Jet Info

ИНФОРМАЦИОННЫЙ БЮЛЛЕТЕНЬ № 4 (131)/2004

СОВРЕМЕННЫЕ ТЕХНОЛОГИИ СОЗДАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (обзор)

КОРПОРАТИВНЫЕ СИСТЕМЫ

СОВРЕМЕННЫЕ ТЕХНОЛОГИИ СОЗДАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (обзор)

А.М. Вендров

СОДЕРЖАНИЕ

Введение

Основные особенности и проблемы современных программных проектов

Накопленный к настоящему времени опыт создания систем ПО показывает, что это сложная и трудоемкая работа, требующая высокой квалификации участвующих в ней специалистов. Однако до настоящего времени создание таких систем нередко выполняется на интуитивном уровне с применением неформализованных методов, основанных на искусстве, практическом опыте, экспертных оценках и дорогостоящих экспериментальных проверках качества функционирования ПО. По данным Института программной инженерии (Software Engineering Institute, SEI) в последние годы до 80% всего эксплуатируемого ПО разрабатывалось вообще без использования какой-либо дисциплины проектирования, методом «code and fix» (кодирования и исправления ошибок).

Проблемы создания ПО следуют из его свойств. Еще в 1975 г. Фредерик Брукс, проанализировав свой уникальный по тем временам опыт руководства крупнейшим проектом разработки операционной системы OS/360, определил перечень неотъемлемых свойств ПО: сложность, согласованность, изменяемость и незримость [Брукс-99]. Что же касается современных крупномасштабных проектов ПО, то они характеризуются, как правило, следующими особенностями:

Характеристики объекта внедрения:

- структурная сложность (многоуровневая иерархическая структура организации) и территориальная распределенность;
- функциональная сложность (многоуровневая иерархия и большое количество функций, выполняемых организацией; сложные взаимосвязи между ними);
- информационная сложность (большое количество источников и потребителей информации (министерства и ведомства, местные органы власти, организации-партнеры), разнообразные формы и форматы представления информации, сложная информационная модель объекта — большое количество информационных сущностей и сложные взаимосвязи между ними), сложная технология прохождения документов;
- сложная динамика поведения, обусловленная высокой изменчивостью внешней среды (изменения в законодательных и нормативных актах, нестабильность экономики и политики) и внутренней среды (структурные реорганизации, текучесть кадров).

Технические характеристики проектов создания ПО:

- различная степень унифицированности проектных решений в рамках одного проекта;
- высокая техническая сложность, определяемая наличием совокупности тесно взаимодействующих компонентов (подсистем), имеющих свои локальные задачи и цели функционирования (транзакционных приложений, предъявляющих повышенные требования к надежности, безопасности и производительности, и приложений аналитической обработки (систем поддержки принятия решений), использующих нерегламентированные запросы к данным большого объема);
- отсутствие полных аналогов, ограничивающее возможность использования каких-либо типовых проектных решений и прикладных систем, высокая доля вновь разрабатываемого ПО;
- большое количество и высокая стоимость унаследованных приложений (существующего прикладного ПО), функционирующих в различной среде (персональные компьютеры, миникомпьютеры, мэйнфреймы), необходимость интеграции унаследованных и вновь разрабатываемых приложений;
- большое количество локальных объектов внедрения, территориально распределенная и неоднородная среда функционирования (СУБД, операционные системы, аппаратные платформы);
- большое количество внешних взаимодействующих систем различных организаций с различными форматами обмена информа-

цией (налоговая служба, налоговая полиция, Госстандарт, Госкомстат, Министерство финансов, МВД, местная администрация).

Организационные характеристики проектов создания ПО:

- различные формы организации и управления проектом: централизованно управляемая разработка тиражируемого ПО, экспериментальные пилотные проекты, инициативные разработки, проекты с участием как собственных разработчиков, так и сторонних компаний на контрактной основе;
- большое количество участников проекта как со стороны заказчиков (с разнородными требованиями), так и со стороны разработчиков (более 100 человек), разобщенность и разнородность отдельных групп разработчиков по уровню квалификации, сложившимся традициям и опыту использования тех или иных инструментальных средств;
- значительная длительность жизненного цикла системы, в том числе значительная временная протяженность проекта, обусловленная масштабами организации-заказчика, различной степенью готовности отдельных ее подразделений к внедрению ПО и нестабильностью финансирования проекта;
- высокие требования со стороны заказчика к уровню технологической зрелости организаций-разработчиков (наличие сертификации в соответствии с международными и отечественными стандартами).

В конце 60-х годов прошлого века в США было отмечено явление под названием «software crisis» (кризис ПО). Это выражалось в том, что большие проекты стали выполняться с отставанием от графика или с превышением сметы расходов, разработанный продукт не обладал требуемыми функциональными возможностями, производительность его была низка, качество получаемого программного обеспечения не устраивало потребителей.

Аналитические исследования и обзоры, выполняемые в течение ряда последних лет ведущими зарубежными аналитиками, показывали не слишком обнадеживающие результаты. Так, например, результаты исследований, выполненных в 1995 году компанией Standish Group, которая проанализировала работу 364 американских корпораций и итоги выполнения более 23 тысяч проектов, связанных с разработкой ПО, выглядели следующим образом:

- только 16,2% завершились в срок, не превысили запланированный бюджет и реализовали все требуемые функции и возможности;
- 52,7% проектов завершились с опозданием, расходы превысили запланированный бюджет, требуемые функции не были реализованы в полном объеме;
- 31,1% проектов были аннулированы до завершения;
- для двух последних категорий проектов бюджет среднего проекта оказался превышенным на 89%, а срок выполнения — на 122%.

В 1998 году процентное соотношение трех перечисленных категорий проектов лишь немного изменилось в лучшую сторону (26%, 46% и 28% соответственно).

В последние годы процентное соотношение трех перечисленных категорий проектов также незначительно изменяется в лучшую сторону, однако, по оценкам ведущих аналитиков, это происходит в основном за счет снижения масштаба выполняемых проектов, а не за счет повышения управляемости и качества проектирования.

В числе причин возможных неудач, по мнению разработчиков, фигурируют:

- нечеткая и неполная формулировка требований к ПО;
- недостаточное вовлечение пользователей в работу над проектом;
- отсутствие необходимых ресурсов;
- неудовлетворительное планирование и отсутствие грамотного управления проектом;
- частое изменение требований и спецификаций;
- новизна и несовершенство используемой технологии;
- недостаточная поддержка со стороны высшего руководства;
- недостаточно высокая квалификация разработчиков, отсутствие необходимого опыта.

Объективная потребность контролировать процесс разработки сложных систем ПО, прогнозировать и гарантировать стоимость разработки, сроки и качество результатов привела в конце 60-х годов прошлого века к необходимости перехода от кустарных к индустриальным способам создания ПО и появлению совокупности инженерных методов и средств создания ПО, объединенных общим названием «программная инженерия» (software engineering). В основе программной инженерии лежит одна фундаментальная идея: проектирование ПО яв-

ляется формальным процессом, который можно изучать и совершенствовать. Освоение и правильное применение методов и средств создания ПО позволяет повысить его качество, обеспечить управляемость процесса проектирования ПО и увеличить срок его жизни.

В то же время, попытки чрезмерной формализации процесса, а также прямого заимствования идей и методов из других областей инженерной деятельности (строительства, производства) привели к ряду серьезных проблем. После двух десятилетий напрасных ожиданий повышения продуктивности процессов создания ПО, возлагаемых на новые методы и технологии, специалисты в индустрии ПО пришли к пониманию, что фундаментальная проблема в этой области — неспособность эффективного управления проектами создания ПО. Невозможно достичь удовлетворительных результатов от применения даже самых совершенных технологий и инструментальных средств, если они применяются бессистемно, разработчики не обладают необходимой квалификацией для работы с ними, и сам проект выполняется и управляется хаотически, в режиме «тушения пожара». Бессистемное применение технологий создания ПО (ТС ПО), в свою очередь, порождает разочарование в используемых методах и средствах (анализ мнений разработчиков показывает, что среди факторов, влияющих на эффективность создания ПО, используемым методам и средствам придается гораздо меньшее значение, чем квалификации и опыту разработчиков). Если в таких условиях отдельные проекты завершаются успешно, то этот успех достигается за счет героических усилий фанатично настроенного коллектива разработчиков. Постоянное повышение качества создаваемого ПО и снижение его стоимости может быть обеспечено только при условии достижения организацией необходимой технологической зрелости, создании эффективной инфраструктуры как в сфере разработки ПО, так и в управлении проектами. В соответствии с моделью SEI CMM (Capability Maturity Model), в хорошо подготовленной (зрелой) организации персонал обладает технологией и инструментарием оценки качества процессов создания ПО на протяжении всего жизненного цикла ПО и на уровне всей организации.

Одна из причин распространенности «хаотического» процесса создания ПО — стремление сэкономить на стадии разработки, не затрачивая времени и средств на обучение разработчиков и внедрение технологического процесса создания ПО. Эти затраты до недавнего времени были довольно значительными и составляли, по различным оценкам (в частности, Gartner Group), более $100 тыс. и около трех лет на внедрение развитой ТС ПО, охватывающей большинство процессов жизненного цикла ПО, в многочисленной команде разработчиков (до 100 чел.). Причина — в «тяжести» технологических процессов. «Тяжелый» процесс обладает следующими особенностями:

- необходимость документировать каждое действие разработчиков;
- множество рабочих продуктов (в первую очередь — документов), создаваемых в бюрократической атмосфере;
- отсутствие гибкости;
- детерминированность (долгосрочное детальное планирование и предсказуемость всех видов деятельности, а также распределение человеческих ресурсов на длительный срок, охватывающий большую часть проекта.

Альтернативой «тяжелому» процессу является адаптивный (гибкий) процесс, основанный на принципах «быстрой разработки ПО», интенсивно развиваемых в последнее десятилетие.

Современные тенденции в программной инженерии

В начале 2001 года века ряд ведущих специалистов в области программной инженерии (Алистер Коберн, Мартин Фаулер, Джим Хайсмит, Кент Бек и другие) сформировали группу под названием Agile Alliance. Слово agile (быстрый, ловкий, стремительный) отражало в целом их подход к разработке ПО, основанный на богатом опыте участия в разнообразных проектах в течение многих лет. Этот подход под названием «Быстрая разработка ПО» (Agile software development) [Коберн-02-1] базируется на четырех идеях, сформулированных ими в документе «Манифест быстрой разработки ПО» (Agile Alliance's Manifesto) и заключающихся в следующем:

- индивидуумы и взаимодействия между ними ценятся выше процессов и инструментов;
- работающее ПО ценится выше всеобъемлющей документации;
- сотрудничество с заказчиками ценится выше формальных договоров;
- реагирование на изменения ценится выше строгого следования плану.

При таком подходе технология занимает в процессе создания ПО вполне определенное место. Она повышает эффективность деятельности разработчиков при наличии любых из следующих четырех условий:

- когда она позволяет людям легче выразить свои мысли;
- когда она выполняет задачи, невыполнимые вручную;
- когда она автоматизирует утомительные и подверженные ошибкам действия.;
- когда она облегчает общение между людьми;

Технология не должна действовать против характера культурных ценностей и познавательной способности человека.

