Министерство образования и науки Российской Федерации

Южно-Российский государственный технический университет
(Новочеркасский политехнический институт) имени М.И. Платова

ПРОГРАММНАЯ ИНЖЕНЕРИЯ

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

учебно-методическое пособие к лабораторным работам

для направления подготовки 09.03.03 «Прикладная информатика»

Новочеркасск
ЮРГТУ(НПИ)
2013

УДК 004.4 (075.8)

Рецензент: кандидат технических наук, доцент Гринченков Д.В.
(ЮРГПУ(НПИ))

Составитель - **В.А.Зуев**

Программная инженерия. Лабораторный практикум: учебно-методическое пособие к лабораторным работам/В.А. Зуев; Южно-Российский государственный технический университет (НПИ) имени М.И. Платова – Новочеркасск: ЛиК, 2013. – 52с.

Представлены материалы для проведения лабораторного практикума по дисциплине «Программная инженерия». Содержит методические рекомендации по применению современных инструментальных средств на этапе проектирования программного обеспечения на основе UML, тестирования, отладки программ и получения метрик кода в среде MS Visual Studio.

Пособие предназначено для студентов, обучающихся по направлению подготовки 09.03..03 Прикладная информатика всех форм обучения.

УДК 004.4(075.8)

Содержание

ВВЕДЕНИЕ

Дисциплина «Программная инженерия» относится к дисциплинам базовой части учебного плана. Она формирует базовые знания проектирования качественных программных систем (ПС) в различных областях приложений. Для решения этих задач требуется грамотная организация процесса создания ПС, реализация технологических принципов и методов (формирования требований, анализа, синтеза и тестирования) промышленного конструирования.

Цель преподавания дисциплины: изучение современных инженерных принципов (методов) создания надежного, качественного программного обеспечения (ПО), удовлетворяющего предъявляемым к нему требованиям; формирование у студентов понимания необходимости применения данных принципов программной инженерии.

Задачи данного практикума дисциплины: освоить на практических примерах и научить студентов формулировать требования к создаваемым программным комплексам; формировать архитектуру программных комплексов различного назначения, разрабатывать программные приложения; владеть навыками разработки программных комплексов для решения прикладных задач, оценки сложности алгоритмов и программ, использования современных технологий программирования, тестирования и документирования программных комплексов; получить представление и уметь использовать основные международные и отечественные стандарты в области программной инженерии.

В лабораторном практикуме студенты получают практические навыки работы с современными технологиями проектирования и программных средств, используя для этого инструментальные средства UML (Rational Rose, Visual Studio), IDEF. В качестве инструментальной среды разработки ПС используется MS Visual Studio .Net, которая позволяет писать программы на разных языках программирования высокого уровня, эффективно выполнять отладку и тестирование, получать числовые метрики кода для оценки сложности программного продукта.

Выполнение лабораторного практикума содержит элементы курсовой работы, поэтому для лабораторных работ используется сквозной объект в индивидуальных заданиях.

Лабораторная работа №1

РАЗРАБОТКА СПЕЦИФИКАЦИЙ СИСТЕМНЫХ ТРЕБОВАНИЙ К ПРОГРАММНОМУ ПРОДУКТУ

Цель работы: изучение требований к создаваемому программному продукту, разработка технического задания

Программа выполнения работы

1. Изучить нормативные документы по разработке технического задания на разработку программного продукта.
2. Разработать техническое задание на программный продукта по заданному варианту.

Содержание отчета

Техническое задание на программный продукт

Методические указания

Техническое задание представляет собой документ, в котором сформулированы основные цели разработки, требования к программному продукту, определены сроки и этапы разработки и регламентирован процесс приемно-сдаточных испытаний. В разработке технического задания участвуют как представители заказчика, так и представители исполнителя. В основе этого документа лежат исходные требования заказчика, анализ передовых достижений техники, результаты выполнения научно-исследовательских работ, предпроектных исследований, научного прогнозирования и т. п.

Основные факторы, определяющие характеристики разрабатываемого программного обеспечения:

- исходные данные и требуемые результаты, которые определяют *функции* программы или системы;
- среда функционирования (программная и аппаратная); может быть задана, а может выбираться для обеспечения параметров, указанных в техническом задании;
- возможное взаимодействие с другим программным обеспечением или специальными техническими средствами - также может

быть определено, а может выбираться исходя из набора выполняемых функций.

Разработка технического задания выполняется в следующей последовательности. Прежде всего, устанавливают набор выполняемых функций, а также перечень и характеристики исходных данных. Затем определяют перечень результатов, их характеристики и способы представления.

Далее уточняют среду функционирования программного обеспечения: конкретную комплектацию и параметры технических средств, версию используемой операционной системы и, возможно, версии и параметры другого установленного программного обеспечения, с которым предстоит взаимодействовать будущему программному продукту.

В случаях, когда разрабатываемое программное обеспечение собирает и хранит некоторую информацию или включается в управление каким-либо техническим процессом, необходимо также четко регламентировать действия программы в случае сбоев оборудования и энергоснабжения.

На техническое задание существует стандарт ГОСТ 19.201-78 «Техническое задание. Требования к содержанию и оформлению». В соответствии с этим стандартом техническое задание должно содержать следующие разделы:

- введение;
- основания для разработки;
- назначение разработки;
- требования к программе или программному изделию;
- требования к программной документации;
- технико-экономические показатели;
- стадии и этапы разработки;
- порядок контроля и приемки.

При необходимости допускается в техническое задание включать приложения.

Рассмотрим более подробно содержание каждого раздела.

Введение должно включать наименование и краткую характеристику области применения программы или программного продукта, а также объекта (например, системы) в котором предполагается их использовать. Назначение введения - продемонстрировать актуальность данной разработки и показать, какое место эта разработка занимает в ряду подобных.

Раздел *Основания для разработки* должен содержать наименование документа, на основании которого ведется разработка, организации, утвердившей данный документ, и наименование или условное обозначение темы разработки. Таким документом может служить план, приказ, договор и т. п.

