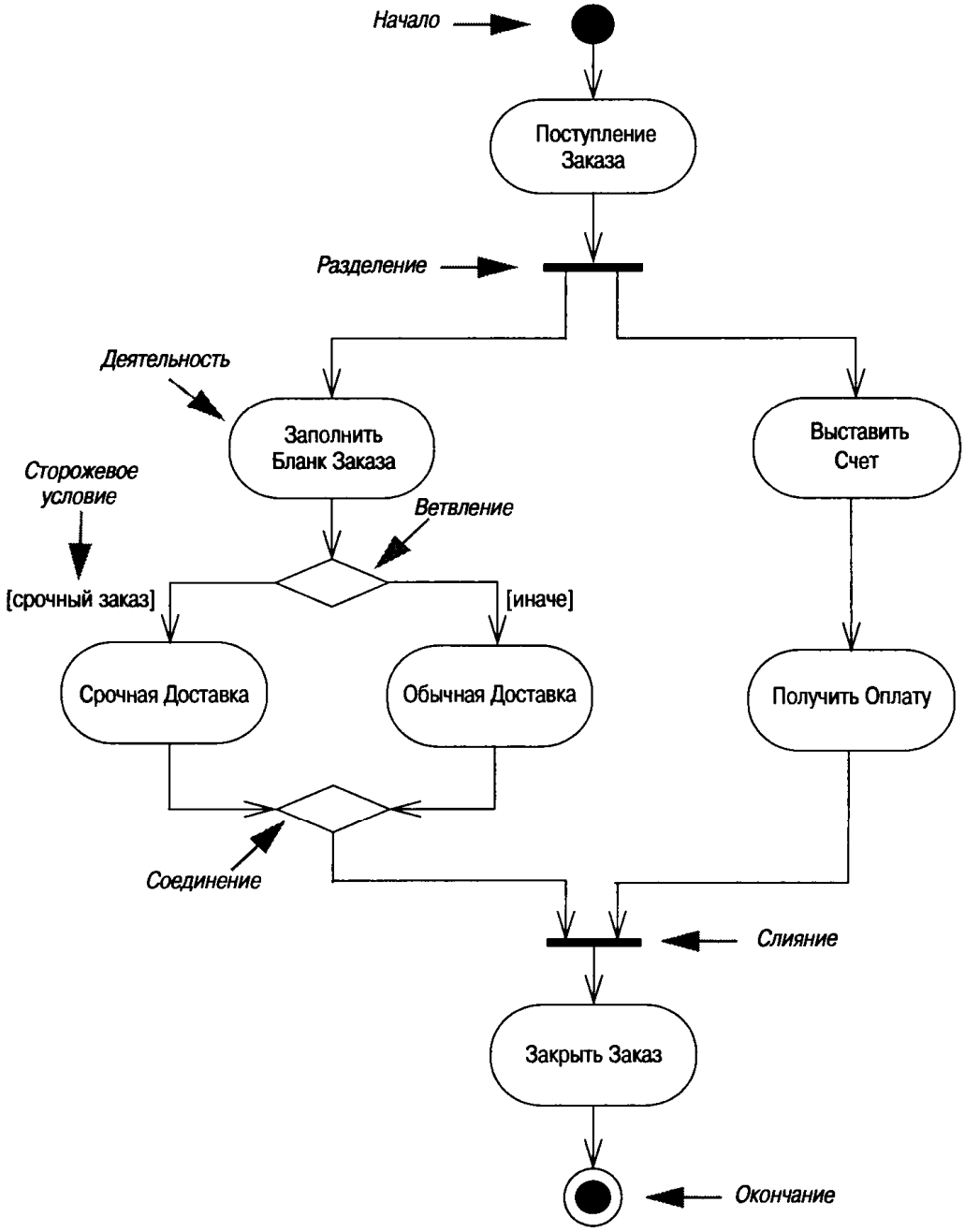

Рис. 2.22 Диаграмма деятельности

Диаграмма деятельности обладает теми же общими свойствами, которые присущи всем остальным диаграммам: именем и графическим наполнением, проецирующим ее на модель. От всех прочих диаграмму деятельности отличает ее специфичное содержание.

Диаграмма деятельности в общем случае состоит из следующих основных элементов:

- состояний деятельности и состояний действия;
- переходов и объектов.

Диаграмма деятельности, по сути, представляет собой проекцию элементов, присутствующих в графе деятельности, разновидности автомата, в которой все или большинство состояний, это состояния деятельности, а все или большинство переходов обусловлены завершением деятельности в состоянии-источнике. Поскольку диаграмма деятельности является автоматом, то к ней применимы все характеристики автоматов. Это означает, в частности, что диаграмма деятельности может содержать простые и составные состояния, точки ветвления, разделения и слияния. Также диаграмма деятельности, как и любая другая диаграмма, может содержать примечания и ограничения.

Состояния действия и состояния деятельности

В потоке управления, моделируемом диаграммой деятельности, происходят различные события. Вы можете вычислить выражение, в результате чего изменяется значение некоторого атрибута или возвращается некоторое значение. Также, например, можно выполнить операцию над объектом, послать ему сигнал или даже создать его или уничтожить. Все эти выполняемые атомарные вычисления называются состояниями действия, поскольку каждое из них есть состояние системы, представляющее собой выполнение некоторого действия. Как показано на рис. 2.23, состояния действия изображаются прямоугольниками с закругленными краями. Внутри такого символа можно записывать произвольное выражение.

Рис. 2.23 Состояния действия

UML не требует использования какого-либо специального языка для записи таких выражений. Можно просто написать любой структурированный текст, а при необходимости конкретизации заимствовать синтаксис и семантику того или иного языка программирования.

Состояния действия не могут быть подвергнуты декомпозиции. Также они атомарны. Это значит, что внутри них могут происходить различные события, но выполняемая в состоянии действия работа не может быть прервана. И наконец, обычно предполагается, что длительность одного состояния действия занимает неощутимо малое время.

В противоположность этому состояния деятельности могут быть подвергнуты дальнейшей декомпозиции, вследствие чего выполняемую деятельность можно представить с помощью других диаграмм деятельности.

Состояния деятельности не являются атомарными, то есть могут быть прерваны. Предполагается, что для их завершения требуется заметное время.

Можно считать, что состояние действия - это частный вид состояния деятельности, а конкретнее - такое состояние, которое не может быть подвергнуто дальнейшей декомпозиции. А состояние деятельности можно представлять себе как составное состояние, поток управления которого включает только другие состояния деятельности и действий. Взгляните более пристально на внутреннюю структуру состояния деятельности, и вы найдете еще одну диаграмму деятельности.

Как видно из рис. 2.24, состояния деятельности и действий обозначаются одинаково, с тем отличием, что у первого могут быть дополнительные части, такие как действия входа и выхода (то есть выполняемые соответственно при входе в состояние и выходе из него), и оно может сопровождаться спецификациями подавтоматов.

Рис. 2.24 Состояния деятельности

Состояния действий и состояния деятельности - это не что иное, как частные случаи состояний автомата. Входя в одно из таких состояний, вы просто выполняете некоторое действие или деятельность, а при выходе управление передается следующему действию или деятельности. Таким образом, состояния деятельности в какой-то мере напоминают стенограмму.

Семантически состояние деятельности эквивалентно транзитивному расширению графа деятельности по месту до тех пор, пока не останутся только действия. Тем не менее, состояния деятельности важны, поскольку они помогают разбить сложные вычисления на более простые части, точно так же, как операции используются для группировки и повторного использования выражений.

Переходы

Когда действие или деятельность в некотором состоянии завершается, поток управления сразу переходит в следующее состояние действия или деятельности. Для описания этого потока используются переходы (Transitions), показывающие путь из одного состояния действия или деятельности в другое.

В UML переход представляется простой линией со стрелкой, как показано ранее на рис. 2.22.

Такие переходы называются переходами по завершении, или не-триггерными (Triggerless), поскольку управление по завершении работы в исходном состоянии немедленно передается дальше. После того как действие в данном исходном состоянии закончилось, выполняется определенное для него действие выхода (если таковое имеется). Далее, безо всякой задержки, поток управления следует

переходу и попадает в очередное состояние действия или деятельности. При этом выполняется определенное для нового состояния действие входа (если таковое имеется), затем - действие или деятельность самого состояния и следующий переход.

Управление может таким образом переходить из состояния в состояние неопределенно долго (в случае незавершающейся деятельности) или до попадания в конечное состояние. Нетриггерные переходы могут иметь сторожевые условия, которые обусловливают их активизацию.

Поток управления должен где-то начинаться и заканчиваться (разумеется, если это не бесконечный поток, у которого есть начало, но нет конца). Можно задать как начальное состояние (закрашенный кружок), так и конечное (закрашенный кружок внутри окружности).