При этом следует четко понимать: при всех достоинствах быстрой разработки ПО этот подход не является универсальным и применим только в проектах определенного класса. Для характеристики таких проектов Алистер Коберн ввел два параметра — критичность и масштаб. Критичность определяется последствиями, вызываемыми дефектами в ПО, ее уровень может иметь одно из четырех значений:

- C — дефекты вызывают потерю удобства;
- D — дефекты вызывают потерю возместимых средств (материальных или финансовых);
- E — дефекты вызывают потерю невозместимых средств;
- L — дефекты создают угрозу человеческой жизни.

Масштаб определяется количеством разработчиков, участвующих в проекте:

- от 1 до 6 человек — малый масштаб;
- от 6 до 20 человек — средний масштаб;
- свыше 20 человек — большой масштаб.

По оценке Коберна, быстрая разработка ПО применима только в проектах малого и среднего масштаба с низкой критичностью (C или D). Общие принципы оценки технологий в таких проектах заключаются в следующем:

- интерактивное общение лицом к лицу — это самый дешевый и быстрый способ обмена информацией;
- избыточная «тяжесть» технологии стоит дорого;
- более многочисленные команды требуют более «тяжелых» и формальных технологий;
- большая формальность подходит для проектов с большей критичностью;
- возрастание обратной связи и коммуникации сокращает потребность в промежуточных и конечных продуктах;
- дисциплина, умение и понимание противостоят процессу, формальности и документированию;
- потеря эффективности в некритических видах деятельности вполне допустима.

Одним из наиболее известных примеров практической реализации подхода быстрой разработки ПО является «Экстремальное программирование» (Extreme Programming — XP) [Коберн-02-1]. Этот метод предназначен для небольших компактных команд, нацеленных на получение как можно более высокого качества и продуктивности, и достигает этого посредством насыщенной, неформальной коммуникации, придания на персональном уровне особого значения умению и навыкам, дисциплине и пониманию, сводя к минимуму все промежуточные рабочие продукты.

Методические основы технологий создания ПО

Визуальное моделирование

Под моделью ПО в общем случае понимается формализованное описание системы ПО на определенном уровне абстракции. Каждая модель определяет конкретный аспект системы, использует набор диаграмм и документов заданного формата, а также отражает точку зрения и является объектом деятельности различных людей с конкретными интересами, ролями или задачами.

Графические (визуальные) модели представляют собой средства для визуализации, описания, проектирования и документирования архитектуры системы. Разработка модели системы ПО промышленного характера в такой же мере необходима, как и наличие проекта при строительстве большого здания. Это утверждение справедливо как в случае разработки новой

системы, так и при адаптации типовых продуктов класса R/3 или BAAN, в составе которых также имеются собственные средства моделирования. Хорошие модели являются основой взаимодействия участников проекта и гарантируют корректность архитектуры. Поскольку сложность систем повышается, важно располагать хорошими методами моделирования. Хотя имеется много других факторов, от которых зависит успех проекта, но наличие строгого стандарта языка моделирования является весьма существенным.

Состав моделей, используемых в каждом конкретном проекте, и степень их детальности в общем случае зависят от следующих факторов:

- сложности проектируемой системы;
- необходимой полноты ее описания;
- знаний и навыков участников проекта;
- времени, отведенного на проектирование.

Визуальное моделирование оказало большое влияние на развитие ТС ПО вообще и CASE-средств в частности. Понятие CASE (Computer Aided Software Engineering) используется в настоящее время в весьма широком смысле. Первоначальное значение этого понятия, ограниченное только задачами автоматизации разработки ПО, в настоящее время приобрело новый смысл, охватывающий большинство процессов жизненного цикла ПО.

CASE-технология представляет собой совокупность методов проектирования ПО, а также набор инструментальных средств, позволяющих в наглядной форме моделировать предметную область, анализировать эту модель на всех стадиях разработки и сопровождения ПО и разрабатывать приложения в соответствии с информационными потребностями пользователей. Большинство существующих CASE-средств основано на методах структурного или объектно-ориентированного анализа и проектирования, использующих спецификации в виде диаграмм или текстов для описания внешних требований, связей между моделями системы, динамики поведения системы и архитектуры программных средств.

Методы структурного анализа и проектирования ПО

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 [Марка-93] 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 [Черемных-01], 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) [Калашян-03] 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) [Конноли-03]. 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).

Методы объектно-ориентированного анализа и проектирования ПО. Язык UML

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 в его фундаментальной книге [Буч-99] и последующих работах.

Большинство современных методов ООАП [Гома-02, Кватрани-03, Ларман-02, Розенберг-02] основаны на использовании языка UML. Унифицированный язык моделирования UML (Unified Modeling Language) [Буч-2000, Рамбо-02, Фаулер-99] 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) 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.

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):
  - 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 [Коберн-02-2]. 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 поразному, 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 попрежнему 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.

Методы моделирования бизнес-процессов и спецификации требований

Моделирование бизнес-процессов является важной составной частью проектов по созданию крупномасштабных систем ПО. Отсутствие таких моделей является одной из главных причин неудач многих проектов.

Назначением будущего ПО является, в первую очередь, решение проблем бизнеса. Требования к ПО формируются на основе бизнес-модели, а критерии проектирования систем прежде всего основываются на наиболее полном их удовлетворении.

Модели бизнес-процессов являются не просто промежуточным результатом, используемым консультантом для выработки каких-либо рекомендаций и заключений. Они представляют собой самостоятельный результат, имеющий большое практическое значение.

На сегодняшний день в моделировании бизнес-процессов преобладает процессный подход. Его основной принцип заключается в структурировании деятельности организации в соответствии с ее бизнес-процессами, а не организационно-штатной структурой. Модель, основанная на организационно-штатной структуре, может продемонстрировать лишь хаос, царящий в организации (о котором в принципе руководству и так известно, иначе оно бы не инициировало соответствующие работы), на ее основе можно только внести предложения об изменении этой структуры. С другой стороны, модель, основанная на бизнес-процессах, содержит в себе и организационно-штатную структуру предприятия.

Процессный подход может использовать любые из перечисленных выше средств моделирования. Однако, в настоящее время наблюдается тенденция интеграции разнообразных методов моделирования и анализа систем, проявляющаяся в форме создания интегрированных средств моделирования. Одним из таких средств является продукт, носящий название ARIS — Architecture of Integrated Information System, разработанный германской фирмой IDS Scheer [Каменнова-01].

Система ARIS представляет собой комплекс средств анализа и моделирования деятельности предприятия. Ее методическую основу составляет совокупность различных методов моделирования, отражающих разные взгляды на исследуемую систему. Одна и та же модель может разрабатываться с использованием нескольких методов, что позволяет использовать ARIS специалистам с различными теоретическими знаниями и настраивать его на работу с системами, имеющими свою специфику.

ARIS поддерживает четыре типа моделей, отражающих различные аспекты исследуемой системы:

- организационные модели, представляющие структуру системы — иерархию организационных подразделений, должностей и конкретных лиц, связи между ними, а также территориальную привязку структурных подразделений;
- функциональные модели, содержащие иерархию целей, стоящих перед аппаратом управления, с совокупностью деревьев функций, необходимых для достижения поставленных целей;
- информационные модели, отражающие структуру информации, необходимой для реализации всей совокупности функций системы;
- модели управления, представляющие комплексный взгляд на реализацию бизнес-процессов в рамках системы.

Для построения перечисленных типов моделей используются как собственные методы моделирования ARIS, так и различные известные методы и языки моделирования — ERM, UML, OMT и др.

В процессе моделирования каждый аспект деятельности предприятия сначала рассматривается отдельно, а после детальной проработки всех аспектов строится интегрированная модель, отражающая все связи между различными аспектами.

Модели в ARIS представляют собой диаграммы, элементами которых являются разнообразные объекты — «функция», «событие», «структурное подразделение», «документ» и т.п. Между объектами устанавливаются разнообразные связи. Каждому объекту соответствует определенный набор атрибутов, которые позволяют ввести дополнительную информацию о конкретном объекте. Значения атрибутов могут использоваться при имитационном моделировании или для проведения стоимостного анализа.

Основная бизнес-модель ARIS — eEPC (extended Event Driven Process Chain — расширенная модель цепочки процессов, управляемых событиями). По существу, она расширяет возможности IDEF0, IDEF3 и DFD, обладая всеми их достоинствами и недостатками. Применение большого числа различных объектов, связанных различными типами связей, может значительно увеличить размер модели и сделать ее плохо читаемой.

Бизнес-процесс в нотации eEPC представляет собой поток последовательно выполняемых работ (процедур, функций), расположенных в порядке их выполнения. Реальная длительность выполнения процедур в eEPC визуально не отражается. Это приводит к тому, что при создании моделей возможны ситуации, когда на одного исполнителя будет возложено выполнение двух задач одновременно. Используемые при построении модели символы логики позволяют отразить ветвление и слияние бизнес-процесса. Для получения информации о реальной длительности процессов необходимо использовать другие инструменты описания, например графики Ганта в системе MS Project.

Ряд современных методов моделирования бизнес-процессов основан на использовании языка UML. Хотя UML изначально предназначался для моделирования систем ПО, его ис-

пользование в другой области стало возможным благодаря наличию в UML механизмов расширения (стереотипов).

Среди таких методов наиболее известными являются метод Ericsson-Penker и метод, реализованный в технологии Rational Unified Process (RUP).

Метод Ericsson-Penker [Eriksson-2000] представляет интерес прежде всего в связи с попыткой применения UML в рамках процессного подхода к моделированию бизнес-процессов. Авторы метода создали свой профиль UML для моделирования бизнес-процессов, введя набор стереотипов, описывающих процессы, ресурсы, правила и цели деятельности организации. Метод использует четыре основные категории бизнес-модели:

- Ресурсы — различные объекты, используемые или участвующие в бизнес-процессах (люди, материалы, информация или продукты).
- Процессы — виды деятельности, изменяющие состояние ресурсов в соответствии с бизнес-правилами.
- Цели — назначение бизнес-процессов Цели могут быть разбиты на подцели и соотнесены с отдельными процессами.
- Бизнес-правила — условия или ограничения выполнения процессов (функциональные, поведенческие или структурные). Правила могут быть определены с использованием языка OCL.

Основной диаграммой UML, используемой в данном методе, является диаграмма деятельности. Метод Eriksson-Penker представляет процесс в виде деятельности со стереотипом «process» (основой данного представления является расширение метода IDEF0). Полная бизнес-модель включает множество представлений, подобных представлениям архитектуры ПО. Каждое представление выражено в одной или более диаграммах UML. Диаграммы могут иметь различные типы и изображать процессы, правила, цели и ресурсы во взаимодействиях друг с другом. Метод использует четыре различных представления бизнес-модели:

- концептуальное представление — структура целей и проблем;
- представление процессов — взаимодействие между процессами и ресурсами (в виде набора диаграмм деятельности);
- структурное представление — структура организации и ресурсов (в виде диаграмм классов);
- представление поведения — поведение отдельных ресурсов и детализация процессов (в виде диаграмм деятельности, состояний и взаимодействия).