Раздел *Назначение разработки* должен содержать описание функционального и эксплуатационного назначения программного продукта с указанием категорий пользователей.

Раздел *Требования к программе или программному изделию* должен включать следующие подразделы:

- требования к функциональным характеристикам;
- требования к надежности;
- условия эксплуатации;
- требования к составу и параметрам технических средств;
- требования к информационной и программной совместимости;
- требования к маркировке и упаковке;
- требования к транспортированию и хранению;
- специальные требования.

Наиболее важным из перечисленных выше является подраздел *Требования к функциональным характеристикам.* В этом разделе должны быть перечислены выполняемые функции и описаны состав, характеристики и формы представления исходных данных и результатов. В этом же разделе при необходимости указывают критерии эффективности: максимально допустимое время ответа системы, максимальный объем используемой оперативной и/или внешней памяти и др.

Примечание. Если разработанное программное обеспечение не будет выполнять указанных в техническом задании функций, то оно считается не соответствующим техническому заданию, т. е. неправильным с точки зрения критериев качества. Универсальность будущего продукта также обычно специально не оговаривается, но подразумевается.

В подразделе *Требования к надежности* указывают уровень надежности, который должен быть обеспечен разрабатываемой системой и время восстановления системы после сбоя. Для систем с обычными требованиями к надежности в этом разделе иногда регламентируют действия разрабатываемого продукта по увеличению надежности результатов (контроль входной и выходной информации, создание резервных копий промежуточных результатов и т. п.).

В подразделе *Условия эксплуатации,* указывают особые требования к условиям эксплуатации: температуре окружающей среды, относительной влажности воздуха и т. п. Как правило, подобные требования формулируют, если разрабатываемая система будет эксплуатироваться в нестандартных условиях или использует специальные внешние устройства, например для хранения информации. Здесь же указывают вид обслуживания, необходимое количество и квалификация персонала. В противном случае допускается указывать, что требования не предъявляются.

В подразделе *Требования к составу и параметрам технических средств* указывают необходимый состав технических средств с указанием их основных технических характеристик: тип микропроцессора, объем памяти, наличие внешних устройств и т. п. При этом часто указывают два варианта конфигурации: минимальный и рекомендуемый.

В подразделе *Требования к информационной и программной совместимости* при необходимости можно задать методы решения, определить язык или среду программирования для разработки, а также используемую операционную систему и другие системные и пользовательские программные средства, с которым должно взаимодействовать разрабатываемое программное обеспечение. В этом же разделе при необходимости указывают, какую степень защиты информации необходимо предусмотреть.

В разделе *Требования к программной документации* указывают необходимость наличия руководства программиста, руководства пользователя, руководства системного программиста, пояснительной записки и т. п. На все эти типы документов также существуют ГОСТы.

В разделе *Технико-экономические показатели* рекомендуется указывать ориентировочную экономическую эффективность, предполагаемую годовую потребность и экономические преимущества по сравнению с существующими аналогами.

В разделе *Стадии и этапы разработки* указывают стадии разработки, этапы и содержание работ с указанием сроков разработки и исполнителей.

В разделе *Порядок контроля и приемки* указывают виды испытаний и общие требования к приемке работы.

В приложениях при необходимости приводят: перечень научно-исследовательских работ, обосновывающих разработку; схемы алго-

ритмов, таблицы, описания, обоснования, расчеты и другие документы, которые следует использовать при разработке.

В зависимости от особенностей разрабатываемого продукта разрешается уточнять содержание разделов, т. е. использовать подразделы, вводить новые разделы или объединять их.

В случаях, если какие-либо требования, предусмотренные техническим заданием, заказчик не предъявляет, следует в соответствующем месте указать «Требования не предъявляются».

Разработка технического задания - процесс трудоемкий, требующий определенных навыков. Наиболее сложным, как правило, является четкое формулирование основных разделов: введения, назначения и требований к программному продукту.

Контрольные вопросы

1. Что понимают под технологичностью программного обеспечения? Почему?
2. Какие типы программных продуктов можно выделить? Чем они различаются?
3. Назовите основные эксплуатационные требования к программным продуктам. Какими средствами и приемами обеспечивается каждый из них? Для каких типов программных систем целесообразно указывать каждый из них?
4. В каких ситуациях необходимы предпроектные исследования? Какие вопросы при этом решают? Что получают в результате таких исследований?
5. Назовите, какой раздел технического задания можно считать основным и почему? Какую информацию должны содержать остальные разделы? В чем основная сложность разработки технического задания?

Варианты заданий

1. Программа учета домашней медиатеки
2. Программа планирования дел «Ежедневник»
3. Информационная система учета услуг в автомастерской
4. Программа информационной поддержки спортивных соревнований
5. Информационно-справочная система для продажи билетов в кинотеатре

6. Программа учета и анализа продаж в продовольственном магазине

7. Информационная система факультета «Абитуриент»

8. Программа информационного обеспечения фестиваля художественной самодеятельности студентов

9. Программа информационной поддержки спартакиады университета

10. Программа учета и анализа доходов и расходов семьи

11. Программа формирования счетов-квитанций для жильцов ТСЖ

12. Система управления теплицей

13. Программа обработки данных аттестации студентов

14. Визуальный конструктор Е-сетей

15. Программа управления очередностью обслуживания клиентов в поликлинике

16. Программа терминала оплаты за услуги населению

17. Программа информационной поддержки спортивных соревнований

18. Программа учета контингента студентов на факультете

19. Программу «Маклер» для учета заявок на обмен квартир и поиска вариантов обмена

20. Компьютерная игра «Сражение роботов»

Лабораторная работа №2

ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ ПРОГРАММНОГО ПРОДУКТА

Цель работы: функциональное моделирование программного продукта с использованием диаграммы вариантов использования UML

Программа выполнения работы

1. Изучить технологию построения диаграммы вариантов использования в среде CASE системы Rational Rose

2. Произвести построение диаграммы вариантов использованиям среды Rational Rose на примере программной системы из лаб. Работы №1.

Методически указания

Работа над моделью в среде IBM Rational Rose начинается с общего анализа проблемы и построения диаграммы вариантов использования, которая отражает функциональное назначение проектируемой программной системы.