Ветвление

Простые последовательные переходы встречаются наиболее часто, но их одних недостаточно для моделирования любого потока управления. Как и в блок-схеме, вы можете включить в модель ветвление, которое описывает различные пути выполнения в зависимости от значения некоторого булевского выражения. Как видно из рис. 2.22, точка ветвления представляется ромбом.

В точку ветвления может входить ровно один переход, а выходить - два или более. Для каждого исходящего перехода задается булевское выражение, которое вычисляется только один раз при входе в точку ветвления. Ни для каких двух исходящих переходов эти сторожевые условия не должны одновременно принимать значение "истина", иначе поток управления окажется неоднозначным. Но эти условия должны покрывать все возможные варианты, иначе поток остановится. Ветвление семантически эквивалентно множественным переходам со сторожевыми условиями.

Для удобства разрешается использовать ключевое слово **else** для пометки того из исходящих переходов, который должен быть выбран в случае, если условия, заданные для всех остальных переходов, не выполнены.

Реализовать итерацию можно, если ввести два состояния действия - в первом устанавливается значение счетчика, во втором оно увеличивается - и точку ветвления, вычисление в которой показывает, следует ли прекратить итерации. Ветвление и итерация возможны также на диаграммах взаимодействий.

UML не предписывает язык для этих выражений. В общем случае вы можете использовать структурированный текст; для большей строгости можно воспользоваться синтаксисом и семантикой определенного языка программирования.

Разделение и слияние

Простые и ветвящиеся последовательные переходы в диаграммах деятельности используются чаще всего. Однако можно встретить и параллельные потоки, и это особенно характерно для моделирования бизнес-процессов.

В UML для обозначения разделения и слияния таких параллельных потоков выполнения используется синхронизационная черта, которая рисуется в виде жирной вертикальной или горизонтальной линии. Каждый из параллельно выполняющихся потоков управления существует в контексте независимого активного объекта, который, как правило, моделируется либо процессом, либо вычислительным потоком.

Рассмотрим, например, параллельные потоки, используемые при оформлении заказа некого товара.

Как показано на рис. 2.22, точка разделения соответствует расщеплению одного потока управления на два выполняющихся параллельно. В этой точке может существовать ровно один входящий переход и два или более исходящих. Каждый исходящий переход представляет один независимый поток управления. После точки разделения деятельности, ассоциированные с каждым путем в графе, продолжают выполняться параллельно. С концептуальной точки зрения имеется в виду истинный параллелизм, то есть одновременное выполнение, но в реальной системе это может, как выполняться (если система функционирует на нескольких узлах), так и не выполняться (если система размещена только на одном узле). В последнем случае имеет место последовательное выполнение с переключением между потоками, что дает лишь иллюзию истинного параллелизма.

Из рисунка также видно, что точка слияния представляет собой механизм синхронизации нескольких параллельных потоков выполнения. В эту точку входят два или более перехода, а выходит ровно один. Выше точки слияния деятельности, ассоциированные с приходящими в нее путями, выполняются параллельно. В точке слияния параллельные потоки синхронизируются, то есть каждый из них ждет, пока все остальные достигнут этой точки, после чего выполнение продолжается в рамках одного потока.

Должен поддерживаться баланс между точками разделения и слияния. Это означает, что число потоков, исходящих из точки разделения, должно быть равно числу потоков, приходящих в соответствующую ей точку слияния.

Деятельности, выполняемые в параллельных потоках, могут обмениваться друг с другом информацией, посылая сигналы. Такой способ организация взаимодействия последовательных процессов называется сопрограммами (Coroutine). В большинстве случаев при моделировании такого взаимодействия применяются активные объекты. Но посылку сигналов и получение ответов можно моделировать и с помощью подавтоматов, ассоциированных с каждым из взаимодействующих состояний деятельности.

Дорожки

При моделировании течения бизнес-процессов иногда бывает полезно разбить состояния деятельности на диаграммах деятельности на группы, каждая из которых представляет отдел компании, отвечающий за ту или иную работу. В UML такие группы называются дорожками (Swimlanes), поскольку визуально каждая группа отделяется от соседних вертикальной чертой, как плавательные дорожки в бассейне (см. рис. 2.25).

Дорожки являются разновидностями пакетов, описывающие связанную совокупность работ.

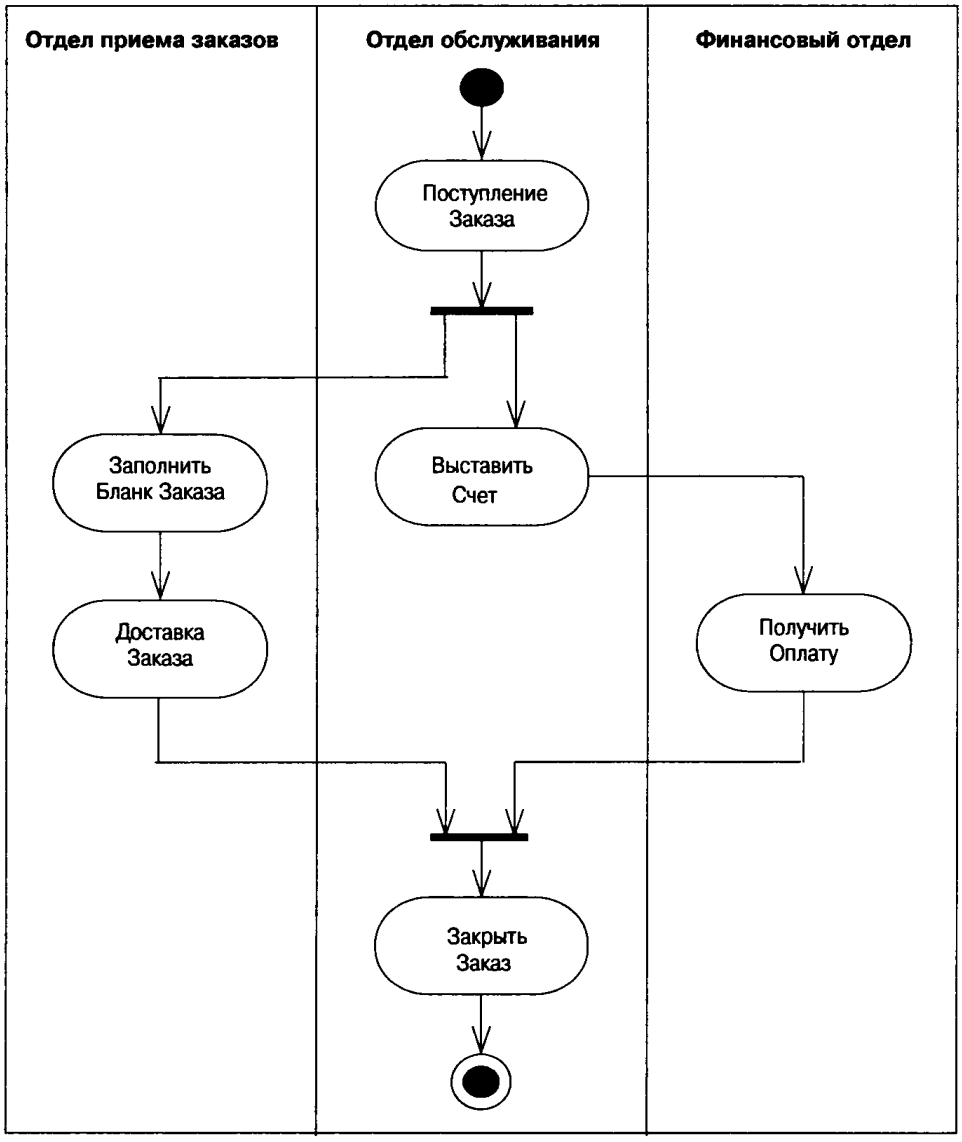

Рис. 2.25 Дорожки

Каждой присутствующей на диаграмме дорожке присваивается уникальное имя. Никакой глубокой семантики дорожка не несет, разве что может отражать некоторую сущность реального мира. Каждая дорожка представляет сферу

ответственности за часть всей работы, изображенной на диаграмме, и, в конечном счете, может быть реализована одним или несколькими классами. На диаграмме деятельности, разбитой на дорожки, каждая деятельность принадлежит ровно одной дорожке, но переходы могут пересекать границы дорожек.

Имеется некоторая связь между дорожками и параллельными потоками выполнения. Концептуально деятельность внутри каждой дорожки обычно рассматривается отдельно от деятельности в соседних дорожках. Это разумно, поскольку в реальном мире подразделения организации, представленные дорожками, как правило, независимы и функционируют параллельно.

Траектория объекта

В потоке управления, ассоциированном с диаграммой деятельности, могут участвовать объекты. К примеру, для последовательности операций по обработке заказа, которая изображена на рис. 2.25, словарь проблемной области будет, вероятно, включать такие классы, как **Заказ** и **Счет**. Некоторые виды деятельности будут порождать объекты-экземпляры этих классов (например, Обработать заказ создаст объект **Заказ**), тогда как другие виды деятельности будут модифицировать эти объекты (например, *Отгрузить заказ* может изменить состояние объекта *Заказ на выполнен*).