Методика моделирования RUP [Крачтен-02] предусматривает построение двух моделей:

- модели бизнес-процессов (Business Use Case Model);
- модели бизнес-анализа (Business Analysis Model).

Модель бизнес-процессов представляет собой расширение модели вариантов использования UML за счет введения набора стереотипов — Business Actor (стереотип действующего лица) и Business Use Case (стереотип варианта использования). Business Actor (действующее лицо бизнес-процессов) — это некоторая роль, внешняя по отношению к бизнес-процессам организации. Business Use Case (вариант использования с точки зрения бизнес-процессов) определяется как описание последовательности действий (потока событий) в рамках некоторого бизнес-процесса, приносящей ощутимый результат конкретному действующему лицу. Это определение подобно общему определению бизнес-процесса, но имеет более точный смысл. В терминах объектной модели Business Use Case представляет собой класс, объектами которого являются конкретные потоки событий в рамках описываемого бизнес-процесса.

Описание Business Use Case может сопровождаться целью процесса, которая, так же, как и в методе Eriksson-Penker, моделируется с помощью класса со стереотипом «goal», а дерево целей изображается в виде диаграммы классов.

Для каждого Business Use Case строится модель бизнес-анализа — объектная модель, описывающая реализацию бизнес-процесса в терминах взаимодействующих объектов (бизнес-объектов — Business Object), принадлежащих к двум классам — Business Worker и Business Entity. Business Worker (исполнитель) — класс, представляющий собой абстракцию исполнителя, выполняющего некоторые действия в рамках бизнес-процесса. Исполнители взаимодействуют между собой и манипулируют различными сущностями, участвуя в реализациях сценариев Business Use Case. Business Entity (сущность) является объектом различных действий со стороны исполнителей.

Кроме диаграммы данных классов, модель бизнес-анализа может включать:

- Диаграммы последовательности (и кооперативные диаграммы), описывающие сценарии Business Use Case в виде последовательности обмена сообщениями между объектами-действующими лицами и объектами-исполнителями. Такие диаграммы помогают явно определить в модели обязанности каждого исполнителя в виде набора его операций.
- Диаграммы деятельности, описывающие взаимосвязи между сценариями одного или различных Business Use Case.
- Диаграммы состояний, описывающие поведение отдельных бизнес-объектов.

Методика моделирования Rational Unified Process обладает следующими достоинствами:

- модель бизнес-процессов строится вокруг участников процессов (заинтересованных лиц) и их целей, помогая выявить все потребности клиентов организации. Такой подход в наибольшей степени применим для организаций, работающих в сфере оказания услуг (торговые организации, банки, страховые компании и т.д.);
- моделирование на основе вариантов использования способствует хорошему пониманию бизнес-модели со стороны заказчиков.

Однако следует отметить, что при моделировании деятельности крупной организации, занимающейся как производством продукции, так и оказанием услуг, необходимо применять различные методики моделирования, поскольку для моделирования производственных процессов более предпочтительным является процессный подход (например, метод Eriksson-Penker).

Спецификация требований к ПО является составной частью процесса управления требованиями [Леффингуэлл-02]. Выявленные в результате применения перечисленных методов требования к ПО оформляются в виде ряда документов и моделей. Так, в технологии RUP функциональные требования к системе моделируются и документируются с помощью вариантов использовании. Стиль их написания зависит от масштаба, количества участников и критичности проекта.

Спецификация требований в RUP не требует обязательного моделирования бизнес-процессов организации, для которых создается ПО, однако, наличие бизнес-моделей существенно упрощает построение системной модели вариантов использования.

Методы анализа и проектирования ПО

Целью анализа требований является трансформация функциональных требований к ПО в предварительный системный проект и создание стабильной основы архитектуры системы. В процессе проектирования системный проект «погружается» в среду реализации с учетом всех нефункциональных требований.

Все современные ТС ПО реализуют ту или иную методику анализа и проектирования ПО. Одна из типичных методик ООАП реализована в технологии RUP. Согласно этой методике, объектно-ориентированный анализ включает два вида деятельности: архитектурный анализ и анализ вариантов использования. Архитектурный анализ выполняется архитектором системы и включает в себя:

- утверждение общих стандартов (соглашений) моделирования и документирования системы;
- предварительное выявление архитектурных механизмов (надежности, безопасности и т.д.);
- формирование набора основных абстракций предметной области (классов анализа);
- формирование начального представления архитектурных уровней.

Анализ вариантов использования выполняется проектировщиками и включает в себя:

- идентификацию классов, участвующих в реализации потоков событий варианта использования;
- распределение поведения, реализуемого вариантом использования, между классами (определение обязанностей классов);
- определение атрибутов и ассоциаций классов.

В потоках событий варианта использования выявляются классы трех типов:

Граничные классы (Boundary) — служат посредниками при взаимодействии внешних объектов с системой. Типы граничных классов: пользовательский интерфейс (обмен информацией с пользователем, без деталей интерфейса — кнопок, списков, окон), системный интерфейс и аппаратный интерфейс (используемые протоколы, без деталей их реализации).

Классы-сущности (Entity) — представляют собой основные абстракции (понятия) разрабатываемой системы, рассматриваемые в рамках конкретного варианта использования.

Управляющие классы (Control) — обеспечивают координацию поведения объектов в системе. Примеры управляющих классов: менеджер транзакций, координатор ресурсов, обработчик ошибок.

Классы анализа отражают функциональные требования к системе и моделируют объекты предметной области. Совокупность классов анализа представляет собой начальную концептуальную модель системы

Наиболее важной частью объектно-ориентированного анализа является распределение обязанностей между классами (в виде операций классов). Оно выполняется с помощью диаграмм взаимодействия. При построении диаграмм взаимодействия возникают проблемы правильного распределения обязанностей между классами. Для их решения существует ряд образцов [Ларман-02].

Атрибуты классов анализа определяются, исходя из знаний о предметной области и требований к системе. Связи между классами (ассоциации) определяются на основе анализа кооперативных диаграмм, затем анализируются и уточняются.

Целью объектно-ориентированного проектирования является адаптация предварительного системного проекта (набора классов «анализа»), составляющего стабильную основу архитектуры системы, к среде реализации с учетом всех нефункциональных требований.

Объектно-ориентированное проектирование включает два вида деятельности:

- проектирование архитектуры системы;
- проектирование элементов системы.

Проектирование архитектуры системы выполняется архитектором системы и включает в себя:

- идентификацию архитектурных решений и механизмов, необходимых для проектирования системы;
- анализ взаимодействий между классами анализа, выявление подсистем и интерфейсов;
- формирование архитектурных уровней;
- проектирование конфигурации системы.

Проектирование элементов системы включает в себя:

- проектирование классов (детализация классов, уточнение операций и атрибутов, моделирование состояний, уточнение связей между классами);
- проектирование баз данных (в зависимости от типа используемой для хранения данных СУБД — объектной или реляционной).

Технологии создания программного обеспечения

Требования, предъявляемые к ТС ПО

ТС ПО в общем случае можно описать следующей системой понятий:

Технология создания ПО — упорядоченная совокупность взаимосвязанных технологических процессов в рамках ЖЦ ПО.

Технологический процесс — совокупность взаимосвязанных технологических операций.

Технологическая операция — основная единица работы, выполняемая определенной ролью, которая:

- подразумевает четко определенную ответственность роли;
- дает четко определенный результат (набор рабочих продуктов), базирующийся на определенных исходных данных (другом наборе рабочих продуктов);
- представляет собой единицу работы с жестко определенными границами, которые устанавливаются при планировании проекта.

Рабочий продукт — информационная или материальная сущность, которая создается, модифицируется или используется в некоторой технологической операции (модель, документ, код, тест и т.п.). Рабочий продукт определяет область ответственности роли и является объектом управления конфигурацией.

Роль — определение поведения и обязанностей отдельного лица или группы лиц в среде

организации-разработчика ПО, осуществляющих деятельность в рамках некоторого технологического процесса и ответственных за определенные рабочие продукты.

Руководство — практическое руководство по выполнению одной или совокупности технологических операций. Руководства включают методические материалы, инструкции, нормативы, стандарты и критерии оценки качества рабочих продуктов.

Инструментальное средство (CASE-средство) — программное средство, обеспечивающее автоматизированную поддержку деятельности, выполняемой в рамках технологических операций.

Основным требованием, предъявляемым к современным ТС ПО, является их **соответствие стандартам и нормативным документам**, связанным с процессами ЖЦ ПО и оценкой технологической зрелости организаций-разработчиков (ISO 12207, ISO 9000, CMM и др.). Согласно этим нормативам, ТС ПО должна поддерживать следующие процессы:

- управление требованиями;
- анализ и проектирование ПО;
- разработка ПО;
- эксплуатация;
- сопровождение;
- документирование;
- управление конфигурацией и изменениями;
- тестирование;
- управление проектом.

Полнота поддержки процессов ЖЦ ПО должна поддерживаться комплексом инструментальных средств (CASE-средств).

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

Другим важным требованием является **адаптируемость к условиям применения**, которая достигается за счет поставки технологии в электронном виде вместе с CASE-средствами и библиотеками процессов, шаблонов, методов, моделей и других компонентов, предназначенных для построения ПО того класса систем, на который ориентирована технология. Электронные технологии должны включать средства, обеспечивающие их адаптацию и развитие по результатам выполнения конкретных проектов. Процесс адаптации заключается в удалении ненужных процессов и действий ЖЦ ПО, в изменении неподходящих или в добавлении собственных процессов и действий, а также методик, стандартов и руководств.

Внедрение ТС ПО в организации

При внедрении ТС ПО следует руководствоваться рекомендациями, приведенными в стандартах [IEEE-1992, IEEE-1995, ISO-1995] (их краткий перевод приведен в [Вендров-2000]). Эти рекомендации достаточно актуальны и ценны, поскольку отражают опыт, накопленный многими зарубежными пользователями и разработчиками ТС ПО в течение длительного периода их существования.

Термин «внедрение» используется в широком смысле и включает все действия — от оценки первоначальных потребностей до полномасштабного использования ТС ПО в различных подразделениях организации. Процесс внедрения ТС ПО состоит из следующих этапов:

1) Определение потребностей в ТС ПО, характеристик объекта внедрения и проектов создания ПО.
2) Определение требований, предъявляемых к ТС ПО (анализ характеристик объекта внедрения и проектов, обоснование требований к ТС ПО, определение приоритетов требований).
3) Оценка вариантов ТС ПО. Предварительная экспертная оценка заключается в анализе доступных ТС ПО на предмет соответствия требованиям, неудовлетворительные варианты (с точки зрения реализации наиболее приоритетных требований) отвергаются, формируется список претендентов. При детализированной оценке для каждой ТС ПО-претендента формируется ее детальное описание. Источники информации для описания — техническая документация поставщика, доступные данные о реальных внедрениях, результаты выполнения пилотных проектов.