В качестве проекта далее будет рассматриваться модель системы управления банкоматом. Достоинством этого проекта является то, что он не требует специального описания предметной области, поскольку предполагает интуитивное знакомство читателей с особенностями функционирования банкомата. При этом разрабатываемая модель системы управления банкоматом используется в качестве сквозного примера, в рамках которого иллюстрируются особенности разработки различных диаграмм языка UML в среде IBM Rational Rose 2003.

Назначение отдельных кнопок данной панели можно узнать также из всплывающих подсказок, которые появляются, если подвести и задержать на некоторое время указатель мыши над той или иной кнопкой (табл. 2.1). На специальной панели инструментов по умолчанию присутствует только часть кнопок с пиктограммами элементов, которые могут быть использованы для построения диаграммы. Добавить кнопки с пиктограммами других графических элементов, например, таких как *бизнес-вариант использования* (business use case), *бизнес-актер* (business actor), сотрудник (business worker), или удалить ненужные кнопки можно с помощью настройки специальной панели инструментов.

Открыть диалоговое окно настройки специальных панелей инструментов для диаграмм в среде IBM Rational Rose 2003 можно с помощью операции главного меню: **Tools→Options** (Инструменты→ Параметры), раскрыв вкладку **Toolbars** (Панели инструментов) и нажав соответствующую кнопку (например, **Use Case diagram**) в группе опций **Customize Toolbars** (Настройка панелей инструментов). Это окно настройки также можно открыть с помощью операции контекстного меню **Customize** (Настройка) при позиционировании курсора на специальной панели инструментов (рис. 2.1).

Таблица 2.1.

Назначение кнопок специальной панели инструментов для диаграммы вариантов использования

Графическое изображение	Всплывающая подсказка	Назначение кнопки
	**Selection Tool**	Превращает изображение курсора в форму стрелки для последующего выделения элементов на диаграмме
	**Text Box**	Добавляет на диаграмму текстовую область
	**Note**	Добавляет на диаграмму примечание
	**Anchor Note to Item**	Добавляет на диаграмму связь примечания с соответствующим графическим элементом диаграммы
	**Package**	Добавляет на диаграмму пакет
	**Use Case**	Добавляет на диаграмму вариант использования
	**Actor**	Добавляет на диаграмму актера
	**Unidirectional Association**	Добавляет на диаграмму направленную *ассоциацию*
	**Dependency or Instantiates**	Добавляет на диаграмму отношение зависимости
	**Generalization**	Добавляет на диаграмму отношение обобщения

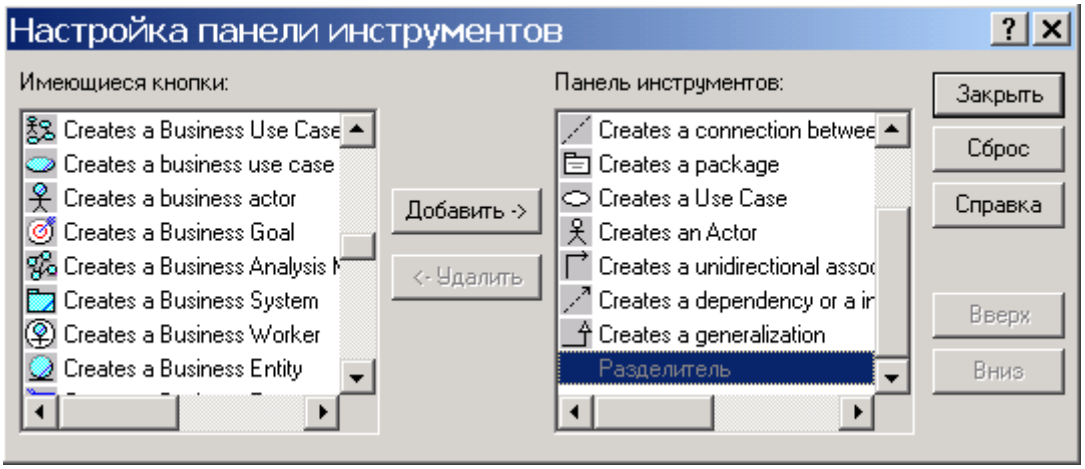


Рис. 2.1. Диалоговое окно настройки панели инструментов

Для построения диаграммы для рассматриваемой модели банкомата следует выполнить следующие действия:

1. Добавить актера с именем Банк, для которого выбрать стереотип Service (*Сервис*), означающий, что банкомат использует некоторые услуги Банка в качестве *сервиса*.

2. Добавить *вариант использования* Получение справки о состоянии счета, для которого выбрать *стереотип* Business Use Case (*Бизнес-вариант использования*).

3. Добавить *вариант использования* Блокирование кредитной карточки.

4. Добавить направленную *ассоциацию* от *бизнес-актера* Клиент Банкомата к *варианту использования* Получение справки о состоянии счета.

5. Добавить направленную *ассоциацию* от *варианта использования* Снятие наличных по кредитной карточке к *сервису* Банк.

6. Добавить направленную *ассоциацию* от *варианта использования* Получение справки о состоянии счета к *сервису* Банк.

7. Добавить отношение зависимости со стереотипом <<include>>, направленное от *варианта использования* Получение справки о состоянии счета к *варианту использования* Проверка ПИН-кода.

8. Добавить отношение зависимости со стереотипом <<extend>>, направленное от *варианта использования* Блокирование кредитной карточки к *варианту использования* Проверка ПИН-кода.

При этом отношение зависимости со стереотипом <<extend>> на данной диаграмме означает следующее. Вариант использования Блокирование кредитной карточки будет выполняться только в том случае,

если в результате проверки ПИН-кода будет установлено, что соответствующая кредитная карточка утрачена ее владельцем или признана недействительной. Построенная таким образом диаграмма вариантов использования будет иметь следующий вид (рис. 2.2).