Как видно из рис. 2.26, относящиеся к деятельности объекты можно включить в диаграмму деятельности и с помощью символа зависимости привязать к той деятельности или переходу, где они создаются, модифицируются или уничтожаются. Такое сочетание зависимостей и объекта называется траекторией объекта (Object flow), поскольку описывает его участие в потоке управления.

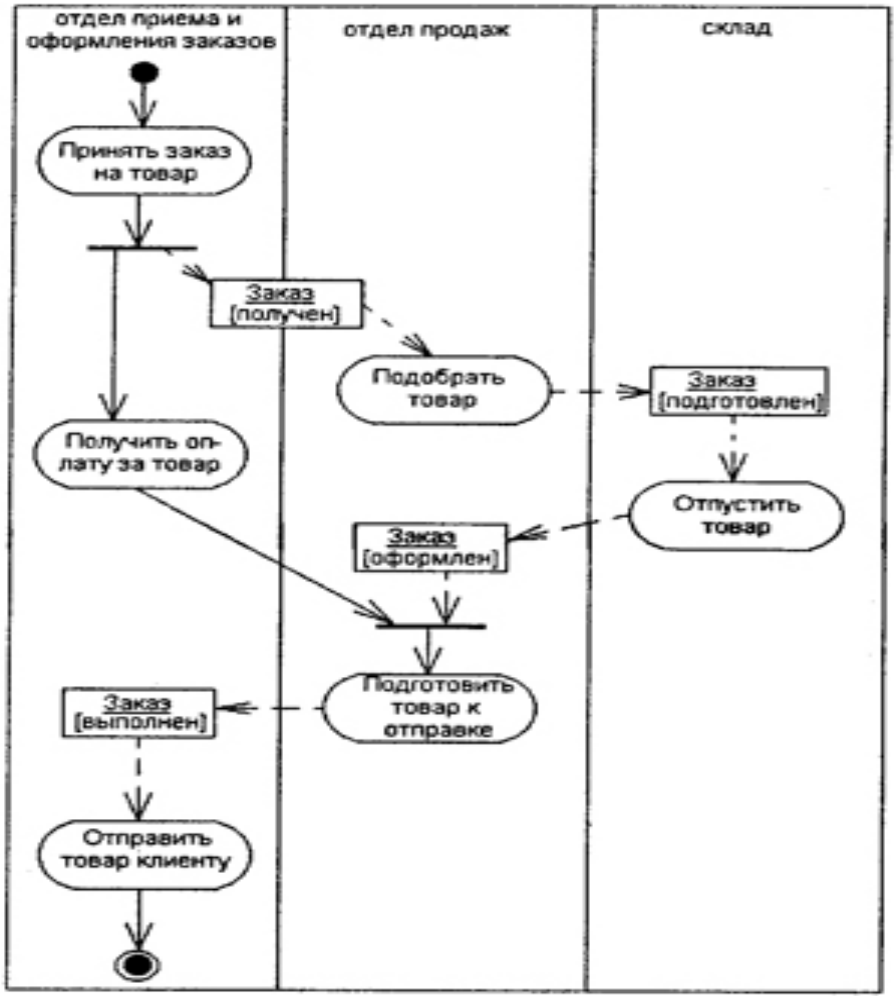

Рис. 2.26 Траектория объекта

Кроме изображения траектории объекта на диаграмме деятельности вы можете показать, как изменяются его роль, состояние и значения атрибутов. Как показано на рисунке, для изображения состояния объекта его имя заключается в скобки и помещается под именем объекта. Аналогично можно представить и значения атрибутов объекта.

Использовать диаграмму деятельности для моделирования некоторого динамического аспекта системы можно в контексте практически любого элемента модели. Но чаще всего они рассматриваются в контексте системы в целом, подсистемы, операции или класса. Можно присоединять диаграммы деятельности к прецедентам и кооперациям (для моделирования динамических аспектов сообщества объектов).

При моделировании динамических аспектов системы диаграммы деятельности применяются в основном двумя способами:

- для моделирования рабочего процесса.

Здесь внимание фокусируется на деятельности с точки зрения актеров, которые сотрудничают с системой. Рабочие процессы часто оказываются с

внешней, обращенной к пользователю стороны программной системы и используются для визуализации, специфицирования, конструирования и документирования бизнес-процессов, составляющих существо разрабатываемой системы. Для такого применения диаграмм деятельности моделирование траекторий объектов имеет особенно важное значение;

- для моделирования операции.

В этом случае диаграммы деятельности используются как блок-схемы для моделирования деталей вычислений. Для такого применения особенно важно моделирование точек ветвления, разделения и слияния. При этом контекст диаграммы деятельности включает параметры операции и ее локальные объекты.

2.5 Базовые понятия теории надежности

Надёжность — свойство объекта сохранять во времени в установленных пределах значения всех параметров, характеризующих способность выполнять требуемые функции в заданных режимах и условиях применения, технического обслуживания, хранения и транспортирования.

Интуитивно надёжность объектов связывают с недопустимостью отказов в работе. Это есть понимание надёжности в «узком» смысле — свойство объекта сохранять работоспособное состояние в течение некоторого времени или некоторой наработки. Иначе говоря, надёжность объекта заключается в отсутствии непредвиденных недопустимых изменений его качества в процессе эксплуатации и хранения. Надёжность тесно связана с различными сторонами процесса эксплуатации. Надёжность в «широком» смысле — комплексное свойство, которое в зависимости от назначения объекта и условий его эксплуатации может включать в себя свойства безотказности, долговечности, ремонтопригодности и сохраняемости, а также определённое сочетание этих свойств.

Для количественной оценки надёжности используют так называемые единичные показатели надёжности (характеризуют только одно свойство надёжности) и комплексные показатели надёжности (характеризуют несколько свойств надёжности).

Безотказность — свойство объекта непрерывно сохранять работоспособное состояние в течение некоторого времени или наработки.

Ремонтопригодность — свойство объекта, заключающееся в приспособленности к поддержанию и восстановлению работоспособного состояния путем технического обслуживания и ремонта.

Долговечность — свойство объекта непрерывно сохранять работоспособность от начала эксплуатации до наступления предельного состояния, то есть такого состояния, когда объект изымается из эксплуатации.

Сохраняемость — свойство объекта сохранять работоспособность в течение всего периода хранения и транспортировки.

Живучесть — свойство объекта сохранять работоспособность при отказе отдельных функциональных узлов.

Отказ — событие, заключающееся в полной или частичной утрате работоспособности.

Сбой — самоустраняющийся отказ или однократный отказ, устраняемый незначительным вмешательством оператора.[2]

Наработка — время или объём работы.[3]

Ресурс — наработка от начала эксплуатации до наступления предельного состояния.

Срок службы — календарная продолжительность от начала эксплуатации до наступления предельного состояния.

Надежность на этапе проектирования

Надежность на этапе проектирования является новой дисциплиной и относится к процессу разработки надежных изделий. Этот процесс включает в себя несколько инструментов и практических рекомендаций и описывает порядок их применения, которыми должна владеть организация для обеспечения высокой надежности и ремонтопригодности разрабатываемого продукта с целью достижения высоких показателей готовности, снижения затрат и максимального срока службы продукта. Как правило, первым шагом в этом направлении является нормирование показателей надежности.

Надежность должна быть «спроектирована» в системе. При проектировании системы назначаются требования к надежности верхнего уровня, затем они разделяются на определенные подсистемы разработчиками, конструкторами и инженерами по надежности, работающими вместе. Проектирование надежности начинается с разработки модели. При этом используют структурные схемы надежности или деревья неисправностей, при помощи которых представляется взаимоотношение между различными частями (компонентами) системы.

Одной из наиболее важных технологий проектирования является введение избыточности или резервирование.

Резервирование – это способ обеспечения надежности изделия за счет дополнительных средств и (или) возможностей, избыточных по отношению к минимально необходимым для выполнения требуемых функций (ГОСТ 27.002). Путем введения избыточности совместно с хорошо организованным мониторингом отказов, даже системы с низкой надежностью по одному каналу могут в целом обладать высоким уровнем надежности. Однако введение избыточности на высоком уровне в сложной системе (например, на уровне двигателя самолета) очень сложно и дорого, что ограничивает такое резервирование.

2.6 Методы расчета надежности[6]

Расчёт надёжности — это процедура определения значений показателей надежности объекта с использованием методов, основанных на их вычислении по справочным данным о надежности элементов объекта, по данным о надежности объектов-аналогов, данным о свойствах материалов и другой информации, имеющейся к моменту расчета.