Критерий	Определение
Минимум трудоемкости создания ПО	Количество человеко-месяцев, затрачиваемых на создание ПО с использованием ТС ПО
Максимум продуктивности	Объем работы (измеряемый в количестве строк кода или функциональных точек), приходящийся на единицу трудоемкости (человеко-месяц) при использовании данной ТС ПО
Максимум качества создаваемого ПО	Количество дефектов в рабочих продуктах при использовании данной ТС ПО
Возврат инвестиций	(Доход от использования ПО - Затраты на создание и сопровождение ПО) / (Затраты на создание и сопровождение ПО)
Минимум затрат на сопровождение ПО	Отношение стоимости сопровождения ПО при использовании данной ТС ПО к совокупным затратам на информационные ТС ПО в организации
Минимум времени внедрения ТС ПО	Временной интервал от начала внедрения ТС ПО до выхода на безубыточный уровень (начало возврата инвестиций в ТС ПО)
Минимум затрат на внедрение ТС ПО	Суммарная стоимость приобретения, обучения и сопровождения ТС ПО
Минимальный срок окупаемости затрат на внедрение ТС ПО	Временной интервал от начала внедрения ТС ПО до полной окупаемости затрат на ее внедрение

Табл. 1. Критерии, применяемые для оценки ТС ПО

4) Выбор ТС ПО. Производится сравнительный анализ технологий и окончательный выбор ТС ПО с помощью экспертной оценки.
5) Адаптация ТС ПО к условиям применения. Производится формирование конкретной рабочей конфигурации ТС ПО, адаптированной к условиям объекта внедрения.

В процессе внедрения ТС ПО собирается статистика и оценивается эффективность ее внедрения с точки зрения ряда критериев (минимум трудоемкости сопровождения ПО, минимум затрат на сопровождение ПО и др.). При изменении условий объекта внедрения и по результатам анализа эффективности внедрения ТС ПО принимается решение: а) о внесении изменений в рабочую конфигурацию ТС ПО; б) о переходе на новую ТС ПО. В случае перехода повторяются пп. 3)-4)-5).

Оценка и выбор ТС ПО

Цель процесса оценки — определение функциональности и качества ТС ПО для последующего выбора. Оценка выполняется в соответствии с конкретными критериями, ее результаты включают как объективные, так и субъективные данные по каждой ТС ПО.

Процессы оценки и выбора тесно взаимосвязаны. По результатам оценки цели выбора и/или критерии выбора и их веса могут потребовать модификации. В таких случаях может понадобиться повторная оценка. Когда анализируются окончательные результаты оценки и к ним применяются критерии выбора, может быть рекомендовано приобретение технологии. Альтернативой может быть отсутствие адекватных технологий, в этом случае рекомендуется разработать новую технологию, модифицировать существующую или отказаться от внедрения.

Процесс выбора включает в себя следующие действия:

- формулировка задач выбора, включая цели, предположения и ограничения;
- выполнение всех необходимых действий по выбору, включая определение и ранжирование критериев, определение технологий-кандидатов, сбор необходимых данных и применение ранжированных критериев к результатам оценки для определения средств с наилучшими показателями;
- выполнение необходимого количества итераций с тем, чтобы выбрать (или отвергнуть) технологии, имеющие сходные показатели.

Типичный процесс оценки и/или выбора может использовать набор критериев различных типов. Каждый критерий должен быть выбран и адаптирован экспертом с учетом особенностей конкретного процесса.

Исходные данные для оценки и выбора — набор параметров (технико-экономических характеристик) ТС ПО:

1. Функциональные характеристики, ориентированные на процессы жизненного цикла ПО (управление проектом, управление требованиями, управление конфигурацией и изменениями, анализ и проектирование ПО и др.).
2. Функциональные характеристики применения (среда функционирования, совместимость с другими ТС ПО, соответствие технологическим стандартам).

3. Характеристики качества (надежность, удобство использования, эффективность, сопровождаемость, переносимость).
4. Общие характеристики (затраты на технологию, лицензионная политика, оценочный эффект от внедрения ТС ПО, инфрастуктура, требуемая для внедрения ТС ПО, доступность и качество обучения, сертификация поставщика, поддержка поставщика).

На основе данного набора параметров анализируются и классифицируются существующие ТС ПО. Общий набор критериев, применяемых для оценки ТС ПО, приведен в табл. 1.

В результате выполненной оценки может оказаться, что ни одна доступная технология не удовлетворяет в нужной мере всем критериям и не покрывает все потребности проекта. В этом случае может применяться набор средств, позволяющий построить на их базе единую технологическую среду.

Выполнение пилотного проекта

Перед полномасштабным внедрением выбранной ТС ПО в организации выполняется пилотный проект, целью которого является экспериментальная проверка правильности решений, принятых на предыдущих этапах, и подготовка к внедрению.

Пилотный проект позволяет получить важную информацию, необходимую для оценки ТС ПО и его поддержки со стороны поставщика после того, как средство установлено.

Важной функцией пилотного проекта является принятие решения относительно приобретения или отказа от использования ТС ПО. Провал пилотного проекта позволяет избежать более значительных и дорогостоящих неудач в дальнейшем, поскольку пилотный проект обычно требует приобретения относительно небольшого количества лицензий и обучения узкого круга специалистов.

Пилотный проект должен обладать следующими характеристиками:

Типичность предметной области. Чтобы облегчить окончательное определение области применения ТС ПО, предметная область пилотного проекта должна быть типичной для обычной деятельности организации. Пилотный проект должен помочь определить любую дополнительную технологию, обучение или поддержку, которые необходимы для перехода от пилотного проекта к широкомасштабному использованию ТС ПО. В рамках этих ограничений пилотный проект должен иметь небольшой, но значимый размер.

Масштабируемость. Результаты, полученные в пилотном проекте, должны показать масштабируемость ТС ПО. Цель — получить четкое представление о масштабах проектов, для которых данная ТС ПО применима.

Представительность. Пилотный проект не должен быть необычным или уникальным для организации. ТС ПО должна использоваться для решения задач, относящихся к предметной области, хорошо понимаемой всей организацией.

Критичность. Пилотный проект должен иметь существенную значимость, чтобы оказаться в центре внимания, но не должен быть критичным для успешной деятельности организации в целом.

Авторитетность. Группа специалистов, участвующих в проекте, должна обладать высоким авторитетом, при этом результаты проекта будут всерьез восприняты остальными сотрудниками организации.

Готовность проектной группы. Проектная группа должна обладать готовностью к нововведениям, технической зрелостью и приемлемым уровнем опыта и знаний в данной ТС ПО и предметной области. С другой стороны, группа должна отражать в миниатюре характеристики организации в целом.

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

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

Возможным решением должно быть одно из следующих:

- Внедрить ТС ПО. В этом случае рекомендуемый масштаб внедрения должен быть определен в терминах структурных подразделений и предметной области.
- Выполнить дополнительный пилотный проект. Такой вариант должен рассматриваться только в том случае, если остались конкретные неразрешенные вопросы относительно внедрения ТС ПО в организации.
- Отказаться от ТС ПО. В этом случае причины отказа от конкретной ТС ПО должны быть определены в терминах потребностей организации или критериев, которые остались неудовлетворенными. Перед тем как

продолжить деятельность по внедрению ТС ПО, потребности организации должны быть пересмотрены на предмет своей обоснованности.
- Отказаться от использования ТС ПО вообще. Пилотный проект может показать, что организация либо не готова к внедрению ТС ПО, либо автоматизация данного аспекта процесса создания и сопровождения ПО не дает никакого эффекта для организации.

Практическое внедрение ТС ПО

Процесс перехода к практическому использованию ТС ПО начинается с разработки и последующей реализации плана перехода. Этот план может отражать поэтапный подход к переходу — от тщательно выбранного пилотного проекта до проектов с существенно возросшим разнообразием характеристик.

План перехода должен охватывать:
- информацию относительно целей, критериев оценки, графика и возможных рисков, связанных с реализацией плана;
- информацию относительно приобретения, установки и настройки ТС ПО;
- информацию относительно интеграции с существующими средствами, включая как интеграцию средств друг с другом, так и их интеграцию в процессы разработки и эксплуатации ПО, существующие в организации;
- ожидаемые потребности в обучении и ресурсы, используемые в течение и после завершения процесса перехода;
- определение стандартных процедур использования ТС ПО.

План перехода должен определять начальную практику применения и процедуры использования средств. Реальное применение любой ТС ПО в конкретной организации и конкретном проекте невозможно без выработки ряда стандартов (правил, соглашений), которые должны соблюдаться всеми участниками проекта (это особенно актуально при коллективной разработке ПО большим количеством групп специалистов). К таким стандартам относятся следующие:
- руководства по моделированию и проектированию;
- соглашения по присвоению имен;
- процедуры контроля качества и процессов приемки, включая расписание экспертиз и используемые методологии;
- процедуры резервного копирования, защиты мастер-копий и конфигурирования базы данных проекта;
- процедуры интеграции с существующими средствами и базами данных;
- процедуры совместного использования данных и контроля целостности БД;
- стандарты и процедуры обеспечения секретности;
- стандарты документирования.
- стандарт проектирования;
- стандарт оформления проектной документации;
- стандарт интерфейса пользователя.

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

Результатом данного этапа является внедрение ТС ПО в повседневную практику организации. Кроме того, поддержка ТС ПО включается в план текущей поддержки ПО в данной организации.

Примеры ТС ПО различных компаний-поставщиков

Технология Rational Unified Process (IBM Rational Software)

На сегодняшний день практически все ведущие компании — разработчики технологий и программных продуктов (IBM, Oracle, Borland, Computer Associates и др.) располагают развитыми технологиями создания ПО, которые создавались как собственными силами, так и за счет приобретения продуктов и технологий, созданных небольшими специализированными компаниями. Выбор в качестве примера четырех перечисленных компаний объясняется их ведущими позициями на мировом рынке ТС ПО [IDC-2002, IDC-2003-1], присутствием на рос-

