**ВВЕДЕНИЕ**

На пути к достижению комплексного подхода при разработке программных средств (ПС) широкое применение получили CASE-продукты (Computer Aided Software Engineering), обеспечивающие поддержку многочисленных технологий проектирования информационных систем, охватывая всевозможные средства автоматизации и весь жизненный цикл (ЖЦ) программного обеспечения (ПО). Диа-азон CASE-средств очень велик, и сегодня практически каждое из них располагает мощной инструментальной базой. CASE-технология включает в себя методологию анализа, проектирования, разработки и сопровождения сложных систем ПО, поддержанную комплексом взаимосвязанных средств автоматизации. Главная цель CASE-подхода – разделить и максимально автоматизировать все этапы разработки ПС. Основные преимущества применения CASE-средств:

– улучшение качества ПО за счет автоматического контроля проекта;

– возможность быстрого создания прототипа будущей системы, что позволяет уже на ранних стадиях разработки оценить результат;

– ускорение процессов проектирования и программирования;

– освобождение разработчиков от выполнения рутинных операций;

– возможность повторного использования ранее созданных компонентов.

Методы объектно-ориентированного анализа и проектирования включают в себя язык моделирования и описание процессов моделирования. Язык моделирования UML (Unified Modeling Language) – это нотация, которая используется методом для описания проектов. Основная идея UML – это возможность моделировать ПО и другие системы как наборы взаимодействующих объектов. UML не привязан к какой-либо конкретной методологии или ЖЦ и может использоваться со всеми существующими методологиями. В настоящее время UML принят в качестве стандартного языка моделирования и получил широкую поддержку в индустрии ПО. Язык взят на вооружение самыми известными производителями ПО: IBM, Microsoft, Hewlett-Packard, Oracle. Большинство современных CASE-средств разрабатывается на основе UML. На сегодняшний день практически все ведущие компании–разработчики технологий и программных продуктов располагают развитыми технологиями создания ПО. В пособии рассмотрена одна из признанных технологий, претендующих на роль мирового корпоративного стандарта, – Rational Unified Process (RUP), а также объектно-ориентированное CASE-средство Rational XDE. Главное отличие это средства от своего предшественника, программы Rational Rose, – полная интеграция с платформой Microsoft Visual Studio.NET, позволяющая в одной оболочке работать как с моделями программной системы, так и с кодом. За счет этого заметно сократилось время синхронизации модели и программного кода, в результате чего повысилась производительность рабочей станции.

**1. CОВРЕМЕННЫЕ ТЕХНОЛОГИИ СОЗДАНИЯ ПРОГРАММНЫХ СРЕДСТВ**

**1.1. Технология Rational Unified Process**

RUP-технология в значительной степени соответствует стандартам и нормативным документам, связанным с ЖЦ ПО . Ее основными принципами являются:

– итерационный и инкрементный подход к созданию ПО;

– планирование и управление проектом на основе функциональных требований к системе;

– 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 в себя 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, поэтому такой подход называют итерационным и инкрементным. На рис. 1.1 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;

– стадия разработки;

– стадия конструирования;

– стадия ввода в действие.

Каждая стадия завершается в четко определенной контрольной точке. 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 она связана с всесторонним изучением всех возможностей реализации проекта. В это же время разрабатывается бизнес-план проекта: определяется, сколько приблизительно он будет стоить и какой доход принесет. Кроме того, выполняется 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;

– начальный проектный глоссарий;

– 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:

– завершенная модель вариантов использования, 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 в себя:

– 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.

Созданная архитектура является основой всей дальнейшей разработки. В будущем неизбежны незначительные изменения в деталях архитектуры, однако серьезные изменения маловероятны. Стадия разработки занимает около пятой части общей продолжительности проекта. Основными признаками ее завершения являются следующие:

– разработчики в состоянии оценить, 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.