Решение вопросов надежности и безопасности современных структурно-сложных технических систем и объектов осуществляется на всех стадиях жизненного цикла, от проектирования и создания, производства, до эксплуатации, использования и утилизации. При этом могут преследоваться следующие цели:

- обоснование количественных требований к надежности объекта или его составным частям;
- сравнительный анализ надежности вариантов схемно-конструктивного построения объекта и обоснование выбора рационального варианта, в том числе по стоимостному критерию;
- определение достигнутого (ожидаемого) уровня надежности объекта и/или его составных частей, в том числе расчетное определение показателей надежности или параметров распределения характеристик надежности составных частей объекта в качестве исходных данных для расчета надежности объекта в целом;
- обоснование и проверку эффективности предлагаемых (реализованных) мер по доработкам конструкции, технологии изготовления, системы технического обслуживания и ремонта объекта, направленных на повышение его надежности;
- решение различных оптимизационных задач, в которых показатели надежности выступают в роли целевых функций, управляемых параметров или граничных условий, в том числе таких, как оптимизация структуры объекта, распределение требований по надежности между показателями отдельных составляющих надежности (например, безотказности и ремонтопригодности), расчет комплектов ЗИП, оптимизация систем технического обслуживания и ремонта, обоснование гарантийных сроков и назначенных сроков службы (ресурса) объекта и др.;
- проверку соответствия ожидаемого (достигнутого) уровня надежности объекта установленным требованиям (контроль надежности), если прямое экспериментальное подтверждение их уровня надежности невозможно технически или нецелесообразно экономически.

На этапе проектирования расчёт надёжности проводится с целью прогнозирования надёжности работы проектируемой системы.

На этапе испытаний и эксплуатации расчёт надёжности проводится для оценки количественных показателей надёжности спроектированной системы.

[6] Лабораторная работа 3. Расчет надежности систем и исследование резервирования

В результате расчета определяются количественные значения показателей надёжности.

Для расчета надёжности используют следующие методы:

1. Структурные методы расчета надежности

Структурные методы являются основными методами расчета показателей надежности в процессе проектирования объектов, поддающихся разукрупнению на элементы, характеристики надежности, которых в момент проведения расчетов известны или могут быть определены другими методами.

Расчет показателей надежности структурными методами в общем случае включает:

– представление объекта в виде структурной схемы, описывающей логические соотношения между состояниями элементов и объекта в целом с учетом структурно-функциональных связей и взаимодействия элементов, принятой стратегии обслуживания, видов и способов резервирования и других факторов;

– описание построенной структурной схемы надежности объекта адекватной математической моделью, позволяющей в рамках введенных предположений и допущений вычислить показатели надежности объекта по данным о надежности его элементов в рассматриваемых условиях применения.

В качестве структурных схем надежности могут применяться:

- схемы функциональной целостности;
- структурные блок-схемы надежности;
- деревья отказов;
- графы состояний и переходов.

2. Логико-вероятностный метод

В логико-вероятностных методах (ЛВМ) исходная постановка задачи и построение модели функционирования исследуемого системного объекта или процесса осуществляется структурными и аналитическими средствами математической логики, а расчет показателей свойств надежности, живучести и безопасности выполняется средствами теории вероятностей.

ЛВМ являются методологией анализа структурно-сложных систем, решения системных задач организованной сложности, оценки и анализа надежности, безопасности и риска технических систем. ЛВМ удобны для исходной формализованной постановки задач в форме структурного описания исследуемых свойств функционирования сложных и высокоразмерных систем. В ЛВМ разработаны процедуры преобразования исходных структурных моделей в искомые расчетные математические модели, что позволяет выполнить их алгоритмизацию и реализацию на ЭВМ.

Основоположником научно-технического аппарата ЛВМ и прикладных аспектов их применения, а также создателем и руководителем научной школы является профессор Рябинин И.А..

3. Общий логико-вероятностный метод

Необходимость распространения ЛВМ на немонотонные процессы привела к созданию общего логико-вероятностного метода (ОЛВМ). В ОЛВМ расчета надежности аппарат математической логики используется для первичного графического и аналитического описания условий реализации функций отдельными и группами элементов в проектируемой системе, а методы теории вероятностей и комбинаторики применяются для количественной оценки безотказности и/или опасности функционирования проектируемой системы в целом. Для использования ОЛВМ должны задаваться специальные структурные схемы функциональной целостности исследуемых систем, логические критерии их функционирования, вероятностные и другие параметры элементов.

В основе постановки и решения всех задач моделирования и расчета надежности систем с помощью ОЛВМ лежит так называемый событийно-логический подход. Этот подход предусматривает последовательное выполнение следующих четырех основных этапов ОЛВМ:

- этап структурно-логической постановки задачи;
- этап логического моделирования;
- этап вероятностного моделирования;
- этап выполнения расчетов показателей надежности.

Существует много методик анализа надежности, специфических для отдельных отраслей промышленности и приложений. Наиболее общие из них следующие.

Анализ видов и последствий отказов (АВПО).

Имитационное моделирование надежности.

Анализ схем функциональной целостности (СФЦ).

Анализ опасностей (Hazard analysis).

Анализ структурных схем надежности (RBD).

Анализ деревьев неисправностей.

Ускоренные испытания.

Модели ускорения жизни.

Модели деградации.

Анализ роста надежности.

Вейбулл-анализ (анализ эмпирических данных испытаний и эксплуатации).

Анализ смеси распределений.

Устранение критичных отказов.

Анализ ремонтопригодности, ориентированной на безотказность.

Анализ диагностики отказов.

Анализ ошибок человека-оператора.

3. Построение корпоративных инфокоммуникационных систем

3.1 Функциональная методика IDEF [7]

Методологию IDEF0 можно считать следующим этапом развития графического языка описания функциональных систем SADT (Structured Analysis and Design Teqnique).

Исторически IDEF0 как стандарт был разработан в 1981 году в рамках обширной программы автоматизации промышленных предприятий, которая носила обозначение ICAM (Integrated Computer Aided Manufacturing). Семейство стандартов IDEF унаследовало свое обозначение от названия этой программы (IDEF=Icam DEFinition), и последняя его редакция была выпущена в декабре 1993 года Национальным Институтом по Стандартам и Технологиям США (NIST).

Целью методики является построение функциональной схемы исследуемой системы, описывающей все необходимые процессы с точностью, достаточной для однозначного моделирования деятельности системы.

В основе методологии лежат четыре основных понятия: **функциональный блок, интерфейсная дуга, декомпозиция, глоссарий**.[8]

Функциональный блок (Activity Box) представляет собой некоторую конкретную *функцию* в рамках рассматриваемой системы. По требованиям стандарта название каждого функционального блока должно быть сформулировано в глагольном наклонении (например, «производить услуги»). На диаграмме функциональный блок изображается прямоугольником. Каждая из четырех сторон функционального блока имеет свое определенное значение (роль), при этом:

верхняя сторона имеет значение «Управление» (Control);

левая сторона имеет значение «Вход» (Input);

правая сторона имеет значение «Выход» (Output);

нижняя сторона имеет значение «Механизм» (Mechanism).

Функциональный блок **Интерфейсная дуга** (Arrow) отображает элемент системы, который обрабатывается функциональным блоком или оказывает иное влияние на *функцию*, представленную данным функциональным блоком. Интерфейсные дуги часто называют потоками или стрелками.

С помощью интерфейсных дуг отображают различные объекты, в той или иной степени определяющие процессы, происходящие в системе. Такими объектами могут быть элементы реального мира (детали, вагоны, сотрудники и т.д.) или потоки данных и информации (документы, данные, инструкции и т.д.).

В зависимости от того, к какой из сторон функционального блока подходит данная интерфейсная дуга, она носит название «входящей», «исходящей» или «управляющей».

---

[7] Практическое занятие 4. Стандарты семейства IDEF

[8] Лабораторная работа 4. Разработка проекта корпоративной инфокоммуникационной системы

Необходимо отметить, что любой функциональный блок по требованиям стандарта должен иметь, по крайней мере, одну управляющую интерфейсную дугу и одну исходящую. Это и понятно – каждый процесс должен происходить по каким-то правилам (отображаемым управляющей дугой) и должен выдавать некоторый результат (выходящая дуга), иначе его рассмотрение не имеет никакого смысла.

Обязательное наличие управляющих интерфейсных дуг является одним из главных отличий стандарта IDEF0 от других методологий классов DFD (Data Flow Diagram) и WFD (Work Flow Diagram).

Декомпозиция (Decomposition) является основным понятием стандарта IDEF0. Принцип декомпозиции применяется при разбиении сложного процесса на составляющие его *функции*. При этом уровень детализации процесса определяется непосредственно разработчиком модели.

Декомпозиция позволяет постепенно и структурированно представлять модель системы в виде иерархической структуры отдельных диаграмм, что делает ее менее перегруженной и легко усваиваемой.

Последним из понятий IDEF0 является глоссарий (Glossary). Для каждого из элементов IDEF0 — диаграмм, функциональных блоков, интерфейсных дуг — существующий стандарт подразумевает создание и поддержание набора соответствующих определений, ключевых слов, повествовательных изложений и т.д., которые характеризуют объект, отображенный данным элементом. Этот набор называется глоссарием и является описанием сущности данного элемента. Глоссарий гармонично дополняет наглядный графический язык, снабжая диаграммы необходимой дополнительной информацией.