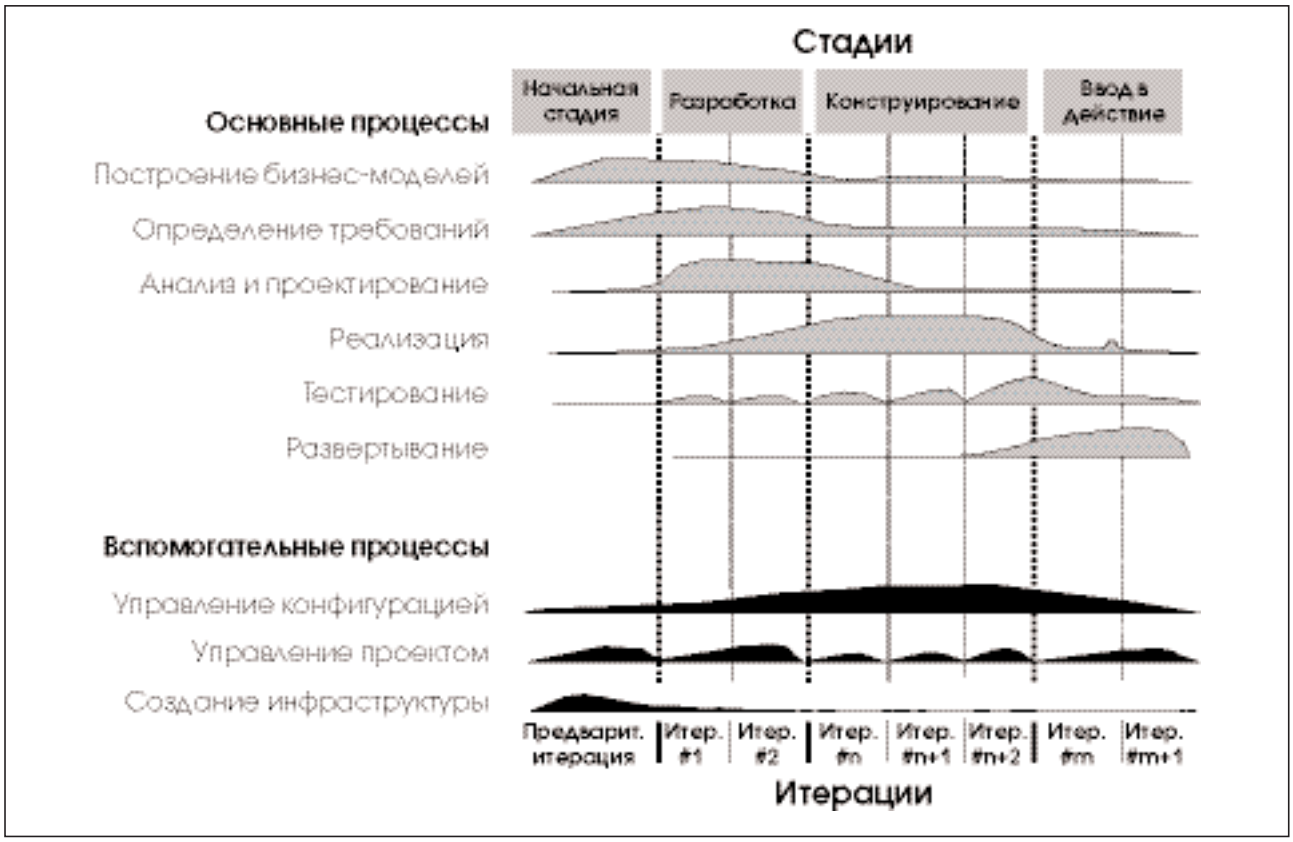

Рис. 1. Общее представление RUP

сийском рынке и ограниченным объемом настоящего обзора.

Одна из наиболее совершенных технологий, претендующих на роль мирового корпоративного стандарта — Rational Unified Process (RUP) [Крачтен-02]. RUP представляет собой программный продукт, разработанный компанией Rational Software (www.rational.com), которая в настоящее время входит в состав IBM.

RUP в значительной степени соответствует стандартам и нормативным документам, связанным с процессами ЖЦ ПО и оценкой технологической зрелости организаций-разработчиков (ISO 12207, ISO 9000, CMM и др.). Ее основными принципами являются:

1) Итерационный и инкрементный (наращиваемый) подход к созданию ПО.
2) Планирование и управление проектом на основе функциональных требований к системе — вариантов использования.
3) Построение системы на базе архитектуры ПО.

Первый принцип является определяющим. В соответствии с ним разработка системы выполняется в виде нескольких краткосрочных мини-проектов фиксированной длительности (от 2 до 6 недель), называемых итерациями. Каждая итерация включает свои собственные этапы анализа требований, проектирования, реализации, тестирования, интеграции и завершается созданием работающей системы.

Итерационный цикл основывается на постоянном расширении и дополнении системы в процессе нескольких итераций с периодической обратной связью и адаптацией добавляемых модулей к существующему ядру системы. Система постоянно разрастается шаг за шагом, поэтому такой подход называют итерационным и инкрементным.

На рис. 1 показано общее представление RUP в двух измерениях. Горизонтальное измерение представляет время, отражает динамические аспекты процессов и оперирует такими понятиями, как стадии, итерации и контрольные точки. Вертикальное измерение отражает статические аспекты процессов и оперирует такими понятиями, как виды деятельности (технологические операции), рабочие продукты, исполнители и дисциплины (технологические процессы).

Согласно RUP, ЖЦ ПО разбивается на отдельные циклы, в каждом из которых создается

новое поколение продукта. Каждый цикл, в свою очередь, разбивается на четыре последовательные стадии:

- начальная стадия (inception);
- стадия разработки (elaboration);
- стадия конструирования (construction);
- стадия ввода в действие (transition).

Каждая стадия завершается в четко определенной контрольной точке (milestone). В этот момент времени должны достигаться важные результаты и приниматься критически важные решения о дальнейшей разработке.

Начальная стадия может принимать множество разных форм. Для крупных проектов начальная стадия может вылиться во всестороннее изучение всех возможностей реализации проекта, которое займет месяцы. Во время начальной стадии вырабатывается бизнес-план проекта — определяется, сколько приблизительно он будет стоить и какой доход принесет. Определяются также границы проекта, и выполняется некоторый начальный анализ для оценки размеров проекта.

Результатами начальной стадии являются:

- общее описание системы: основные требования к проекту, его характеристики и ограничения;
- начальная модель вариантов использования (степень готовности — 10-20%);
- начальный проектный глоссарий (словарь терминов);
- начальный бизнес-план;
- план проекта, отражающий стадии и итерации;
- один или несколько прототипов.

На стадии разработки выявляются более детальные требования к системе, выполняется высокоуровневый анализ предметной области и проектирование для построения базовой архитектуры системы, создается план конструирования и устраняются наиболее рискованные элементы проекта.

Результатами стадии разработки являются:

- модель вариантов использования (завершенная по крайней мере на 80%), определяющая функциональные требования к системе;
- перечень дополнительных требований, включая требования нефункционального характера и требования, не связанные с конкретными вариантами использования;
- описание базовой архитектуры будущей системы;
- работающий прототип;
- уточненный бизнес-план;
- план разработки всего проекта, отражающий итерации и критерии оценки для каждой итерации.

Самым важным результатом стадии разработки является описание базовой архитектуры будущей системы. Эта архитектура включает:

- модель предметной области, которая отражает понимание бизнеса и служит отправным пунктом для формирования основных классов предметной области;
- технологическую платформу, определяющую основные элементы технологии реализации системы и их взаимодействие.

Эта архитектура является основой всей дальнейшей разработки, она служит своего рода проектом для последующих стадий. В дальнейшем неизбежны незначительные изменения в деталях архитектуры, однако, серьезные изменения маловероятны.

Стадия разработки занимает около пятой части общей продолжительности проекта. Основными признаками завершения стадии разработки являются два события:

- разработчики в состоянии оценить с достаточно высокой точностью, сколько времени потребуется на реализацию каждого варианта использования;
- идентифицированы все наиболее серьезные риски, и степень понимания наиболее важных из них такова, что известно, как справиться с ними.

Сущность планирования заключается в определении последовательности итераций конструирования и вариантов использования, реализуемых на каждой итерации. Итерации на стадии конструирования являются одновременно инкрементными и повторяющимися:

- итерации являются инкрементными в соответствии с той функцией, которую они выполняют. Каждая итерация добавляет очередные конструкции к вариантам использования, реализованным во время предыдущих итераций;
- итерации являются повторяющимися по отношению к разрабатываемому коду. На каждой итерации некоторая часть существующего кода переписывается с целью сделать его более гибким.

Результатом стадии конструирования является продукт, готовый к передаче конечным

пользователям. Как минимум, он содержит следующее:

- ПО, интегрированное на требуемых платформах;
- руководства пользователя;
- описание текущей реализации.

Назначением стадии ввода в действие является передача готового продукта в распоряжение пользователей. Данная стадия включает:

- бета-тестирование, позволяющее убедиться, что новая система соответствует ожиданиям пользователей;
- параллельное функционирование с существующей (legacy) системой, которая подлежит постепенной замене;
- конвертирование баз данных;
- оптимизацию производительности;
- обучение пользователей и специалистов службы сопровождения.

Статический аспект RUP представлен четырьмя основными элементами:

- роли;
- виды деятельности;
- рабочие продукты;
- дисциплины.

Понятие «роль» (role) определяет поведение и ответственность личности или группы личностей, составляющих проектную команду. Одна личность может играть в проекте много различных ролей.

Под видом деятельности конкретного исполнителя понимается единица выполняемой им работы. Вид деятельности (activity) соответствует понятию технологической операции. Он имеет четко определенную цель, обычно выражаемую в терминах получения или модификации некоторых рабочих продуктов (artifacts), таких, как модель, элемент модели, документ, исходный код или план. Каждый вид деятельности связано с конкретной ролью. Продолжительность вида деятельности составляет от нескольких часов до нескольких дней, он обычно выполняется одним исполнителем и порождает только один или весьма небольшое количество рабочих продуктов. Любой вид деятельности должен являться элементом процесса планирования. Примерами видов деятельности могут быть планирование итерации, определение вариантов использования и действующих лиц, выполнение теста на производительность. Каждый вид деятельности сопровождается набором руководств (guidelines), представляющих собой методики выполнения технологических операций.

Дисциплина (discipline) соответствует понятию технологического процесса и представляет собой последовательность действий, приводящую к получению значимого результата.

В рамках RUP определены шесть основных дисциплин:

- построение бизнес-моделей;
- определение требований;
- анализ и проектирование;
- реализация;
- тестирование;
- развертывание;

и три вспомогательных:

- управление конфигурацией и изменениями;
- управление проектом;
- создание инфраструктуры.

RUP как продукт входит в состав комплекса Rational Suite, причем каждая из перечисленных выше дисциплин поддерживается определенным инструментальным средством комплекса. Физическая реализация RUP представляет собой Web-сайт, включающий следующие компоненты:

- описание всех элементов динамического и статического аспекта RUP;
- навигатор по всем элементам RUP, глоссарий и средство быстрого обучения технологии;
- руководства для всех участников проектной команды, охватывающие весь жизненный цикл ПО. Руководства представлены в двух видах: для осмысления процесса на верхнем уровне, и в виде подробных наставлений по повседневной деятельности;
- наставления по использованию инструментальных средств, входящих в состав Rational Suite;
- примеры и шаблоны проектных решений для Rational Rose;
- шаблоны проектной документации для SoDa;
- шаблоны в формате Microsoft Word, предназначенные для поддержки документации по всем процессам и действиям жизненного цикла ПО;
- планы в формате Microsoft Project, отражающие итерационный характер разработки ПО.

Адаптация RUP к потребностям конкретной организации или проекта обеспечивается с

помощью Rational Process Workbench (RPW) — специального набора инструментов и шаблонов для настройки и публикации Web-сайтов на основе RUP. RPW поддерживает три основные функции моделирования технологических процессов:

- определение процесса;
- описание процесса;
- представление процесса.

Библиотека элементов процесса содержит текстовую информацию о каждом элементе в модели процесса, все текстовые страницы RUP, а RPW — необходимые шаблоны для создания новых страниц описания. RPW генерирует описание процессов, включающее текст и графику, в виде Web-сайта, соединяя модели процессов и библиотеку описаний в единое целое.