Стадия конструирования напрямую связана с проработкой итераций. На стадии конструирования они являются одновременно инкрементными и повторяющимися. Инкрементность связана с добавлением новых конструкций к вариантам использования, реализованным во время предыдущих итераций. Повторяемость относится 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 и содержащий следующее:

– ПО, интегрированное на требуемых платформах;

– руководства пользователя;

– описание текущей реализации.

Стадия ввода в действие связана с передачей готового продукта в распоряжение пользователей. Она включает в себя:

– бета-тестирование, позволяющее убедиться, что новая система соответствует ожиданиям пользователей;

– параллельное функционирование с существующей 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. Роль 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 соответствует понятию технологической операции. Он имеет четко определенную цель, обычно выражаемую в терминах получения или модификации некоторых рабочих продуктов, таких, как модель, элемент модели, документ, исходный код или план. Каждый вид деятельности связан 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, представляющих собой методики выполнения технологических операций.

Дисциплина 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.

Имеется три вспомогательные дисциплины: управление конфигурацией и изменениями; управление проектом; создание инфраструктуры.

**1.2. Унифицированный язык моделирования UML**

Создание UML началось в 1994 г., в 1995 г. появилась первая спецификация языка, а в настоящее время UML является общепризнанным стандартом моделирования. В UML-модели есть два аспекта :

– статическая структура – описывает, какие типы объектов важны для моделирования системы и как они взаимосвязаны;

– динамическое поведение – описывает ЖЦ этих объектов и то, как они взаимодействуют друг с другом для обеспечения требуемой функциональности системы.

Основные цели создания унифицированного языка моделирования:

1. Предоставить пользователям готовый к применению выразительный язык визуального моделирования, позволяющий разрабатывать осмысленные модели и обмениваться ими.

2. Предусмотреть механизмы для расширения базовых концепций.

3. Обеспечить независимость UML от конкретных языков программирования и процессов разработки.

4. Создать формальную основу для понимания языка моделирования.

5. Стимулировать рост рынка объектно-ориентированных инструментальных средств.

6. Интегрировать лучший практический опыт.

Семантика языка UML представляет собой некоторую метамодель, которая определяет абстрактный синтаксис и семантику понятий объектного моделирования на языке UML. Семантика определяется для двух видов объектных моделей: структурных моделей и моделей поведения. Структурные модели, известные также как статические модели, описывают структуру сущностей или компонентов некоторой системы, включая их классы, интерфейсы, атрибуты и отношения. Модели поведения, называемые иногда динамическими моделями, описывают поведение или функционирование объектов системы, включая их методы, взаимодействие и сотрудничество между ними, а также процесс изменения состояний отдельных компонентов и системы в целом.

Для решения столь широкого диапазона задач моделирования разработана достаточно полная семантика для всех компонентов графической нотации. Требования семантики языка UML конкретизируются при построении отдельных видов диаграмм.

**1.3. Основные окна и пункты меню программы Rational XDE**

Рассмотрим программное средство Rational XDE – один из признанных лидеров на мировом рынке CASE-продуктов. Программа потеряла часть своей универсальности по сравнению с Rational Rose [4, 5]. В основном потери касались возможности генерации программного кода практически для любых языков программирования. В Rational XDE для .NET возможна синхронизация модели и кода только для языков, которые поддерживаются Microsoft Visual Studio.NET. При этом Rational XDE позволяет создавать Free Form, в которых не отслеживается нотация UML и которые могут включать в себя значки из разных UML-диаграмм .

Язык С#, на котором генерируется код в среде Rational XDE, разработан компанией Microsoft для платформы .NET. Она представляет собой обширную библиотеку классов, инфраструктуру и инструментальные средства для создания межплатформенных, не зависящих от языка программирования приложений. На платформе .NET создана ASP.NET – технология активных серверных страниц. В приложении ASP.NET доступна вся библиотека .NET Framework, значительно ускоряющая и облегчающая разработку сложных сетевых программных систем .