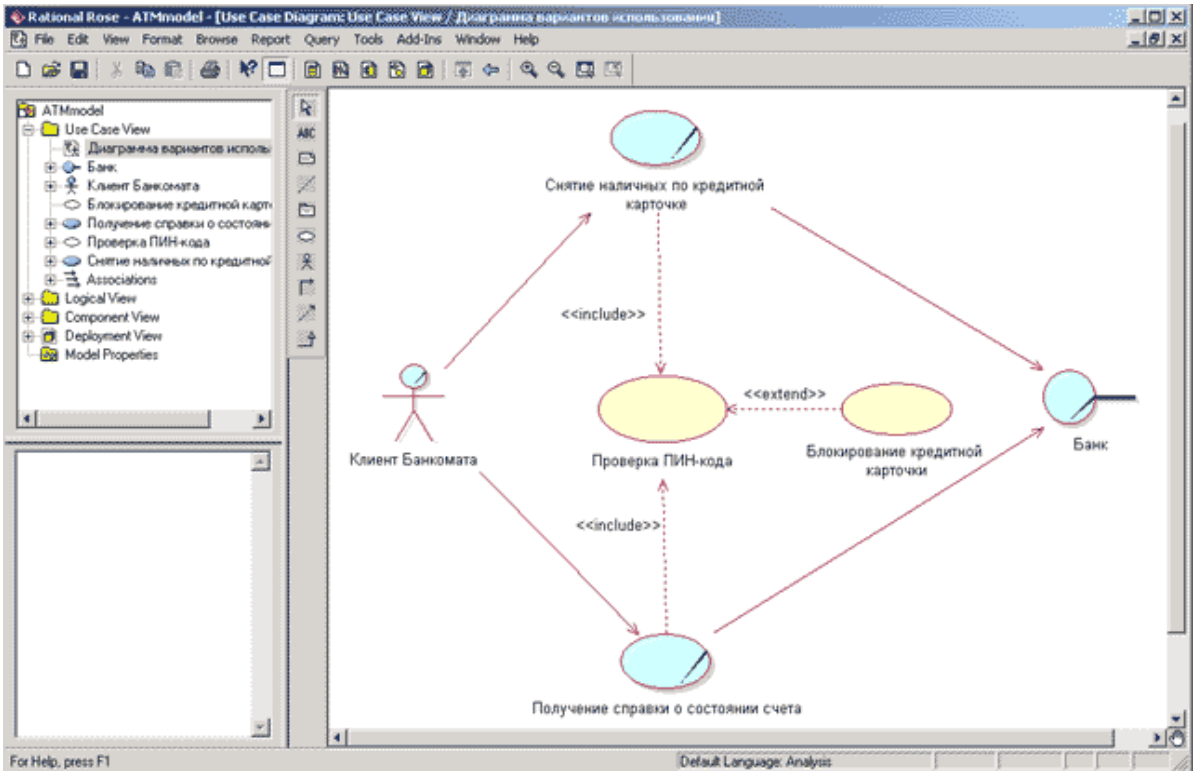


Рис. 2.2. Диаграмма вариантов использования для модели банкоматов

Контрольные вопросы

1. Из каких элементов состоит диаграмма Use Case?
2. Какие отношения разрешены между элементами диаграммы Use Case?
3. Для чего применяют диаграммы Use Case?
4. Чем отличаются друг от друга отношения включения и расширения с точки зрения управления?
5. Каково назначение спецификации элемента Use Case и как она оформляется?
6. Что такое сценарий элемента Use Case?

Лабораторная работа №3

РАСЧЕТ ХАРАКТЕРИСТИК МОДУЛЬНОЙ ПРОГРАММНОЙ СИСТЕМЫ

Цель работы: оценка показателей связности и сцепления модульной программной системы

Программа выполнения работы

1. Изучить методические указания по расчету характеристик модульной программной системы и показателей ее сложности.
2. Вычислить характеристики модульности и сложности программы для заданного варианта.

Методически указания

Модуль - фрагмент программного текста, являющийся строительным блоком для физической структуры системы. Как правило, модуль состоит из интерфейсной части и части-реализации.

Модульность — свойство системы, которая может подвергаться декомпозиции на ряд внутренне связанных и слабо зависящих друг от друга модулей. По определению Г. Майерса, модульность — свойство ПО, обеспечивающее интеллектуальную возможность создания сколь угодно сложной программы.

Связность модуля (Cohesion) — это мера зависимости его частей. Связность — внутренняя характеристика модуля. Чем выше связность модуля, тем лучше результат проектирования, то есть тем «черней» его ящик (капсула, защитная оболочка модуля), тем меньше «ручек управления» на нем находится и тем проще эти «ручки».

Для измерения связности используют понятие силы связности (СС). Существует 7 типов связности:

1. Связность по совпадению (СС=0). В модуле отсутствуют явно выраженные внутренние связи.

2. Логическая связность (СС=1). Части модуля объединены по принципу функционального подобия. Например, модуль состоит из разных подпрограмм обработки ошибок. При использовании такого модуля клиент выбирает только одну из подпрограмм.

Недостатки:

- сложное сопряжение;
- большая вероятность внесения ошибок при изменении сопряжения ради одной из функций.

3. Временная связность (СС=3). Части модуля не связаны, но необходимы в один и тот же период работы системы.

Недостаток: сильная взаимная связь с другими модулями, отсюда — сильная чувствительность внесению изменений.

4. Процедурная связность (СС=5). Части модуля связаны порядком выполняемых ими действий, реализующих некоторый сценарий поведения.

5. Коммуникативная связность (СС=7). Части модуля связаны по данным (работают с одной и той же структурой данных).

6. Информационная (последовательная) связность (СС=9). Выходные данные одной части используются как входные данные в другой части модуля.

7. Функциональная связность (СС=10). Части модуля вместе реализуют одну функцию.

Отметим, что типы связности 1,2,3 — результат неправильного планирования архитектуры, а тип связности 4 — результат небрежного планирования архитектуры приложения.

Алгоритм определения уровня связности модуля.