RUP опирается на интегрированный комплекс инструментальных средств Rational Suite. Он существует в следующих вариантах:

- Rational Suite AnalystStudio — предназначен для определения и управления полным набором требований к разрабатываемой системе;
- Rational Suite DevelopmentStudio — предназначен для проектирования и реализации ПО;
- Rational Suite TestStudio — представляет собой набор продуктов, предназначенных для автоматического тестирования приложений;
- Rational Suite Enterprise — обеспечивает поддержку полного жизненного цикла ПО и предназначен как для менеджеров проекта, так и отдельных разработчиков, выполняющих несколько функциональных ролей в команде разработчиков.

В состав Rational Suite, кроме самой технологии RUP как продукта, входят следующие компоненты:

- Rational Rose — средство визуального моделирования (анализа и проектирования), использующее язык UML;
- Rational XDE — средство анализа и проектирования, интегрируемое с платформами MS Visual Studio .NET и IBM WebSphere Studio Application Developer;
- Rational Requisite Pro — средство управления требованиями, предназначенное для организации совместной работы группы разработчиков. Оно позволяет команде разработчиков создавать, структурировать, устанавливать приоритеты, отслеживать, контролировать изменения требований, возникающих на любом этапе разработки компонентов приложения;
- Rational Rapid Developer — средство быстрой разработки приложений на платформе Java 2 Enterprise Edition;
- Rational ClearCase — средство управления конфигурацией ПО;
- Rational SoDA — средство автоматической генерации проектной документации;
- Rational ClearQuest — средство для управления изменениями и отслеживания дефектов в проекте на основе средств e-mail и Web;
- Rational Quantify — средство количественного определения узких мест, влияющих на общую эффективность работы программы;
- Rational Purify — средство для локализации трудно обнаруживаемых ошибок времени выполнения программы;
- Rational PureCoverage — средство идентификации участков кода, пропущенных при тестировании;
- Rational TestManager — средство планирования функционального и нагрузочного тестирования;
- Rational Robot — средство записи и воспроизведения тестовых сценариев;
- Rational TestFactory — средство тестирования надежности;
- Rational Quality Architect — средство генерации кода для тестирования.

Одно из основных инструментальных средств комплекса Rational Rose [Кватрани-03] представляет собой семейство объектно-ориентированных CASE-средств и предназначено для автоматизации процессов анализа и проектирования ПО, а также для генерации кодов на различных языках и выпуска проектной документации. Rational Rose реализует процесс объектно-ориентированного анализа и проектирования ПО, описанный в RUP. В основе работы Rational Rose лежит построение диаграмм и спецификаций UML, определяющих архитектуру системы, ее статические и динамические аспекты. В составе Rational Rose можно выделить шесть основных структурных компонентов: репозиторий, графический интерфейс пользователя, средства просмотра проекта (браузер), средства контроля проекта, средства сбора статистики и генератор документов. К ним добавляются генераторы кодов для каждого поддерживаемого языка, состав которых меняется от версии к версии.

Репозиторий представляет собой базу данных проекта. Браузер обеспечивает «навига-

цию» по проекту, в том числе перемещение по иерархиям классов и подсистем, переключение от одного вида диаграмм к другому и т. д. Средства контроля и сбора статистики дают возможность находить и устранять ошибки по мере развития проекта, а не после завершения его описания. Генератор отчетов формирует тексты выходных документов на основе содержащейся в репозитории информации.

Средства автоматической генерации кода, используя информацию, содержащуюся в диаграммах классов и компонентов, формируют файлы описаний классов. Создаваемый таким образом скелет программы может быть уточнен путем прямого программирования на соответствующем языке (основные языки, поддерживаемые Rational Rose — C++ и Java).

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

- диаграммы UML, в совокупности представляющие собой модель разрабатываемой программной системы;
- спецификации классов, объектов, атрибутов и операций;
- заготовки текстов программ.

Тексты программ являются заготовками для последующей работы программистов. Состав информации, включаемой в программные файлы, определяется либо по умолчанию, либо по усмотрению пользователя. В дальнейшем эти исходные тексты развиваются программистами в полноценные программы.

Инструментальное средство Rational XDE представляет собой развитие возможностей Rational Rose в части синхронизации модели и кода (исключающей необходимость прямой и обратной генерации кода). Rational XDE обеспечивает:

- синхронизацию между кодом и моделью;
- отображение элементов кода Java и C# в UML;
- автоматическую синхронизацию с настраиваемым разрешением конфликтов;
- одновременное отображение кода и модели;
- постоянную синхронизацию модели UML, как части проекта Java или C#.

Технология Oracle

Методическую основу ТС ПО корпорации Oracle (www.oracle.com) составляет метод Oracle (Oracle Method) — комплекс методов, охватывающий большинство процессов ЖЦ ПО. В состав комплекса входят:

- CDM (Custom Development Method) — разработка прикладного ПО;
- PJM (Project Management Method) — управление проектом;
- AIM (Application Implementation Method) — внедрение прикладного ПО;
- BPR (Business Process Reengineering) — реинжиниринг бизнес-процессов;
- OCM (Organizational Change Management) — управление изменениями, и др.

Метод CDM оформлен в виде консалтингового продукта CDM Advantage — библиотеки стандартов и руководств (включающего также PJM). Он представляет собой развитие достаточно давно созданного Oracle CASE-Method, известного по использованию CASE-средств фирмы Oracle и книгам Р. Баркера. По существу, CDM является методическим руководством по разработке прикладного ПО с использованием инструментального комплекса Oracle Developer Suite, а сам процесс проектирования и разработки тесно связан с Oracle Designer и Oracle Forms.

В соответствии с CDM ЖЦ ПО формируется из определенных этапов (фаз) проекта и процессов, каждый из которых выполняется в течение нескольких этапов (рис. 2):

- стратегия (определение требований);
- анализ (формулирование детальных требований к системе);
- проектирование (преобразование требований в детальные спецификации системы);
- реализация (написание и тестирование приложений);
- внедрение (установка новой прикладной системы, подготовка к началу эксплуатации);
- эксплуатация.

На этапе стратегии определяются цели создания системы, приоритеты и ограничения, разрабатывается системная архитектура и составляется план разработки. На этапе анализа строятся модель информационных потребностей (диаграмма «сущность-связь»), диаграмма функциональной иерархии (на основе функциональной декомпозиции системы), матрица перекрестных ссылок и диаграмма потоков данных.

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

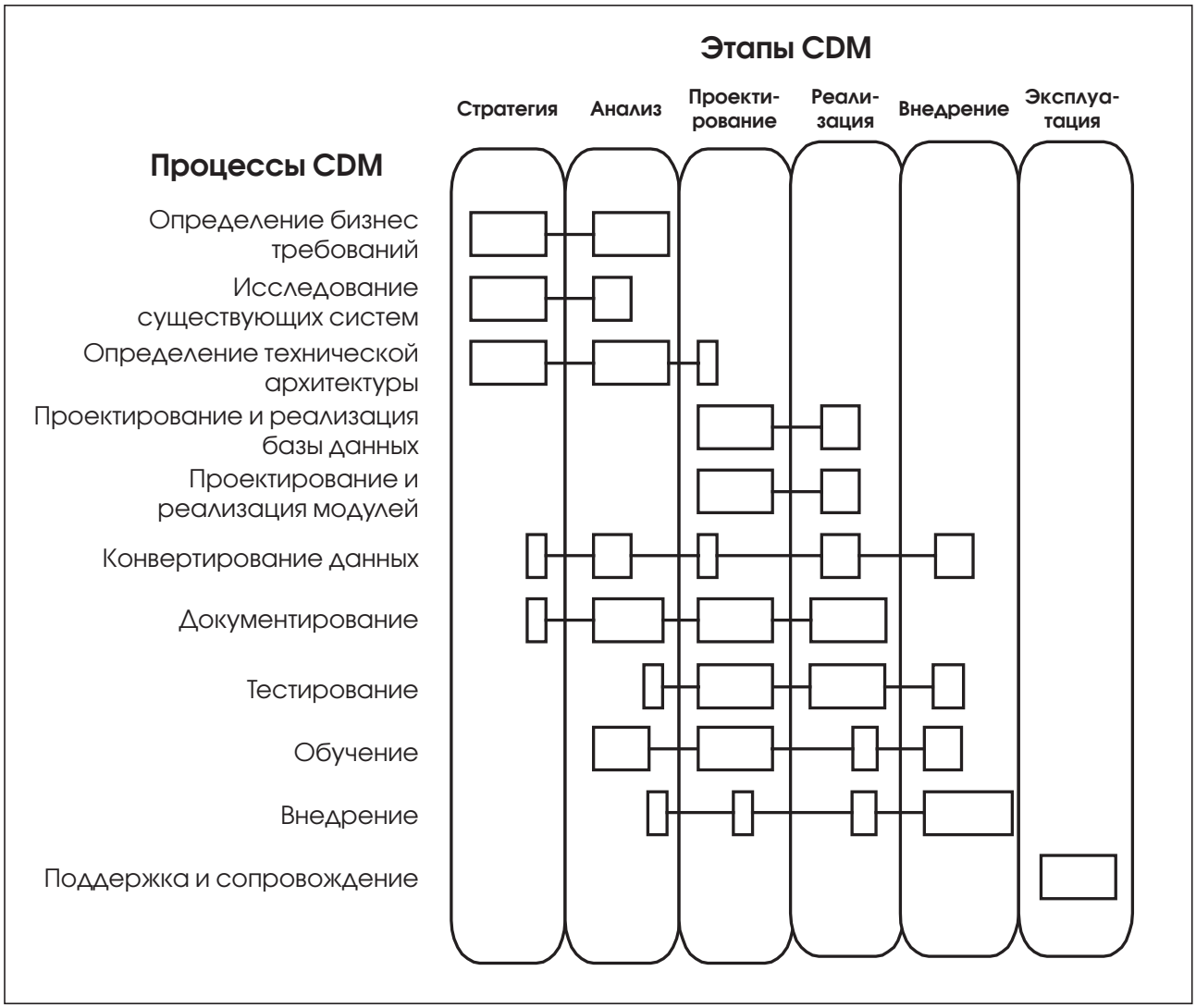

Рис. 2. Этапы и процессы CDM

На этапе реализации создается БД, строятся прикладные системы, производится их тестирование, проверка качества и соответствия требованиям пользователей. Создается системная документация, материалы для обучения и руководства пользователей.

На этапах внедрения и эксплуатации анализируются производительность и целостность системы, выполняется поддержка и, при необходимости, модификация системы.

Процессы CDM:

- определение бизнес-требований, или постановка задачи (Business Requirements Definition);
- исследование существующих систем (Existing Systems Examination). Выполнение этого процесса должно обеспечить понимание состояния существующего технического и программного обеспечения для планирования необходимых изменений;
- определение технической архитектуры (Technical Architecture);
- проектирование и реализация базы данных (Database Design and Build). Процесс предусматривает проектирование и реализацию реляционной базы данных, включая создание индексов и других объектов БД;
- проектирование и реализация модулей (Module Design and Build). Этот процесс является основным в проекте. Он включает непосредственное проектирование приложения и создание кода прикладной программы;
- конвертирование данных (Data Conversion). Цель этого процесса — преобразовывать, перенести и проверить согласованность и непротиворечивость данных, оставшихся в

наследство от «старой» системы и необходимых для работы в новой системе;
- документирование (Documentation);
- тестирование (Testing);
- обучение (Training);
- внедрение, или переход к новой системе (Transition). Этот процесс включает решение задач установки, ввода новой системы в эксплуатацию, прекращения эксплуатации старых систем;
- поддержка и сопровождение (Post-System Support).