На рис. 1.1 показано главное окно программы Rational XDE. На его внешнем виде отразилась интеграция с Microsoft Visual Studio.NET. В результате рабочий стол Visual Studio.NET изменился и на него добавились окна, отвечающие за моделирование программной системы. В центре экрана расположено окно документов, в котором можно открывать код, ресурсы и диаграммы UML, создаваемые в модели. Справа в окне Explorer добавилась закладка Model Explorer, позволяющая перемещаться по модели. Под ней – окно XDE Code Properties, которое показывает свойства выбранной диаграммы. Внизу – закладка Model Documentation, отражающая документацию модели. Слева, в окне Toolbox, добавилось большое количество инструментов, необходимых для работы с UML-моделями в отдельных разделах, которые появляются в момент активизации одной из диаграмм .

Основная работа с элементами модели осуществляется при помощи окон Model Explorer и рабочего стола диаграммы. При этом за перемещение по модели отвечает Model Explorer, а редактирование лучше выполнять на рабочем столе. Одним из отличий Rational XDE от предыдущей версии явилось использование окна Toolbox, содержащего дополнительные инструменты для работы над проектом. Строки инструментов (Toolbar) также остались, но их роль сократилась до управления основными режимами, а создание новых элементов теперь производится с использованием Toolbox.

Model Explorer позволяет осуществлять навигацию по элементам модели, представленным в иерархическом виде. Из контекстного меню можно добавлять, удалять и изменять как диаграммы UML, так и элементы диаграмм, синхронизировать исходный код и диаграммы, работать с шаблонами кода, т.е.

полностью управлять созданием модели программной системы. Model Explorer представляет собой аналог стандартного обозревателя Windows.

В окне Model Documentation отображается текст документации, относящийся к модели, диаграмме или выделенному элементу на ней. Окно докумен тации представляет собой текстовый редактор, позволяющий описывать цели создания, поведение и другую информацию. Документация в Rational XDE обновляется вместе с моделью. В случае если разработчик внес в исходный код комментарий, а после этого модель обновилась, то все комментарии сразу же будут отображены в окне документации.

Окно Toolbox сменило строку инструментов Toolbar. С помощью Toolbox выполняются основные функции по работе с элементами на диаграмме.

Аналогично тому, как это происходило в Rational Rose, в среде Rational XDE все действия над объектами выполняются посредством контекстного меню. Оно зависит от набора функций, доступных для применения к конкретному объекту. При установке Rational XDE добавляет свои пункты в главное меню Visual Studio .NET, изменяет некоторые пункты, установленные по умолчанию, а также добавляет свои строки инструментов. Это связано с тем, что после установки Rational XDE в среде Visual Studio .NET можно работать с графическими диаграммами, а не только с программным кодом и ресурсами проекта. Рассмотрим пункты меню, добавленные Rational XDE, и предоставляемые ими возможности по работе с моделями или их элементами.

Modeling. Этот раздел меню предназначен для добавления диаграмм и их элементов в проект. Меню Modeling показано на рис. 1.2. Пункты моделирования позволяют добавлять в проект UML-элементы или диаграммы; проверять корректность диаграмм; проверять из проекта ссылки на другие модели; исправлять ошибки во внешних ссылках; устанавливать пути доступа к файлам модели; использовать шаблоны при построении модели.

Diagram. Этот раздел меню предназначен для общего управления значками на диаграммах. Меню Diagram показано на рис. 1.3. Его пункты позволяют: автоматически расставить значки на текущей диаграмме; изменить на диаграмме положение выделенного элемента; переместить фокус просмотра на выделенный элемент диаграммы; добавить на диаграмму элементы, связанные с выделенным; активизировать окно настройки визуализации соединений; работать с надписями на стрелках соединителях; управлять перерисовкой диаграммы и найти отмеченный на ней элемент .