1. Если модуль — единичная проблемно-ориентированная функция, то уровень связности — функциональный; конец алгоритма. В противном случае перейти к пункту 2.
2. Если действия внутри модуля связаны, то перейти к пункту 3. Если действия внутри модуля никак не связаны, то перейти к пункту 6.
3. Если действия внутри модуля связаны данными, то перейти к пункту 4. Если действия внутри модуля связаны потоком управления, перейти к пункту 5.
4. Если порядок действий внутри модуля важен, то уровень связности — информационный. В противном случае уровень связности — коммуникативный. Конец алгоритма.
5. Если порядок действий внутри модуля важен, то уровень связности — процедурный. В противном случае уровень связности — временной. Конец алгоритма.
6. Если действия внутри модуля принадлежат к одной категории, то уровень связности — логический. Если действия внутри модуля не

принадлежат к одной категории, то уровень связности — по совпадению. Конец алгоритма.

Возможны более сложные случаи, когда с модулем ассоциируются несколько уровней связности. В этих случаях следует применять одно из двух правил:

- правило параллельной цепи. Если все действия модуля имеют несколько уровней связности, то модулю присваивают самый сильный уровень связности;
- правило последовательной цепи. Если действия в модуле имеют разные уровни связности, то модулю присваивают самый слабый уровень связности.

Например, модуль может содержать некоторые действия, которые связаны процедурно, а также другие действия, связные по совпадению. В этом случае применяют правило последовательной цепи и в целом модуль считают связным по совпадению.

Сцепление модулей. Сцепление (Coupling) — мера взаимозависимости модулей по данным. Сцепление — внешняя характеристика модуля, которую желательно уменьшать.

Количественно сцепление измеряется степенью сцепления (СЦ). Выделяют 6 типов сцепления.

1. Сцепление по данным (СЦ=1). Модуль А вызывает модуль В.

Все входные и выходные параметры вызываемого модуля — простые элементы данных.

2. Сцепление по образцу (СЦ=3). В качестве параметров используются структуры данных.
3. Сцепление по управлению (СЦ=4). Модуль А явно управляет функционированием модуля В (с помощью флагов или переключателей), посылая ему управляющие данные.
4. Сцепление по внешним ссылкам (СЦ=5). Модули А и В ссылаются на один и тот же глобальный элемент данных.
5. Сцепление по общей области (СЦ=7). Модули разделяют одну и ту же глобальную структуру данных.
6. Сцепление по содержанию (СЦ=9). Один модуль прямо ссылается на содержание другого модуля (не через его точку входа). Например, коды их команд перемежаются друг с другом.

Сложность программной системы. В простейшем случае сложность системы определяется как сумма мер сложности ее модулей. Сложность модуля может вычисляться различными способами.

М. Холстед предложил меру длины N модуля:

$$N \approx n_1 log_2 (n_1) + n_2 log_2 (n_2),$$

где n_1 — число различных операторов, n_2 — число различных операндов.

В качестве второй метрики М. Холстед рассматривал объем V модуля (количество символов для записи всех операторов и операндов текста программы):

$$V = N \text{ x } log_2 (n_1 + n_2).$$

Вместе с тем известно, что любая сложная система состоит из элементов и системы связей между элементами и что игнорировать внутрисистемные связи неразумно.

Том МакКейб при оценке сложности ПС предложил исходить из топологии внутренних связей. Для этой цели он разработал метрику цикломатической сложности:

$$V(G) = E\text{-}N + 2,$$

где E — количество дуг, а.N — количество вершин в управляющем графе ПС. Таким образом, при комплексной оценке сложности ПС необходимо рассматривать меру сложности модулей, меру сложности внешних связей (между модулями) и меру сложности внутренних связей (внутри модулей).

Контрольные вопросы

1. Поясните понятия модуля и модульности. Зачем используют модули?
2. В чем состоит принцип информационной закрытости? Какие достоинства он имеет?
3. Что такое связность модуля?
4. Какие существуют типы связности?
5. Дайте характеристику функциональной связности.
6. Дайте характеристику информационной связности.
7. Охарактеризуйте коммуникативную связность.
8. Охарактеризуйте процедурную связность.
9. Дайте характеристику временной связности.
10. Дайте характеристику логической связности.
11. Охарактеризуйте связность по совпадению.
12. Что такое сцепление модуля?
13. Какие существуют типы сцепления?
14. Дайте характеристику сцепления по данным.
15. Дайте характеристику сцепления по образцу.

16. Охарактеризуйте сцепление по управлению.
17. Охарактеризуйте сцепление по внешним ссылкам.
18. Дайте характеристику сцепления по общей области.
19. Дайте характеристику сцепления по содержанию.
20. Что значит «улучшать сцепление»?
21. Какие подходы к оценке сложности системы вы знаете?

Лабораторная работа №4

РАЗРАБОТКА ДИАГРАММ КЛАССОВ НА ЯЗЫКЕ UML

Цель работы: построение диаграммы классов UML

Программа выполнения работы

1. Изучить назначение и методику разработки диаграммы классов.
2. Изучить технологию создания диаграмм классов UML в Rational Rose.
3. Разработать UML диаграмму классов для заданного варианта.

Методически указания

- Диаграмма *классов* является основным логическим представлением модели и содержит детальную информацию о внутреннем устройстве объектно-ориентированной программной системы или, используя современную терминологию, об архитектуре программной системы. Активизировать рабочее окно диаграммы *классов* можно несколькими способами:

- окно диаграммы *классов* появляется по умолчанию в рабочем окне диаграммы после создания нового проекта;

- щелкнуть на кнопке с изображением диаграммы *классов* на стандартной панели инструментов;

- раскрыть логическое представление (Logical View) в браузере проекта и дважды щелкнуть на пиктограмме Main (Главная);

- выполнить операцию главного меню: Browse→Class Diagram (Обзор→Диаграмма *классов*).