Модель IDEF0 всегда начинается с представления системы как единого целого – одного функционального блока с интерфейсными дугами, простирающимися за пределы рассматриваемой области. Такая диаграмма с одним функциональным блоком называется **контекстной диаграммой**.

В пояснительном тексте к контекстной диаграмме должна быть указана **цель** (Purpose) построения диаграммы в виде краткого описания и зафиксирована **точка зрения** (Viewpoint).

Определение и формализация цели разработки IDEF0-модели является крайне важным моментом. Фактически цель определяет соответствующие области в исследуемой системе, на которых необходимо фокусироваться в первую очередь.

Точка зрения определяет основное направление развития модели и уровень необходимой детализации. Четкое фиксирование точки зрения позволяет разгрузить модель, отказавшись от детализации и исследования отдельных элементов, не являющихся необходимыми, исходя из выбранной точки зрения на систему. Правильный выбор точки зрения существенно сокращает временные затраты на построение конечной модели.

Выделение *подпроцессов*. В процессе декомпозиции функциональный блок, который в контекстной диаграмме отображает систему как единое целое, подвергается детализации на другой диаграмме. Получившаяся диаграмма второго уровня содержит функциональные блоки, отображающие главные подфункции функционального блока контекстной диаграммы, и называется дочерней (Child Diagram) по отношению к нему (каждый из функциональных блоков, принадлежащих дочерней диаграмме, соответственно называется дочерним блоком – Child Box). В свою очередь, функциональный блок — предок называется родительским блоком по отношению к дочерней диаграмме (Parent Box), а диаграмма, к которой он принадлежит – родительской диаграммой (Parent Diagram). Каждая из подфункций дочерней диаграммы может быть далее детализирована путем аналогичной декомпозиции соответствующего ей функционального блока. В каждом случае декомпозиции функционального блока все интерфейсные дуги, входящие в данный блок или исходящие из него, фиксируются на дочерней диаграмме. Этим достигается структурная целостность IDEF0–модели.

Иногда отдельные интерфейсные дуги высшего уровня не имеет смысла продолжать рассматривать на диаграммах нижнего уровня, или наоборот — отдельные дуги нижнего отражать на диаграммах более высоких уровней – это будет только перегружать диаграммы и делать их сложными для восприятия. Для решения подобных задач в стандарте IDEF0 предусмотрено понятие туннелирования. Обозначение «туннеля» (Arrow Tunnel) в виде двух круглых скобок вокруг начала интерфейсной дуги обозначает, что эта дуга не была унаследована от функционального родительского блока и появилась (из «туннеля») только на этой диаграмме. В свою очередь, такое же обозначение вокруг конца (стрелки) интерфейсной дуги в непосредственной близи от блока–приемника означает тот факт, что в дочерней по отношению к этому блоку диаграмме эта дуга отображаться и рассматриваться не будет. Чаще всего бывает, что отдельные объекты и соответствующие им интерфейсные дуги не рассматриваются на некоторых промежуточных уровнях иерархии, – в таком случае они сначала «погружаются в туннель», а затем при необходимости «возвращаются из туннеля».

Обычно IDEF0-модели несут в себе сложную и концентрированную информацию, и для того, чтобы ограничить их перегруженность и сделать удобочитаемыми, в стандарте приняты соответствующие ограничения сложности.

Рекомендуется представлять на диаграмме от трех до шести функциональных блоков, при этом количество подходящих к одному функциональному блоку (выходящих из одного функционального блока) интерфейсных дуг предполагается не более четырех.

Стандарт IDEF0 содержит набор процедур, позволяющих разрабатывать и согласовывать модель большой группой людей, принадлежащих к разным

областям деятельности моделируемой системы. Обычно процесс разработки является итеративным и состоит из следующих условных этапов:

- Создание модели группой специалистов, относящихся к различным сферам деятельности предприятия. Эта группа в терминах IDEF0 называется авторами (Authors). Построение первоначальной модели является динамическим процессом, в течение которого авторы опрашивают компетентных лиц о структуре различных процессов, создавая модели деятельности подразделений. При этом их интересуют ответы на следующие вопросы:

  Что поступает в подразделение «на входе»?

  - Какие *функции* и в какой последовательности выполняются в рамках подразделения?
  - Кто является ответственным за выполнение каждой из *функций*?
  - Чем руководствуется исполнитель при выполнении каждой из *функций*?
  - Что является результатом работы подразделения (на выходе)?

На основе имеющихся положений, документов и результатов опросов создается черновик (Model Draft) модели.

- Распространение черновика для рассмотрения, согласований и комментариев. На этой стадии происходит обсуждение черновика модели с широким кругом компетентных лиц (в терминах IDEF0 — читателей) на предприятии. При этом каждая из диаграмм черновой модели письменно критикуется и комментируется, а затем передается автору. Автор, в свою очередь, также письменно соглашается с критикой или отвергает ее с изложением логики принятия решения и вновь возвращает откорректированный черновик для дальнейшего рассмотрения. Этот цикл продолжается до тех пор, пока авторы и читатели не придут к единому мнению.
- Официальное утверждение модели. Утверждение согласованной модели происходит руководителем рабочей группы в том случае, если у авторов модели и читателей отсутствуют разногласия по поводу ее адекватности. Окончательная модель представляет собой согласованное представление о предприятии (системе) с заданной точки зрения и для заданной цели.

Наглядность графического языка IDEF0 делает модель вполне читаемой и для лиц, которые не принимали участия в проекте ее создания, а также эффективной для проведения показов и презентаций. В дальнейшем на базе построенной модели могут быть организованы новые проекты, нацеленные на производство изменений в модели.

3.2 Функциональная методика потоков данных [9]

Целью методики является построение модели рассматриваемой системы в виде диаграммы потоков данных (Data Flow Diagram — DFD), обеспечивающей правильное описание выходов (отклика системы в виде данных) при заданном воздействии на вход системы (подаче сигналов через внешние интерфейсы). Диаграммы потоков данных являются основным средством моделирования функциональных требований к проектируемой системе.

При создании диаграммы потоков данных используются четыре основных понятия: потоки данных, процессы (работы) преобразования входных потоков данных в выходные, внешние сущности, накопители данных (хранилища).

Потоки данных являются абстракциями, использующимися для моделирования передачи информации (или физических компонент) из одной части системы в другую. Потоки на диаграммах изображаются именованными стрелками, ориентация которых указывает направление движения информации.

Назначение **процесса** (работы) состоит в продуцировании выходных потоков из входных в соответствии с действием, задаваемым именем процесса. Имя процесса должно содержать глагол в неопределенной форме с последующим дополнением (например, «получить документы по отгрузке продукции»). Каждый процесс имеет уникальный номер для ссылок на него внутри диаграммы, который может использоваться совместно с номером диаграммы для получения уникального индекса процесса во всей модели.

Хранилище (накопитель) данных позволяет на указанных участках определять данные, которые будут сохраняться в памяти между процессами. Фактически хранилище представляет «срезы» потоков данных во времени. Информация, которую оно содержит, может использоваться в любое время после ее получения, при этом данные могут выбираться в любом порядке. Имя хранилища должно определять его содержимое и быть существительным.

Внешняя сущность представляет собой материальный объект вне контекста системы, являющейся источником или приемником системных данных. Ее имя должно содержать существительное, например, «склад товаров». Предполагается, что объекты, представленные как внешние сущности, не должны участвовать ни в какой обработке.

Кроме основных элементов, в состав DFD входят словари данных и миниспецификации.

Словари данных являются каталогами всех элементов данных, присутствующих в DFD, включая групповые и индивидуальные потоки данных, хранилища и процессы, а также все их атрибуты.

Миниспецификации обработки — описывают DFD-процессы нижнего уровня. Фактически миниспецификации представляют собой алгоритмы описания

[9] Лабораторная работа 6. Проектирование потоков данных корпоративной инфокоммуникационной системы

задач, выполняемых процессами: множество всех миниспецификаций является полной спецификацией системы.

Процесс построения DFD начинается с создания так называемой основной диаграммы типа «звезда», на которой представлен моделируемый процесс и все внешние сущности, с которыми он взаимодействует. В случае сложного основного процесса он сразу представляется в виде декомпозиции на ряд взаимодействующих процессов. Критериями сложности в данном случае являются: наличие большого числа внешних сущностей, многофункциональность системы, ее распределенный характер. Внешние сущности выделяются по отношению к основному процессу. Для их определения необходимо выделить поставщиков и потребителей основного процесса, т.е. все объекты, которые взаимодействуют с основным процессом. На этом этапе описание взаимодействия заключается в выборе глагола, дающего представление о том, как внешняя сущность использует основной процесс или используется им. Например, основной процесс – «учет обращений граждан», внешняя сущность – «граждане», описание взаимодействия – «подает заявления и получает ответы». Этот этап является принципиально важным, поскольку именно он определяет границы моделируемой системы.