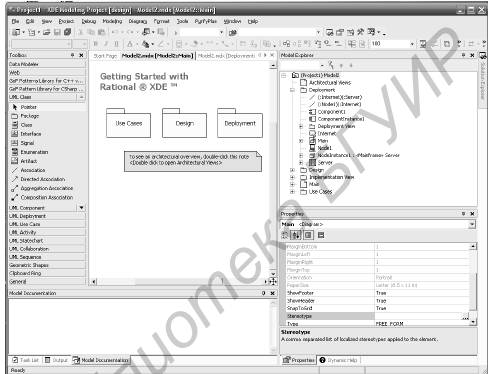


Рис. 1.1. Главное окно Rational XDE

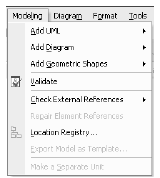


Рис. 1.2. Меню Modeling

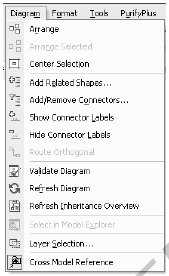


Рис. 1.3. Меню Diagram

Format. Этот раздел меню позволяет управлять форматированием диаграммы. Меню Format показано на рис. 1.4. Его пункты позволяют показать или скрыть список атрибутов, операций и сигналов в классе; показать или скрыть сигнатуры операций или сигналов; изменять стиль визуализации элементов диаграмм и их стереотипов; изменять стиль выделенной линии; поддерживать заданный размер элемента диаграммы; включить или отключить показ имени родительского контейнера для выделенного элемента; установить одинаковые настройки для всех выделенных элементов диаграммы.

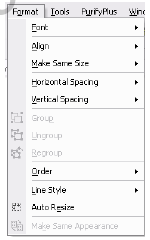


Рис. 1.4. Меню Format

**2. ЛАБОРАТОРНЫЕ РАБОТЫ**

**2.1. Лабораторная работа №1**

**АНАЛИЗ ТРЕБОВАНИЙ И ПОСТРОЕНИЕ ДИАГРАММЫ USE CASE**

**Цель работы:**

* научиться строить диаграммы Use Case в среде автоматизированного синтеза Rational XDE;
* разработать диаграмму Use Case для проектируемой системы.

**Задание**: описать функциональные требования к системе и представить сценарии поведения ее объектов с помощью диаграммы Use Case.

**Порядок выполнения работы**

1. Изучить теоретическую часть лабораторной работы.

2. Ответить на контрольные вопросы.

3. Построить диаграмму Use Case по предложенной тематике.

**Построение диаграммы Use Case**

Моделирование системы начинается с анализа требований к ней, что напрямую связано с определением прецедентов в системе. Прецеденты и связи между ними могут быть представлены с помощью диаграммы Use Case.

Рассмотрим построение модели виртуального книжного магазина, начав с выделения актеров и их ролей в системе.

1. Покупатель книги – любой пользователь сети интернет, зарегистрировавшийся в магазине как покупатель.

2. Администратор магазина – работник, который проверяет наличие заказов пользователей, формирует их, если они есть на складе, и отправляет покупателям с посыльным.

3. Директор магазина – получает отчет о заказанных и отправленных потребителям книгах.

Тогда можно описать функции, которые должна выполнять программа.

1. Просмотр книжного каталога (все пользователи).

2. Регистрация пользователей.

3. Работа с корзиной покупателя (зарегистрированные пользователи).

4. Оформление заказа на покупку (зарегистрированные пользователи).

5. Просмотр статуса заказа (для зарегистрированных пользователей).

6. Изменение статуса заказа (для администратора).

7. Просмотр списка заказов (для руководителя).

8. Редактирование книжного каталога (для администратора).

9. Изменение данных пользователя (для администратора и зарегистрированного пользователя).

Все диаграммы в Rational XDE создаются в модели. Ее построение начинается с создания проекта приложения, для чего необходимо выполнить следующие действия .