Процессы состоят из последовательностей взаимосвязанных задач.

CDM предоставляет возможность выбрать требуемый подход к разработке. Это возможно, поскольку каждый процесс базируется на известных зависимостях между задачами одного типа и не зависит от того, на какие этапы будет разбит проект.

При определении подхода к разработке оценивается масштаб, степень сложности и критичность будущей системы. При этом учитываются стабильность требований, сложность и количество бизнес-правил, количество автоматически выполняемых функций, разнообразие и количество пользователей, степень взаимодействия с другими системами, критичность приложения для основного бизнес-процесса компании и целый ряд других.

В соответствии с этими факторами в CDM выделяются два основных подхода к разработке:

Классический подход (Classic). Этапы данного подхода представлены на рис. 2. Классический подход применяется для наиболее сложных и масштабных проектов, он предусматривает последовательный и детерминированный порядок выполнения задач. Для таких проектов характерно большое количество реализуемых бизнес-правил, распределенная архитектура, критичность приложения. Применение классического подхода также рекомендуется при нехватке опыта у разработчиков, неподготовленности пользователей, нечетко определенной задаче. Продолжительность таких проектов от 8 до 36 месяцев.

Подход быстрой разработки (Fast Track). Данный подход, в отличие от каскадного классического, является итерационным и основан на методе DSDM (Dynamic Systems Development Method). В этом подходе четыре этапа — стратегия, моделирование требований, проектирование и генерация системы и внедрение в эксплуатацию. Подход используется для реализации небольших и средних проектов с несложной архитектурой системы, гибкими сроками и четкой постановкой задач. Продолжительность проекта от 4 до 16 месяцев.

PJM — это определенная дисциплина ведения проекта, позволяющая гарантировать, что цели проекта, четко определенные в его начале, остаются в центре внимания на протяжении всего проекта. В основе PJM лежит метод, ориентированный на выполнение самостоятельных процессов (под процессом понимается набор связанных задач, выполнением которых достигается определенная цель проекта). Так же, как и CDM, метод руководства проектом представляется в виде четко определенной операционной схемы, в которой выделяются процессы, этапы, задачи, результаты решения задач и зависимости между задачами:
- Управление проектом и предоставление отчетности (Control and Reporting). Этот процесс содержит задачи, в результате решения которых определяются границы проекта и подход к разработке, происходит управление изменениями и контролируется возможный риск;
- Управление работой (Work Management). Процесс содержит задачи, помогающие контролировать работы, выполняемые в проекте;
- Управление ресурсами (Resource Management). Здесь решаются задачи, связанные с обеспечением каждого этапа исполнителями;
- Управление качеством (Quality Management). Процесс управления качеством гарантирует, что в проект отвечает требованиям пользователя в течение всего процесса разработки;
- Управление конфигурацией (Configuration Management).

Цикл решения задач PJM состоит из отдельных этапов. Количество этапов зависит от выбранного подхода к разработке. Задачи PJM можно распределить внутри каждого процесса по трем группам — задачи планирования, управления и завершения, и по уровням — отнести задачу на уровень проекта или на уровень отдельного этапа.

По аналогии с CDM, в PJM предусмотрено широкое использование шаблонов разрабатываемых документов.

Комплекс Oracle Developer Suite содержит набор интегрированных средств разработки для быстрого создания приложений. Он включает средства моделирования, программирования на Java, разработки компонентов, бизнес-анализа и составления отчетов. Все эти средства используют общие ресурсы, что позволяет совместно работать над одним проектом группе разработчиков. Oracle Developer Suite интегрирован с Oracle Database и Oracle Application Server, образуя единую платформу для создания и установки приложений.

Oracle Developer Suite поддерживает стандарты J2EE: Enterprise Java Beans (EJB), сервлеты и страницы JavaServer (JSP). В него также входят анализатор XML, процессор XSLT, процессор схем XML и XSQL-сервлет для разработки XML-приложений.

В Oracle Developer Suite встроена поддержка языка UML для разработки приложений на основе моделей. Модели хранятся в общем репозитории Oracle, который предназначен для поддержки больших коллективов разработчиков.

Oracle Developer Suite включает в себя:

- Oracle Designer — средство моделирования и генерации приложений;
- Oracle Forms — средство быстрой разработки приложений;
- Oracle Reports — визуальное средство разработки отчетов;
- Oracle JDeveloper — средство визуального программирования на языке Java;
- Oracle Discoverer — средство для разработки аналитических приложений;
- Oracle Warehouse Builder — система для построения хранилищ данных;
- Oracle Portal — средство разработки информационного портала организации.

CASE-средство Oracle Designer является интегрированным средством, обеспечивающим в совокупности со средствами разработки приложений поддержку ЖЦ ПО.

Oracle Designer представляет собой семейство методов и поддерживающих их программных продуктов. Базовый метод Oracle Designer (CDM) — структурный метод проектирования систем, охватывающий полностью все стадии ЖЦ ПО. Версия Oracle Designer для объектно-реляционной СУБД Oracle содержит также расширение в виде средств объектного моделирования, базирующихся на стандарте UML.

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

- Repository Administrator — средства управления репозиторием (создание и удаление приложений, управление доступом к данным со стороны различных пользователей, экспорт и импорт данных);
- Repository Object Navigator — средство доступа к репозиторию, обеспечивающее многооконный объектно-ориентированный интерфейс доступа ко всем элементам репозитория;
- Process Modeler — средство анализа и моделирования бизнес-процессов;
- Systems Modeler — набор средств построения функциональных и информационных моделей проектируемой системы, включающий средства для построения диаграмм «сущность-связь» (Entity-Relationship Diagrammer), диаграмм функциональных иерархий (Function Hierarchy Diagrammer), диаграмм потоков данных (Data Flow Diagrammer) и средство анализа и модификации связей объектов репозитория различных типов (Matrix Diagrammer);
- Systems Designer — набор средств проектирования ПО, включающий средство построения структуры реляционной базы данных (Data Diagrammer), а также средства построения диаграмм, отображающих взаимодействие с данными, иерархию, структуру и логику приложений, реализуемую хранимыми процедурами на языке PL/SQL (Module Data Diagrammer, Module Structure Diagrammer и Module Logic Navigator);
- Server Generator — генератор описаний объектов БД Oracle (таблиц, индексов, ключей, последовательностей и т.д.);
- Forms Generator — генератор приложений для Oracle Forms. Генерируемые приложения включают в себя различные экранные формы, средства контроля данных, проверки ограничений целостности и автоматические подсказки;
- Repository Reports — генератор стандартных отчетов, интегрированный с Oracle Reports.

Репозиторий Oracle Designer представляет собой хранилище всех проектных данных и может работать в многопользовательском режиме, обеспечивая параллельное обновление информации несколькими разработчиками. В процессе проектирования автоматически поддер-

живаются перекрестные ссылки между объектами словаря и могут генерироваться более 70 стандартных отчетов о моделируемой предметной области. Физическая среда хранения репозитория — база данных Oracle.

Технология Borland

Компания Borland (www.borland.com) в результате развития собственных разработок и приобретения целого ряда компаний представила интегрированный комплекс инструментальных средств, реализующих управление полным жизненным циклом приложений (Application Life Cycle Management, ALM). В соответствии с технологией Borland процесс создания ПО включает в себя пять основных этапов:

- определение требований;
- анализ и проектирование;
- разработка;
- тестирование и профилирование;
- развертывание.

Выполнение всех этапов координируется процессом управления конфигурацией и изменениями.

Определение требований реализуется с помощью системы управления требованиями CaliberRM, которая стала частью семейства продуктов Borland в результате покупки компании Starbase. CaliberRM сохраняет требования в базе данных, документы с их описанием создаются с помощью встроенного механизма генерации документов MS Word на базе заданных шаблонов. Система обеспечивает экспорт данных в таблицы MS Access и импорт из MS Word. CaliberRM поддерживает различные методы визуализации зависимостей между требованиями, с помощью которых пользователь может ограничить область анализа, необходимого в случае изменения того или иного требования. Имеется модуль, который использует данные требования для оценки трудозатрат, рисков и расходов, связанных с реализацией требований.

Средство анализа и проектирования Together ControlCenter [Кармайкл-03] разработано компанией TogetherSoft. В основе его применения лежит один из вариантов подхода «Быстрой разработки ПО» под названием Feature Driven Development (FDD) [Палмер-02].

Together ControlCenter — интегрированная среда проектирования и разработки, поддерживающая визуальное моделирование на UML с последующим написанием приложений для платформ J2EE (Java) и .Net (C#, C++ и Visual Basic). Кроме базовой версии, имеется уменьшенный вариант системы для индивидуальных разработчиков и небольших групп (Together Solo), а также редакции для платформы IBM WebSphere и среды разработки Jbuilder.

В системе реализована технология LiveSource, которая обеспечивает синхронизацию между проектом приложения и изменениями — при внесении изменений в исходные тексты меняется модель программы, а при изменении модели надлежащим образом изменяется текст на языке программирования. Это исключает необходимость вручную модифицировать модель или переписывать код. Контроль версий осуществляется благодаря функциональной интеграции Together и системы StarTeam. Поддерживается также интеграция с системой управления конфигурацией Rational ClearCase.

Инструментальные средства тестирования появилась в составе комплекса Borland в результате покупки компании Optimizeit. К ним относятся Optimizeit Suite 5, Optimizeit Profiler for .NET и Optimizeit ServerTrace. Первые две системы позволяют выявить потенциальные проблемы использования аппаратных ресурсов — памяти и процессорных мощностей на платформах J2EE и .Net соответственно. Интеграция Optimizeit Suite 5 в среду разработки Jbuilder, а Optimizeit Profiler — в C#Builder и Visual Basic .Net позволяет проводить контрольные испытания приложений по мере разработки и ликвидировать узкие места производительности. Система Optimizeit ServerTrace предназначена для управления производительностью серверных J2EE-приложений с точки зрения достижения заданного уровня обслуживания и сбора контрольных данных по виртуальным Java-машинам.

Сущность концепции ALM сосредоточена в системе управления конфигурацией и изменениями: именно она объединяет основные фазы ЖЦ ПО. Такой системой является StarTeam, разработанная компанией Starbase. Она выполняет функции контроля версий, управления изменениями, отслеживания дефектов, управления требованиями (в интеграции с CaliberRM), управления потоком задач и управления проектом.

StarTeam совместима с интерфейсом Microsoft Source Code Control и интегрируется с любой системой разработки, которая поддерживает этот API. Кроме того, в системе реализованы средства интеграции со средствами разработки и моделирования Together, JBuilder, Delphi, C++Builder и C#Builder.