- При этом появляется новое окно с чистым рабочим листом диаграммы *классов* и специальная панель инструментов, содержащая кнопки с изображением графических примитивов, необходимых для разработки диаграммы *классов* (табл. 4.1). Назначение отдельных кнопок панели можно узнать также из всплывающих подсказок. На

специальной панели инструментов по умолчанию присутствует только часть пиктограмм элементов, которые могут быть использованы для построения диаграммы *классов*. Добавить кнопки с пиктограммами других графических элементов таких как, например, отношения агрегации и композиции, шаблон, *класс* бизнес-сущность, *управляющий класс*, или удалить ненужные кнопки можно с помощью настройки специальной панели инструментов. Соответствующее диалоговое окно настройки специальной панели инструментов для диаграммы *классов* можно вызвать аналогично другим панелям с помощью операции контекстного меню **Customize** (Настройка) при позиционировании курсора на специальной панели инструментов.

Таблица 4.1.

Назначение кнопок специальной панели инструментов для диаграммы классов

Графическое изображение	Всплывающая подсказка	Назначение кнопки
	Selection Tool	Превращает изображение курсора в форму стрелки для последующего выделения элементов на диаграмме
	Text Box	Добавляет на диаграмму текстовую область
	Note	Добавляет на диаграмму примечание
	Anchor Note to Item	Добавляет на диаграмму связь примечания с соответствующим графическим элементом диаграммы
	Class	Добавляет на диаграмму *класс*
	Interface	Добавляет на диаграмму *интерфейс*
	Unidirectional Association	Добавляет на диаграмму направленную *ассоциацию*
	Association Class	Добавляет на диаграмму *ассоциацию класс*
	Package	Добавляет на диаграмму пакет
	Dependency or Instantiates	Добавляет на диаграмму отношение зависимости
	Generalization	Добавляет на диаграмму отношение обобщения
	Realize	Добавляет на диаграмму отношение реализации

Для построения диаграммы классов рассматриваемой модели банкомата следует описанным выше способом добавить оставшиеся классы и *ассоциации*, а также специфицировать стереотипы, атрибуты и операции этих классов. С этой целью следует выполнить следующие действия:

1. Для класса IИнтерфейс Банка добавить операцию: проверить идентификатор карточки (идентификатор карточки: Integer) с квантором видимости public. В качестве типа возвращаемого результата для этой операции следует выбрать тип Boolean (логический), а в качестве целочисленного аргумента задать идентификатор карточки. Для задания аргумента необходимо перейти на вкладку Detail (Подробно) окна спецификации свойств даной операции и после добавления аргумента с помощью операции контекстного меню Insert ввести имя аргумента и его тип Integer в соответствующие поля ввода.
2. Для класса IИнтерфейс Банка добавить операцию: открыть счет клиента (идентификатор карточки: Integer) с квантором видимости public. В качестве целочисленного аргумента этой операции следует задать идентификатор карточки.
3. Для класса IИнтерфейс Банка добавить операцию: проверить баланс клиента (идентификатор карточки: Integer, введенная сумма наличных: Currency) с квантором видимости public. В качестве типа возвращаемого результата для этой операции следует выбрать тип Boolean (логический). В качестве первого целочисленного аргумента этой операции следует задать идентификатор карточки, а в качестве второго аргумента - введенная сумма наличных с типом Currency (Денежный).
4. Для класса IИнтерфейс Банка добавить операцию: уменьшить счет клиента (идентификатор карточки: Integer, введенная сумма наличных: Currency) с квантором видимости public. В качестве типа возвращаемого результата для этой операции следует выбрать тип Boolean (логический). В качестве первого целочисленного аргумента этой операции следует задать идентификатор карточки, а в качестве второго аргумента - введенная сумма наличных с типом Currency (Денежный).
5. Для класса Устройство чтения карточки добавить операцию: прочитать идентификатор карточки() с квантором видимости public.

В качестве типа возвращаемого результата для этой операции следует выбрать тип Integer (целочисленный), а в секцию документации данной операции следует ввести поясняющий текст: «Вызывается после того, как кредитная карточка вставлена в Устройство чтения карточки».

6. Для класса Устройство чтения карточки добавить операцию: прочитать ПИН-код() с квантором видимости public. В качестве типа возвращаемого результата для этой операции следует выбрать тип Integer (целочисленный), а в секцию документации данной операции следует ввести поясняющий текст: «Вызывается после того, как кредитная карточка вставлена в Устройство чтения карточки».

7. Для класса Устройство чтения карточки добавить операцию: вернуть кредитную карточку() с квантором видимости public. В секцию документации данной операции следует ввести поясняющий текст: «Вызывается после завершения транзакции».

8. Для класса Устройство чтения карточки добавить операцию: блокировать кредитную карточку() с квантором видимости public. В секцию документации данной операции следует ввести поясняющий текст: «Вызывается после того, как установлен факт утраты кредитной карточки владельцем».

9. Добавить класс с именем Экран Банкомата, для которого выбрать стереотип boundary. Данный класс также находится на границе моделируемой системы, на что и указывает этот стереотип. В секцию документации данного класса следует ввести поясняющий текст: «Устанавливается на банкомате».

10. Для класса Экран Банкомата добавить операцию: показать меню опций() с квантором видимости public.

11. Для класса Экран Банкомата добавить операцию: показать меню снятия суммы() с квантором видимости public.

12. Добавить класс с именем Клавиатура Банкомата, для которого выбрать стереотип boundary. В секцию документации данного класса следует ввести поясняющий текст: «Устанавливается на банкомате».

13. Для класса Клавиатура Банкомата добавить операцию: ввести ПИН-код() с квантором видимости public. В качестве типа возвращаемого результата для этой операции следует выбрать тип Integer, а в секцию документации данной операции следует ввести поясняющий текст: «Вызывается после того, как клиент ввел значение ПИН-кода с клавиатуры».

14. Для класса Клавиатура Банкомата добавить операцию: ввести тип транзакции() с квантором видимости public. В качестве типа возвращаемого результата для этой операции следует выбрать тип Boolean (логический), а в секцию документации данной операции следует ввести поясняющий текст: «Возвращает значение Истина, если клиент выбирает снятие наличных, и значение Ложь, если клиент выбирает получение справки о состоянии счета».