Для всех внешних сущностей строится таблица событий, описывающая их взаимодействие с основным потоком. Таблица событий включает в себя наименование внешней сущности, событие, его тип (типичный для системы или исключительный, реализующийся при определенных условиях) и реакцию системы.

На следующем шаге происходит декомпозиция основного процесса на набор взаимосвязанных процессов, обменивающихся потоками данных. Сами потоки не конкретизируются, определяется лишь характер взаимодействия. Декомпозиция завершается, когда процесс становится простым, т.е.:

1. процесс имеет два-три входных и выходных потока;
2. процесс может быть описан в виде преобразования входных данных в выходные;
3. процесс может быть описан в виде последовательного алгоритма.

Для простых процессов строится миниспецификация – формальное описание алгоритма преобразования входных данных в выходные.

Миниспецификация удовлетворяет следующим требованиям: для каждого процесса строится одна спецификация; спецификация однозначно определяет входные и выходные потоки для данного процесса; спецификация не определяет способ преобразования входных потоков в выходные; спецификация ссылается на имеющиеся элементы, не вводя новые; спецификация по возможности использует стандартные подходы и *операции*.

После декомпозиции основного процесса для каждого *подпроцесса* строится аналогичная таблица внутренних событий.

Следующим шагом после определения полной таблицы событий выделяются **потоки данных**, которыми обмениваются процессы и внешние сущности. Простейший способ их выделения заключается в анализе таблиц событий. События преобразуются в потоки данных от инициатора события к запрашиваемому процессу, а реакции – в обратный поток событий. После построения входных и выходных потоков аналогичным образом строятся внутренние потоки. Для их выделения для каждого из внутренних процессов выделяются поставщики и потребители информации. Если поставщик или потребитель информации представляет процесс сохранения или запроса информации, то вводится хранилище данных, для которого данный процесс является интерфейсом.

После построения потоков данных диаграмма должна быть проверена на полноту и непротиворечивость. Полнота диаграммы обеспечивается, если в системе нет «повисших» процессов, не используемых в процессе преобразования входных потоков в выходные. Непротиворечивость системы обеспечивается выполнением наборов формальных правил о возможных типах процессов: на диаграмме не может быть потока, связывающего две внешние сущности – это взаимодействие удаляется из рассмотрения; ни одна сущность не может непосредственно получать или отдавать информацию в хранилище данных – хранилище данных является пассивным элементом, управляемым с помощью интерфейсного процесса; два хранилища данных не могут непосредственно обмениваться информацией – эти хранилища должны быть объединены.

К преимуществам методики DFD относятся:

- возможность однозначно определить внешние сущности, анализируя потоки информации внутри и вне системы;
- возможность проектирования сверху вниз, что облегчает построение модели «как должно быть»;
- наличие спецификаций процессов нижнего уровня, что позволяет преодолеть логическую незавершенность функциональной модели и построить полную функциональную спецификацию разрабатываемой системы.

К недостаткам модели отнесем: необходимость искусственного ввода управляющих процессов, поскольку управляющие воздействия (потоки) и управляющие процессы с точки зрения DFD ничем не отличаются от обычных; отсутствие понятия времени, т.е. отсутствие анализа временных промежутков при преобразовании данных (все ограничения по времени должны быть введены в спецификациях процессов).

3.3 CASE–методология и инструментальные средства проектирования[10]

Процесс внедрения CASE-средств состоит из следующих этапов:

- 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.

Модель процесса оценки и выбора, рассматриваемая ниже (рис. 3.1), 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.

[10] Лабораторная работа 5. Создание функциональной модели с помощью CASE-средств

Рис. 3.1. 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.

3.4 MRP-системы

MRP (*Material Requirements Planning* — планирование потребности в материалах) — система планирования потребностей в материалах, одна из наиболее популярных в мире логистических концепций, на основе которой разработано и функционирует большое число микрологистических систем.

На концепции MRP базируется построение логистических систем «толкающего типа». В России, как правило, представлена различными программными продуктами иностранного производства. Появление более развитой концепции MRP II и развитие программ класса ERP, снижение их стоимости, привело к тому, что программные продукты класса MRP можно встретить очень редко, как правило, в составе устаревших информационных систем предприятий.

Один из главных разработчиков MRP Дж. Орлиски писал: «планирование потребностей в материалах в узком смысле состоит из ряда логически связанных процедур, решающих правил и требований, переводящих производственное расписание в „цепочку требований“, синхронизированных во времени, и запланированных „покрытий“ этих требований для каждой единицы запаса компонентов, необходимых для выполнения производственного расписания. MRP-система перепланирует последовательность требований и покрытий в результате

изменений либо в производственном расписании, либо в структуре запасов, либо в атрибутах продукта».

MRP-система применяется при работе с материалами, компонентами, полуфабрикатами и их частями, спрос, на который зависит от спроса на специфическую готовую продукцию, то есть спрос на исходные материальные ресурсы сильно зависит от спроса потребителей на конечную продукцию. Также MRP-система может работать с широкой номенклатурой материальных ресурсов.

Основные цели MRP

- удовлетворение потребности в материалах, компонентах и продукции для планирования производства и доставки потребителям;
- поддержка низких уровней запасов;
- планирование производственных операций, расписаний доставки, закупочных операций.

Система MRP позволяет определить сколько, и в какие сроки необходимо произвести конечной продукции. Затем система определяет время и необходимые количества материальных ресурсов для удовлетворения потребностей производственного расписания

Структура MRP системы

1. Материалы - все сырье и отдельные комплектующие, составляющие конечный продукт. В дальнейшем мы не будем делать различий между понятиями «материал» и «комплектующий».
2. MRP-система, MRP-программа - компьютерная программа, работающая по MRP алгоритму.
3. Статус материала является основным указателем на текущее состояние материала. Каждый отдельный материал, в каждый момент времени, имеет статус в рамках MRP-системы, например:
   - материал есть в наличии на складе,
   - материал есть на складе, но зарезервирован для других целей
   - материал присутствует в текущих заказах
   - заказ на материал планируется

Как видно, статус материала отражает степень готовности этого материала быть пущенным в производственный процесс.

4. Страховой запас (safety stock) материала необходим для поддержания процесса производства в случае возникновения непредвиденных и неустранимых задержек в его поставках. По сути, в идеальном случае, если механизм поставок полагать безупречным, MRP-методология не постулирует обязательное наличие страхового запаса, и его объемы устанавливаются различными для каждого конкретного случая, в зависимости от сложившейся ситуации с поступлением материалов.
5. Потребность в материале в MRP-программе представляет собой определенную количественную единицу, отображающую возникшую в

некоторой момент времени в течение периода планирования необходимость в заказе данного материала.

Различают понятия полной потребности в материале, которая отображает то количество, которое требуется пустить в производство, и чистой потребности, при вычислении которой учитывается наличие всех страховых и зарезервированных запасов данного материала. Заказ в системе автоматически создается по возникновению отличной от нуля чистой потребности.

Формула вычисления чистой потребности такова:

Чистая потребность = полная потребность - инвентаризовано на руках - страховой запас - зарезервировано для других заказов

MRP-система как черный ящик

Основные элементы MRP системы можно разделить на элементы, предоставляющие информацию, программная реализация алгоритмической основы MRP и элементы, представляющие результат функционирования программной реализации MRP.

На рис. 3.2 показаны входные и выходные параметры для MRP-системы.

Рис. 3.2 Входные и выходные параметры для MRP-системы

Входные данные:

Программа производства (Основной Производственный План-график (ОПП), Master Production Schedule (MPS))

Основной производственный план, как правило, формируется для пополнения запаса готовой продукции или удовлетворения заказов потребителей.

На практике разработка ОПП представляется петлей планирования. Первоначально формируется черновой вариант для оценки возможности обеспечения реализации по материальным ресурсам и мощностям.

Система MRP осуществляет детализацию ОПП в разрезе материальных составляющих. Если необходимая номенклатура и ее количественный состав не присутствует в свободном или заказанном ранее запасе или в случае

неудовлетворительных по времени планируемых поставок материалов и комплектующих, ОПП должен быть соответствующим образом скорректирован.

После проведения необходимых итераций ОПП утверждается как действующий и на его основе осуществляется запуск производственных заказов.

Перечень составляющих конечного продукта (Ведомость материалов и состав изделия (BM), Bill Of Materials (BOM))

Ведомость материалов (BM) представляет собой номенклатурный перечень материалов и их количества для производства некоторого узла или конечного изделия. Совместно с составом изделия BM обеспечивает формирование полного перечня готовой продукции, количества материалов и комплектующих для каждого изделия и описание структуры изделия (узлы, детали, комплектующие, материалы и их взаимосвязи).