1. Выбрать пункт меню File=>New.

2. В предложенном окне выбрать Blank Model и создать пустую модель, после чего в нее можно добавлять диаграммы.

3. Из контекстного меню модели выбрать пункт Add Diagram.

4. Из подменю выбрать Add Use Case.

5. Переименовать добавленную диаграмму в Прецеденты.

В Rational XDE не включены специальные значки для бизнес-анализа, что отражает сокращенный процесс разработки .NET приложений, однако это не мешает полноценно использовать диаграмму прецедентов для определения требований к системе. Все инструменты диаграммы Use Case активны и находятся в окне Toolbox. Их набор приведен на рис. 2.1.

Значок Access позволяет показать зависимость одного элемента диаграммы от другого на уровне доступа.

Значок Include отражает прецедент, являющийся частью главного прецедента.

Значок Import показывает зависимость одного элемента диаграммы от другого, когда один элемент импортирует информацию из другого.

Значок Extend отражает расширение прецедента и используется, чтобы показать часть главного прецедента, который обрабатывается не всегда, а в определенных случаях или при определенных условиях.

Значком Association обозначают простые связи между элементами.



Рис. 2.1. Toolbox для диаграммы Use Case

Значок Direct Association позволяет обозначать направленные связи между элементами. Эта связь более сильная, чем простая ассоциация, и позволяет точнее показать отношения между элементами на диаграмме. Например, отразить актера, который инициализирует прецедент. Чтобы преобразовать Association в Direct Association, необходимо выполнить следующее:

1. Выделить связь.

2. Из контекстного меню связи выбрать пункт Properties Window.

3. В окне свойств перейти к строке End2Navigable и изменить ее значение на False.

Значок Constraint применяется для указания ограничений, налагаемых на прецеденты.

Значок Constraint Attachment позволяет соединить элемент Constraint с любым элементом на диаграмме.

На основе ранее описанных ролей актеров и функций системы строится диаграмма прецедентов. Ее окончательный вариант приведен на рис. 2.2.

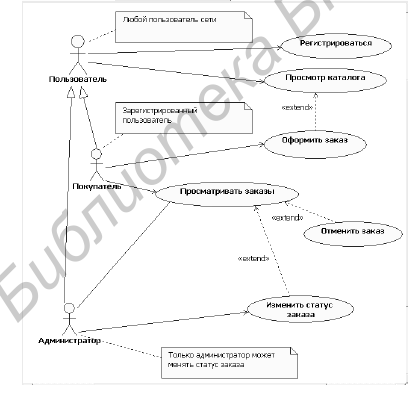


Рис. 2.2. Диаграмма Use Case (прецедентов)

Здесь показаны три актера и их роли. Пользователь может зарегистрироваться и просматривать каталог. После регистрации он становится покупателем и может оформлять, просматривать и отменять заказы. Администратор также может просматривать заказы или менять их статус. На диаграмме использованы направленные ассоциации для отражения инициализации прецедентов актерами и простые ассоциации – в остальных случаях. Прецеденты оформления заказа, изменения статуса и удаления являются расширением соответствующих прецедентов, что отражено связью Extend.

**Контрольные вопросы**

1. Какой вариант не позволит создать новый элемент Actor на диаграмме прецедентов:

а) находясь в окне диаграммы Use Case, выбрать из контекстного меню Add UML => Actor;

б) щелкнуть по значку Actor в Toolbox, а затем на диаграмме;

в) находясь в окне Model Explorer, из контекстного меню диаграммы Use Case выбрать Add UML => Actor.

2. Есть прецедент, который инициализируется актером. Какой тип связи обычно используется для их соединения: a) Association; б) Direct Association; в) Dependency.

3. Есть два прецедента, один из которых возникает при определенных обстоятельствах при выполнении другого. Какую связь нужно использовать для отражения такого взаимодействия: a) Include; б) Dependency; в) Extend.