15. Для класса Клавиатура Банкомата добавить операцию: ввести сумму снятия наличных() с квантором видимости public. В качестве типа возвращаемого результата для этой операции следует выбрать тип Currency (Денежный), а в секцию документации данной операции следует ввести поясняющий текст: «Вызывается после того, как клиент ввел значение снимаемой суммы с клавиатуры».

16. Добавить класс с именем Устройство выдачи наличных, для которого выбрать стереотип boundary. В секцию документации данного класса следует ввести поясняющий текст: «Устанавливается на банкомате».

17. Для класса Устройство выдачи наличных добавить операцию: выдать наличные() с квантором видимости public. В секцию документации данной операции следует ввести поясняющий текст: «Вызывается после того, как выполнено снятие запрошенной клиентом суммы со счета».

18. Добавить класс с именем Принтер Банкомата, для которого выбрать стереотип boundary. В секцию документации данного класса следует ввести поясняющий текст: «Устанавливается на банкомате».

19. Для класса Принтер Банкомата добавить операцию: распечатать чек() с квантором видимости public. В секцию документации данной операции следует ввести поясняющий текст: «Вызывается по дополнительному запросу клиента».

20. Добавить направленную *ассоциацию* от класса Контроллер Банкомата к классу Устройство чтения карточки. В качестве *кратности* концов этой *ассоциации* установить значение 1.

21. Добавить направленную *ассоциацию* от класса Контроллер Банкомата к классу Принтер Банкомата. В качестве *кратности* концов этой *ассоциации* установить значение 1.

22. Добавить направленную *ассоциацию* от класса Контроллер Банкомата к классу Клавиатура Банкомата. В качестве *кратности* концов этой *ассоциации* установить значение 1.

23. Добавить направленную *ассоциацию* от класса Контроллер Банкомата к классу Устройство выдачи наличных. В качестве *кратности* концов этой *ассоциации* установить значение 1.

24. Добавить направленную *ассоциацию* от класса Контроллер Банкомата к классу Экран Банкомата. В качестве *кратности* концов этой *ассоциации* установить значение 1.

25. Добавить направленную *ассоциацию* от класса Контроллер Банкомата к классу IКонтроллер Банка. В качестве *кратности* конца этой *ассоциации* для первого класса установить значение 0..n, а *кратности* конца *ассоциации* для второго класса установить значение 1. В качестве стереотипа данной *ассоциации* выбрать из вложенного списка значение <<communicate>>. Применение данного стереотипа означает, что между этими классами должна существовать физическая взаимосвязь.

Построенная в результате указанных действий диаграмма классов будет иметь следующий вид (рис. 4.1).

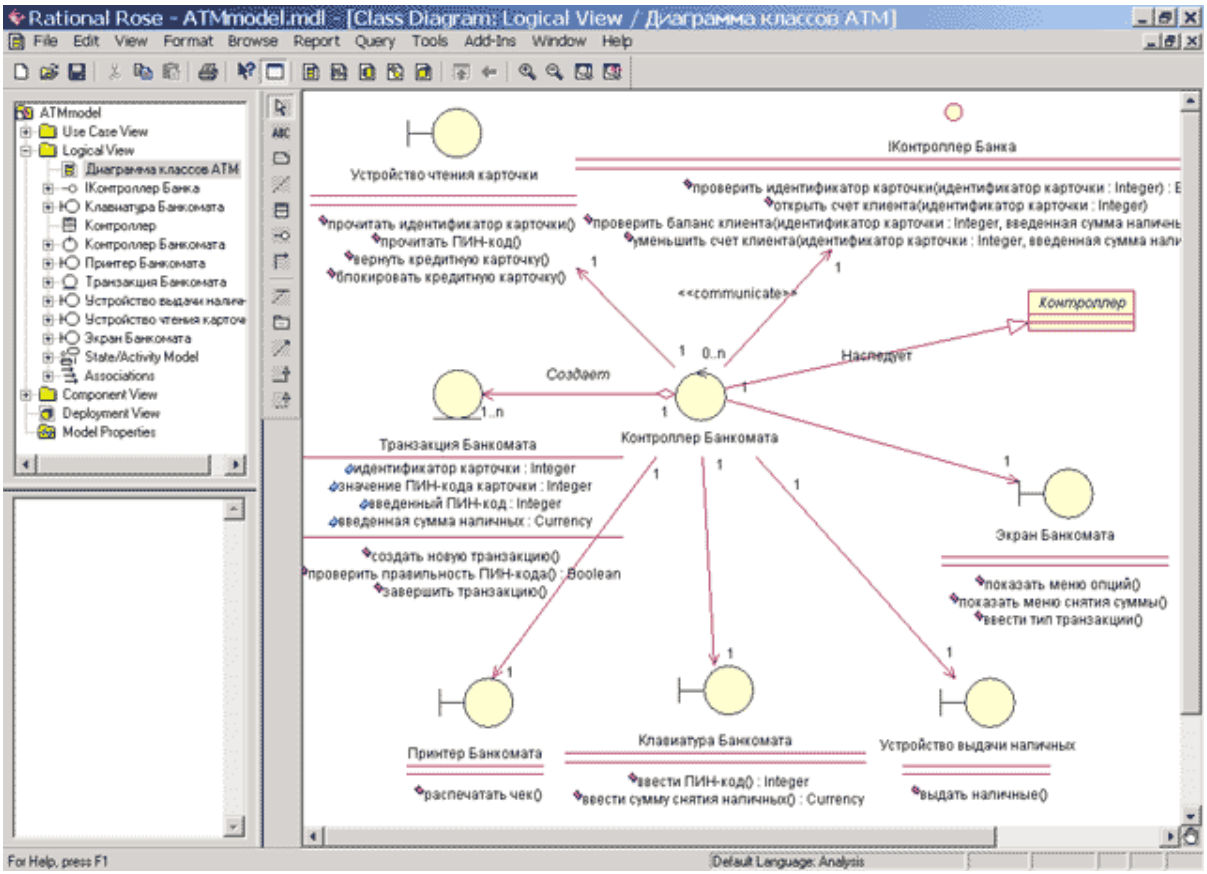


Рис. 4.1. Диаграмма классов для разрабатываемой модели банкомата

Контрольные вопросы

1. Поясните назначение статических моделей объектно-ориентированных программных систем.
2. Что является основным средством для представления статических моделей?
3. Как используются статические модели?
4. Какие секции входят в графическое обозначение класса?
5. Какие секции класса можно не показывать?
6. Поясните общий синтаксис представления свойства.
7. Какие уровни видимости вы знаете? Их смысл?
8. Какие характеристики свойств вам известны?
9. Какой смысл имеет класс-ассоциация?
10. Чем отличается агрегация от композиции? Разновидностями какого отношения (в UML) они являются?
11. Что обозначает в UML простая зависимость?
12. Какой смысл имеет отношение обобщения?
13. Какие недостатки у множественного наследования?
14. Что такое абстрактный класс (операция) и как он (она) отображается?
15. Как обозначить корневой класс?