Ведомость материалов и состав изделия представляют собой таблицы базы данных, информация которых корректно отражает соответствующие данные, при изменении физического состава изделия или BM состояние таблиц должно быть своевременно скорректировано.

Описание состояния материалов (Состояние запасов, Stock/Requirement List)

Текущее состояние запасов отражается в соответствующих таблицах базы данных с указанием всех необходимых характеристик учетных единиц. Каждая учетная единица, вне зависимости от вариантов ее использования в одном изделии или многих готовых изделиях должна иметь только одну идентифицирующую запись с уникальным кодом. Как правило, идентификационная запись учетной единицы содержит большое количество параметров и характеристик, используемых MRP системой, которые можно классифицировать следующим образом:

1. Общие данные: код, описание, тип, размер, вес.
2. Данные запаса: единица запаса, единица хранения, свободный запас, оптимальный запас, запланированный к заказу, заказанный запас, распределенный запас, признак партии/серии.
3. Данные по закупкам и продажам: единица закупки/продажи, основной поставщик, цена.
4. Данные по производству и производственным заказам и т.д.

Записи учетных единиц обновляются всякий раз при выполнении операций с запасами, например, запланированные к закупке, заказанные к поставке, оприходованные, брак и т.д.

Основные операции

На основании входных данных MRP система выполняет следующие основные операции:

1. На основании ОПП определяется количественный состав конечных изделий для каждого периода времени планирования

2. К составу конечных изделий добавляются запасные частей, не включенных в ОПП
3. Для ОПП и запасных частей определяется общая потребность в материальных ресурсах в соответствии с ВМ и составом изделия с распределением по периодам времени планирования
4. Общая потребность материалов корректируется с учетом состояния запасов для каждого периода времени планирования
5. Осуществляется формирование заказов на пополнение запасов с учетом необходимых времен опережения

Выходные данные

Результатами работы MRP системы являются:

1. План-график снабжения материальными ресурсами производства - количество каждой учетной единицы материалов и комплектующих для каждого периода времени для обеспечения ОПП.

Для реализации плана-графика снабжения система порождает план-график заказов в привязке к периодам времени, который используется для размещения заказов поставщикам материалов и комплектующих или для планирования самостоятельного изготовления

1. Изменения плана-графика снабжения - внесение корректировок в ранее сформированный план-график снабжения производства
2. Ряд отчетов, необходимых для управления процессом снабжения производства

3.5 MRP II

Метод MRP следует двум важнейшим принципам:

1. Логике «зависимого спроса», т.е. Если есть потребность в конечном изделии, значит есть потребность во всех его компонентах;
2. Обеспечивать требуемые компоненты как можно позднее, чтобы уровень запасов был минимальным.

Чтобы следовать этим двум принципам, системе требуется большой объем информации. Для расчета потребностей в компонентах нижнего уровня требуется «спецификация» на каждое конечное изделие - по которой определяются компоненты, время начала и завершения работ, этапы производства - и данные о «состоянии запасов» - чтобы определить, сколько требуемых компонентов имеется в запасе и в незавершенном производстве. В результате автоматизированных вычислений очень быстро формируется план потребностей. Этот план потребностей является стержнем в системах MRPII.

На планирование потребностей в материалах влияет точность спецификаций и записей о состоянии запасов. Допущенная ошибка может привести к тому, что будет вычислено неправильное количество или заказаны не те компоненты; эта ошибка не может быть исправлена до тех пор, пока не будет обнаружена

физически, и часто на это уходит несколько недель. Надежность и быстродействие ранних систем означали, что на прогон системы уходило очень много времени: от 24-х до 48 часов. Поэтому прогоны делались нечасто, и было невозможно проверять выполнимость основного плана производства посредством повторных прогонов алгоритма MRP. Поэтому основной план часто не выполнялся и устаревал.

Также было невозможно быстро корректировать данные или отражать в плане изменения, каждодневно возникающие на складах и на производстве. Обычно в результате этого появлялось существенное отличие между формально принятым планом потребностей и неформально действующими листками «дефицита», подгоняющими выполнение плана. Решения, предлагаемые системой MRP, часто игнорировались, в то время, как заказы на работы нагромождались друг на друга на одном конце предприятия и в конечном итоге вытягивались и отгружались заказчику на другом конце, после того, как получали достаточно высокий приоритет, задерживая при этом все другие изделия. Неудивительно, что первые внедрения получили нелестную оценку.

Изобретение менее дорогостоящих вычислительных систем реального времени и опыт работы с MRPI привели к разработке в конце 70-х годов систем MRP в замкнутом цикле, которые нашли в настоящее время широкое применение.

Термин замкнутый цикл означает, что функционирование системы происходит с учетом обратной связи от одной функции к другой. Здесь уже другие требования к планированию материалов. Информация передается обратно через вычислительную систему, но при этом никакие действия не предпринимаются. Принятие решения о корректировке плана остается за человеком.

В системе планирования по замкнутому циклу важное значение отводится контролю за ходом выполнения, чтобы, планы на будущее соответствовали тому, что происходило до этого на самом деле.

В 80-х годах принципы MRP в замкнутом цикле были распространены за пределы управления материалами. Планирование производственных ресурсов предполагает планирование всех ресурсов, включая оборудование, людские ресурсы, материальные запасы и денежные средства. Данный метод позволяет воспользоваться преимуществами одной системы всем службам предприятия от отдела сбыта до службы маркетинга, отдела снабжения, финансового отдела, конструкторского отдела, а также на производстве.

Ключевыми возможностями систем MRPII являются обратная связь по фактическому состоянию производства и заказов на закупку, более тщательная проверка выполнимости основного плана производства и внесение изменений в производственный план посредством приблизительного планирования мощности, анализа «что-если» и выполнения алгоритма MRP с учетом частых изменений. MRPII становится главной частью любой интегрированной вычислительной системы на производственных предприятиях.

Таким образом, MRPII системы объединяют процедуры обработки заказов на продажу, бухгалтерского учета, закупок и выписки счетов-фактур с производством на основе одной базы данных реального времени.

В то же время, MRPII системы не контролируют конструкторские разработки, составление сметы, кадры, сбыт и распределение продукции, обслуживание, т.е. подразделения не объединены в одну систему. Этот круг вопросов рассматривался разработчиками систем в 90-х годах, чтобы обеспечить полностью интегрированные системы для управления производственными предприятиями, в основе которых были заложены принципы MRPII, и был реализован в системах ERP.

MRPII-система должна состоять из следующих функциональных модулей (см. рис.3.1):

1. Планирование развития бизнеса (Составление и корректировка бизнес-плана)
2. Планирование деятельности предприятия
3. Планирование продаж
4. Планирование потребностей в сырье и материалах
5. Планирование производственных мощностей
6. Планирование закупок
7. Выполнение плана производственных мощностей
8. Выполнение плана потребности в материалах
9. Осуществление обратной связи

Модуль планирования развития бизнеса определяет миссию компании: её нишу на рынке, оценку и определение прибылей, финансовые ресурсы. Фактически, он утверждает, в условных финансовых единицах, что компания собирается произвести и продать, и оценивает, какое количество средств необходимо инвестировать в разработку и развитие продукта, чтобы выйти на планируемый уровень прибыли. Таким образом, выходным элементом этого модуля является бизнес-план.

Модуль планирования продаж оценивает (обычно в единицах готового изделия), какими должны быть объем и динамика продаж, чтобы был выполнен установленный бизнес-план. Изменения плана продаж, несомненно, влекут за собой изменения в результатах других модулей.

Модуль планирования производства утверждает план производства всех видов готовых изделий и их характеристики. Для каждого вида изделия в рамках выпускаемой линии продукции существует своя собственная программа производства. Таким образом, совокупность производственных программ для всех видов выпускаемых изделий, представляет собой производственный план предприятия в целом.

Модуль планирования потребности в материалах (или видах услуг) на основе производственной программы для каждого вида готового изделия определяет

требуемое расписание закупки и/или внутреннего производства всех материалов комплектующих этого изделия, и, соответственно, их сборку.

Модуль планирования производственных мощностей преобразует план производства в конечные единицы загрузки рабочих мощностей (станков, рабочих, лабораторий и т.д.)

Модуль обратной связи позволяет обсуждать и решать возникающие проблемы с поставщиками комплектующих материалов, дилерами и партнерами. Тем самым, этот модуль собственно и реализует знаменитый «принцип замкнутой петли» (closed loop principle) в системе. Обратная связь особенно необходима при изменении отдельных планов, оказавшихся невыполнимыми и подлежащих пересмотру.

Рис. 3.4 Взаимодействие модулей в MRPII-системе

3.6 Концепция ERP-систем

Основные понятия производственного менеджмента (в том числе и термин «ERP») можно считать вполне устоявшимися. В этой области признанным «стандартом де-факто» служит терминология Американской ассоциации по

управлению запасами и производством (American Production and Inventory Control Society, APICS). Основные термины и определения приводятся в Словаре APICS, который регулярно обновляется по мере развития теории и практики управления. Именно в этом издании содержится наиболее полное и точное определение ERP-системы.