В технологии Borland выделяется три уровня интеграции. **Функциональная (touch-**

point) интеграция позволяет обратиться из одной системы к функциям другой, выбрав соответствующий пункт меню. Например, интерфейс управления изменениями StarTeam непосредственно отображается в системах Together, C#Builder и Visual Studio .Net. Такая интеграция дает возможность разделять информацию между системами, но не обеспечивает единого рабочего пространства, вынуждает пользователя переключать окна и приводит к дублированию процессов управления структурой проекта. **Встроенная (embedded)** интеграция обеспечивает работу с одной системой непосредственно в среде другой. Например, не выходя из среды разработки Jbuilder, можно просматривать графики производительности, которые создает система Optimizeit. Самый высокий уровень интеграции — **синергетический (synergistic)**, позволяющий сочетать функции двух различных продуктов незаметно для разработчиков. Для большинства продуктов Borland и других поставщиков синергетическая интеграция пока остается делом будущего, однако ее принципы уже начинают реализовываться.

Технология Computer Associates

Компания Computer Associates (www.ca.com) предлагает комплексы инструментальных средств поддержки различных процессов ЖЦ ПО:

- AllFusion Modeling Suite — интегрированный комплекс CASE-средств [Маклаков-03], включающий следующие продукты:
  - AllFusion Process Modeler (BPwin) — функциональное моделирование;
  - AllFusion ERwin Data Modeler (ERwin) — моделирование данных;
  - AllFusion Component Modeler (Paradigm Plus) — объектно-ориентированный анализ и проектирование с использованием UML и возможностью генерации кода;
  - AllFusion Model Manager (Model Mart) — организация совместной работы команды разработчиков;
  - AllFusion Data Model Validator (ERwin Examiner) — проверка структуры и качества моделей данных.
- AllFusion Change Management Suite — комплекс средств управления конфигурацией и изменениями.
- AllFusion Process Management Suite — средства управления процессами и проектами для различных типов приложений.

CASE-средства ERwin и BPwin были разработаны фирмой Logic Works, которая в 1998 году вошла в состав PLATINUM Technology, а затем Computer Associates.

BPwin — средство моделирования бизнес-процессов, реализующее метод IDEF0, а также поддерживающее диаграммы потоков данных и IDEF3. В процессе моделирования BPwin позволяет переключиться с нотации IDEF0 на любой ветви модели на нотацию IDEF3 или DFD и создать смешанную модель. BPwin поддерживает функционально-стоимостной анализ (ABC).

Семейство продуктов ERwin представляет собой набор средств концептуального моделирования данных, использующих метод IDEF1X. ERwin реализует проектирование схемы БД, генерацию ее описания на языке целевой СУБД (Oracle, Sybase, DB2, Microsoft SQL Server и др.) и реверсный инжиниринг существующей БД. ERwin выпускается в нескольких конфигурациях, ориентированных на наиболее распространенные средства разработки приложений.

Для управления групповой разработкой используется средство Model Mart, обеспечивающее многопользовательский доступ к моделям, созданным с помощью ERwin и BPwin. Модели хранятся на центральном сервере и доступны для всех участников группы проектирования.

Model Mart удовлетворяет ряду требований, предъявляемым к средствам управления разработкой крупных систем, а именно:

- Совместное моделирование. Каждый участник проекта имеет инструмент поиска и доступа к интересующей его модели в любое время. При совместной работе используются три режима: незащищенный, защищенный и режим просмотра. В режиме просмотра запрещается любое изменение моделей. В защищенном режиме модель, с которой работает один пользователь, не может быть изменена другими пользователями. В незащищенном режиме пользователи могут работать с общими моделями в реальном масштабе времени.
- Создание библиотек решений. Model Mart позволяет формировать библиотеки стандартных решений, включающие наиболее удачные фрагменты реализованных проектов, накапливать и использовать типовые модели, объединяя их при необходимости «сборки» больших систем. На основе существующих баз данных с помощью ERwin возможно восстановление моделей (реверсный инжиниринг), которые в процессе ана-

лиза пригодности их для новой системы могут объединяться с типовыми моделями из библиотек моделей.

- Управление доступом. Для каждого участника проекта определяются права доступа, в соответствии с которыми, они получают возможность работать только с определенными моделями. Права доступа могут быть определены как для групп, так и для отдельных участников проекта. Роль специалистов, участвующих в различных проектах может меняться, поэтому в Model Mart можно определять и управлять правами доступа участников проекта к библиотекам, моделям и даже к специфическим областям модели.

ти значительные выгоды разработчикам. В то же время проблема обоснованного выбора и эффективного применения ТС ПО в крупномасштабных проектах остается актуальной. Невозможно достичь удовлетворительных результатов от применения даже самых совершенных технологий, если они применяются бессистемно, разработчики не обладают необходимой квалификацией для работы с ними, и сам проект выполняется и управляется хаотически. Систематический, обоснованный подход к выбору и применению ТС ПО может сократить время и повысить качество разработки ПО, обеспечить высокую степень его независимости от конкретных разработчиков, а также снизить затраты на разработку и сопровождение ПО.

Заключение

По прогнозам IDC [IDC-2003-2], рынок ТС ПО, испытавший определенный кризис в 2002 году, в ближайшее пятилетие ожидает устойчивый рост в среднем на 6,3% в год. Определяющим фактором для развития этой тенденции является стремление компаний-разработчиков повысить продуктивность своей работы, сократить сроки вывода новых продуктов на рынок, контролировать расходы и быстро получать отдачу от инвестиций. Достижению этих целей способствует использование сред разработки, позволяющих снизить сложность процессов создания ПО, увеличить их эффективность, уменьшить затраты на разработку и максимально использовать потенциал новых технологий. Аналитики сходятся на том, что основное направление развития инструментальных средств — их сквозная интеграция, переход от частично интегрированных средств к интегрированным комплексам, объединяющим возможности управления требованиями, моделирования, разработки, тестирования, управления конфигурацией и изменениями и развертывания приложений. В ближайшие годы такие комплексы, помимо перечисленных возможностей будут включать в себя средства управления потоками работ и проектами. Рынок таких инструментальных средств ожидает глобальная консолидация, обещающая принес-

Библиография

[Брукс-99] Брукс Ф. Мифический человеко-месяц или как создаются программные системы: Пер. с англ. — СПб.: Символ-Плюс, 1999.

[Буч-99] Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++. 2-е изд.: Пер. с англ. — М.: Издательство Бином, СПб.: Невский диалект, 1999.

[Буч-2000] Буч Г., Рамбо Дж., Джекобсон А. Язык UML. Руководство пользователя.: Пер. с англ. — М.: ДМК, 2000.

[Вендров-2000] Вендров А.М. Проектирование программного обеспечения экономических информационных систем: Учебник. — М.: Финансы и статистика, 2000.

[Гома-02] Гома Х. UML. Проектирование систем реального времени, распределенных и параллельных приложений.: Пер. с англ. — М.: ДМК, 2002.

[Калашян-03] Калашян А.Н., Калянов Г.Н. Структурные модели бизнеса: DFD-технологии. — М.: Финансы и статистика, 2003.

[Каменнова-01] Каменнова М., Громов А., Ферапонтов М., Шматалюк А. Моделирование бизнеса. Методология ARIS. — М.: Вести-МетаТехнология, 2001.

[Кармайкл-03] Кармайкл Э., Хейвуд Д. Быстрая и качественная разработка программного обеспечения.: Пер. с англ.: — М.: Вильямс, 2003.
[Кватрани-03] Кватрани Т. Визуальное моделирование с помощью Rational Rose 2002 и UML..: Пер. с англ. — М.: Вильямс, 2003.
[Коберн-02-1] Коберн А. Быстрая разработка программного обеспечения.: Пер. с англ. — М.: ЛОРИ, 2002.
[Коберн-02-2] Коберн А. Современные методы описания функциональных требований к системам.: Пер. с англ. — М.: ЛОРИ, 2002.
[Конноли-03] Конноли Т., Бегг К. Базы данных: проектирование, реализация и сопровождение. Теория и практика. 3-е издание.: Пер. с англ.: — М.: Вильямс, 2003.
[Крачтен-02] Крачтен Ф. Введение в Rational Unified Process.: Пер. с англ. — М.: Вильямс, 2002.
[Ларман-02] Ларман К. Применение UML и шаблонов проектирования. 2-е издание.: Пер. с англ. — М.: Вильямс, 2002.
[Леффингуэлл-02] Леффингуэлл Д., Уидриг Д. Принципы работы с требованиями к программному обеспечению. Унифицированный подход.: Пер. с англ.: — М.: Вильямс, 2002.
[Маклаков-03] Маклаков С.В. Создание информационных систем с AllFusion Modeling Suite. — М.: Диалог-МИФИ, 2003.
[Марка-93] Марка Д.А., МакГоуэн К. Методология структурного анализа и проектирования. — М.: МетаТехнология, 1993.
[Палмер-02] Палмер С.Р., Фелсинг Дж.М. Практическое руководство по функционально-ориентированной разработке ПО.: Пер. с англ.: — М.: Вильямс, 2002.
[Рамбо-02] Рамбо Дж., Буч Г., Якобсон А. UML. Специальный справочник: Пер. с англ. — СПб: Питер, 2002.
[Розенберг-02] Розенберг Д., Скотт К. Применение объектно-ориентированного моделирования с использованием UML и анализ прецедентов.: Пер. с англ. — М.: ДМК, 2002.
[Фаулер-99] Фаулер М., Скотт К. UML в кратком изложении. Применение стандартного языка объектного моделирования.: Пер. с англ. — М.: Мир, 1999.
[Черемных-01] Черемных С.В., Семенов И.О., Ручкин В.С. Структурный анализ систем: IDEF-технологии. — М.: Финансы и статистика, 2001.
[Eriksson-2000] Eriksson, Hans-Erik and Penker, Magnus. Business Modeling with UML: Business Patterns at work. Wiley Computer Publishing, 2000.
[IDC-2002] Worldwide Analysis, Modeling, and Design Tools Forecast and Analysis, 2002 - 2006. IDC, 2002. http://www.idc.com.
[IDC-2003-1] Worldwide Analysis, Modeling, Design and Construction Tools Competitive Analysis, 2003: 2002 Shares and Current Outlook. IDC, 2003. http://www.idc.com.
[IDC-2003-2] Worldwide Application Development and Deployment Forecast Summary, 2003 — 2007. IDC, 2003. http://www.idc.com.
[IEEE-1992] IEEE Std 1209-1992. IEEE Recommended Practice for the Evaluation and Selection of CASE Tools.
[IEEE-1995] IEEE Std 1348-1995. IEEE Recommended Practice for the Adoption of Computer-Aided Software Engineering (CASE) Tools.
[ISO-1995] ISO/IEC 14102:1995(E). Information technology — Guideline for the evaluation and selection of CASE Tools.

Jet Info
ИНФОРМАЦИОННЫЙ БЮЛЛЕТЕНЬ
Издается с 1995 года
Издатель: компания Джет Инфо Паблишер

Главный редактор: Дмитриев В.Ю. *(vlad@jet.msk.su)*
Технический редактор: Овчинникова Г.Ю. *(galya@jet.msk.su)*
Россия, 127015, Москва, Б. Новодмитровская, 14/1
тел. (095) 411 76 01
факс (095) 411 76 02
email: JetInfo@jet.msk.su http://www.jetinfo.ru

Подписной индекс по каталогу Роспечати **32555**