В соответствии со Словарем APICS, термин «ERP-система» (Enterprise Resource Planning - Управление ресурсами предприятия) может употребляться в двух значениях.

ERP-система – информационная система для идентификации и планирования всех ресурсов предприятия, которые необходимы для осуществления продаж, производства, закупок и учета в процессе выполнения клиентских заказов.

ERP методология - это методология эффективного планирования и управления всеми ресурсами предприятия, которые необходимы для осуществления продаж, производства, закупок и учета при исполнении заказов клиентов в сферах производства, дистрибьюции и оказания услуг.

Таким образом, термин ERP может означать не только информационную систему, но и соответствующую методологию управления, реализуемую и поддерживаемую этой информационной системой.

Отличия ERP от MRPII

В настоящее время практически все разработчики MRPII-/ERP-систем относят свои системы к классу ERP.

«ERP» - очень модная аббревиатура, способная увеличить продажи системы, по сути не принадлежащей к этому классу. Дело доходит до того, что начинают позиционировать финансово-управленческие системы со слабым производственным блоком как «полноценные ERP-системы», вводя потребителей в заблуждение. Эта путаница усугубляется отсутствием ERP-стандарта.

Проведем сравнительную характеристику систем двух классов – ERP и MRPII.

Сразу следует отметить, что и для MRPII-систем, и для ERP-систем основным является производство. Они, безусловно, развиваются в связи с запросами рынка: добавляются новые функциональности, решения переносятся на новые технологические платформы. Однако производственные подсистемы остаются центральными для рассматриваемых систем, и различия между MRPII-/ERP-системами лежат именно в области планирования производства. Связаны эти различия с глубиной реализации планирования, что обусловлено ориентацией этих систем на различные сегменты рынка.

ERP-системы создаются для больших многофункциональных и территориально распределенных производственных корпораций (например, холдингов, ТНК, ФПГ и т. д.). MRPII-системы ориентированы на рынок средних предприятий, которым не требуется вся мощность ERP-систем.

Собственно, различие MRPII- и ERP-систем понятно уже из их названия: с одной стороны, планирование корпоративных ресурсов (Enterprise Resources Planning), с другой - планирование производственных ресурсов (Manufacturing Resources Planning).

Существенные же отличия ERP от MRP II можно выразить следующей формулой:

ERP = MRPII + реализация всех типов производства + интегрирование планирования ресурсов по различным направлениям деятельности компании + многозвенное планирование

Безусловно, многие MRPII-системы развиваются с позиций глубины планирования и по некоторым параметрам приближаются к ERP-системам. Однако «по некоторым» не значит «по всем», поэтому с употреблением термина «ERP» нужно обращаться осторожно.

В то же время среди ERP, MRPII-систем не все могут предложить решения по системе планирования и управления производством процессного типа.

Современный рынок информационных управленческих систем состоит из тройки (по другим оценкам - пятерки) систем-лидеров, которые, собственно, и относятся к классу ERP, и множества «продвинутых» систем класса MRPII.

Безусловными лидерами являются системы SAP R/3 немецкой компании SAP AG, Oracle Applications американской компании Oracle и Baan, разработанная нидерландской компанией Baan (в мае 2000 года компания Baan была приобретена британским холдингом Invensys). Иногда к этому «элитному» списку добавляют OneWorld компании J.D.Edwards и PeopleSoft, выпускаемую одноименной компанией.

Что же касается MRPII-систем, то тут наблюдается большее количество решений, каждое из которых несет в себе уникальное сочетание функциональных и технологических особенностей. Все они отличаются различной степенью проработки производственных, финансовых и иных функций, поэтому с помощью консультантов предприятия могут подобрать систему, более всего отвечающую их запросам. Поэтому «MRPII» - это не признак ущербности системы, а показатель того, что система ориентирована на рынок средних предприятий.

Свойства ERP-систем

Главная цель концепции ERP - распространить принципы MRPII (Manufactory Resource Planning, планирование производственных ресурсов) на управление современными корпорациями. Концепция ERP представляет собой надстройку над методологией MRPII. Не внося никаких изменений в механизм планирования производственных ресурсов, она позволяет решить ряд дополнительных задач, связанных с усложнением структуры компании.

Концепция ERP до сих пор не стандартизована. Когда возникает вопрос об отнесении конкретной информационной системы управления к классу развитых MRP II-систем или к классу ERP, специалисты расходятся во мнениях, поскольку

выделяют различные критерии принадлежности системы классу ERP. Однако, суммируя различные точки зрения, можно указать основные черты, которыми должны обладать ERP-системы.

Системы класса ERP отличает набор следующих свойств:

1. Универсальность с точки зрения типов производств;
2. Поддержка многозвенного производственного планирования;
3. Более широкая (по сравнению с mrpii) сфера интегрированного планирования ресурсов;
4. Включение в систему мощного блока планирования и учета корпоративных финансов;
5. Внедрение в систему средств поддержки принятия решений.

Заключение

В заключении все же отметим недостатки языка, которые конечно полезно знать и использовать их для правильного определения степени применимости UML:

Избыточность языка. UML часто критикуется, как неоправданно большой и сложный. Он включает много избыточных или практически неиспользуемых диаграмм и конструкций. Чаще это можно услышать в отношении UML 2.0, чем UML 1.0, так как более новые ревизии включают больше «разработанных-комитетом» компромиссов.

Неточная семантика. Так как UML определён комбинацией себя (абстрактный синтаксис), OCL (языком описания ограничений — формальной проверки правильности) и подробной семантики, то он лишен скованности присущей языкам, точно определённым техниками формального описания. В некоторых случаях абстрактный синтаксис UML и OCL противоречат друг другу, в других случаях они неполные. Неточность описания самого UML одинаково отражается на пользователях и поставщиках инструментов, приводя к несовместимости инструментов из-за уникального трактования спецификаций.

Проблемы при изучении и внедрении. Вышеописанные проблемы делают проблематичным изучение и внедрение UML, особенно когда руководство насильно заставляет использовать UML инженеров при отсутствии у них предварительных навыков.

Только код отражает код. Ещё одно мнение, что важны рабочие системы, а не красивые модели. Как лаконично выразился Джек Ривс, «The code is the design» («Код и есть проект»). В соответствии с этим мнением, существует потребность в лучшем способе написания ПО. UML ценится при подходах, которые компилируют модели для генерирования исходного или выполнимого кода. Однако этого всё же может быть недостаточно, так как UML не имеет свойств

полноты по Тьюрингу и любой сгенерированный код будет ограничен тем, что может разглядеть или предположить интерпретирующий UML инструмент.

Кумулятивная нагрузка. Рассогласование нагрузки (Cumulative Impedance/Impedance mismatch). Рассогласование нагрузки — термин из теории системного анализа для обозначения неспособности входа системы воспринять выход другой. Как в любой системе обозначений UML может представить одни системы более кратко и эффективно, чем другие. Таким образом, разработчик склоняется к решениям, которые более комфортно подходят к переплетению сильных сторон UML и языков программирования. Проблема становится более очевидной, если язык разработки не придерживается принципов ортодоксальной объектно-ориентированной доктрины (не старается соответствовать традиционным принципам ООП).

Пытается быть всем для всех. UML — это язык моделирования общего назначения, который пытается достигнуть совместимости со всеми возможными языками разработки. В контексте конкретного проекта, для достижения командой проектировщиков определённой цели, должны быть выбраны применимые возможности UML. Кроме того, пути ограничения области применения UML в конкретной области проходят через формализм, который не полностью сформулирован, и который сам является объектом критики.

Список литературы

1. Брукс П. Проектирование процесса проектирования: записки компьютерного эксперта.: Пер. с англ. - М.: Вильямс, 2012. – 464 с.
2. Эванс Э. Предметно-ориентированное проектирование (DDD): структуризация сложных программных систем.: Пер. с англ. - М.: Вильямс, 2011. – 448 с.
3. Фримен Э., Сьерра К., Б. Бейтс. Паттерны проектирования. – СПб.: Питер, 2011. – 656 с.
4. Гвоздева Т. В. Баллод Б. А. Проектирование информационных систем. Феникс, 2009. – 508с.
5. Эванс Э. Предметно-ориентированное проектирование (DDD). Структуризация сложных программных систем. Вильямс, 2010. – 448с.
6. Лешек А. Мацяшек Анализ и проектирование информационных систем с помощью UML. Вильямс, 2009. - 816 стр.
7. Буч Г., РамбоД., Якобсон И. Введение в UML от создателей языка.2-е изд. – М.: ДМК Пресс, 2011. – 496 с.
8. Тидвелл Д. Разработка пользовательских интерфейсов. 2-е изд. – СПб.: Питер, 2011. – 480 с.

9. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Д. Приемы объектно-ориентированного проектирования. Библиотека программиста. – СПб.: Питер, 2010. – 368 с.