Лабораторная работа №5

РАЗРАБОТКА ДИАГРАММ ВЗАИМОДЕЙСТВИЯ ОБЪЕКТОВ НА ЯЗЫКЕ UML

Цель работы: построение диаграмм взаимодействия UML

Программа выполнения работы

1.Изучить методику представления взаимодействия программных объектов с помощью диаграмм кооперации UML.

2.Изучить технологию построения диаграмм UML в среде Rational Rose.

Методически указания

Диаграмма кооперации является разновидностью диаграммы взаимодействия, и в контексте языка UML описывает динамический аспект взаимодействия объектов при реализации отдельных вариан-

тов использования. Активизировать рабочее окно диаграммы кооперации в программе IBM Rational Rose 2003 можно несколькими способами:

- Щелкнуть на кнопке с изображением диаграммы взаимодействия на стандартной панели инструментов и выбрать для построения новую диаграмму кооперации.

- Выполнить операцию главного меню: Browse→Interaction Diagram (Браузер→Диаграмма взаимодействия) и выбрать для построения новую диаграмму кооперации.

- Выполнить операцию контекстного меню: New→Collaboration Diagram (Новая→Диаграмма кооперации) для логического представления или представления вариантов использования в браузере проекта.

- При этом появляется новое окно с чистым рабочим листом диаграммы кооперации и специальная панель инструментов, содержащая кнопки с изображением графических примитивов, необходимых для разработки диаграммы кооперации (табл. 5.1). Назначение отдельных кнопок панели можно узнать из всплывающих подсказок.

Таблица 5.1.

Назначение кнопок специальной панели инструментов диаграммы кооперации

Графическое изображение	Всплывающая подсказка	Назначение кнопки
	**Selection Tool**	Превращает изображение курсора в форму стрелки для последующего выделения элементов на диаграмме
	**Text Box**	Добавляет на диаграмму текстовую область
	**Note**	Добавляет на диаграмму примечание
	**Anchor Note to Item**	Добавляет на диаграмму *связь* примечания с соответствующим графическим элементом диаграммы
	**Object**	Добавляет на диаграмму *объект*
	**Class Instance**	Добавляет на диаграмму экземпляр класса
	**Object Link**	Добавляет на диаграмму *связь*
	**Link To Self**	Добавляет на диаграмму рефлексивную *связь*

	**Link Message**	Добавляет на *связь* диаграммы прямое *сообщение*
	**Reverse Link Message**	Добавляет на *связь* диаграммы обратное *сообщение*
	**Data Token**	Добавляет на *связь* диаграммы элемент прямого потока данных
	**Reverse Data Token**	Добавляет на *связь* диаграммы элемент обратного потока данных

На специальной панели инструментов по умолчанию присутствуют практически все кнопки с пиктограммами элементов, которые могут быть использованы для построения диаграммы. В данной лекции в качестве примера рассматривается процесс построения диаграммы кооперации, которая представляет собой реализацию варианта использования Снятие наличных по кредитной карточке применительно к разрабатываемому проекту системы управления банкоматом. В модели данная диаграмма кооперации соответствует этому варианту использования и может быть размещена в представлении вариантов использования (**Use Case View**). После активизации новой диаграммы кооперации одним из описанных выше способов следует в качестве имени данной диаграммы задать: Снятие наличных по кредитной карточке.

В общем случае работа с диаграммой кооперации состоит в добавлении объектов, *связей* и *сообщений*, а также редактировании их свойств. При этом изменения, вносимые в диаграмму кооперации, автоматически вносятся в диаграмму последовательности, что можно увидеть в любой момент, активизировав последнюю нажатием клавиши <F5>.

Для построения диаграммы кооперации рассматриваемого примера следует выполнить следующие действия:

1. Добавить *объекты* классов с именами: Контроллер Банкомата, Транзакция Банкомата, Клавиатура Банкомата, Экран Банкомата, Принтер Банкомата, Устройство выдачи наличных и IИнтерфейс Банка.

2. Добавить *связи*, соединяющие *объекты* классов с именами: Контроллер Банкомата с Устройством чтения карточки, Контроллер Банкомата с Транзакцией Банкомата, Контроллер Банкомата с Клавиатурой Банкомата, Контроллер Банкомата с Экраном Банкомата, Контроллер Банкомата с Принтером Банкомата, Контроллер Банко-

мата с Устройством выдачи наличных и Контроллер Банкомата с IИнтерфейсом Банка.

3. Добавить *сообщение*: проверить идентификатор карточки (Integer) , направленное от *объекта* класса Контроллер Банкомата к *объекту* класса IИнтерфейс Банка.

4. Добавить *сообщение*: ввести ПИН-код(), направленное от *объекта* класса-актера Клиент Банкомата к *объекту* класса Клавиатура Банкомата.

5. Добавить *сообщение*: прочитать ПИН-код(), направленное от *объекта* класса Контроллер Банкомата к *объекту* класса Устройство чтения карточки.

6. Добавить *сообщение*: создать новую транзакцию(), направленное от *объекта* класса Контроллер Банкомата к *объекту* класса Транзакция Банкомата.

7. Добавить *сообщение*: проверить правильность ПИН-кода(), направленное от *объекта* класса Контроллер Банкомата к *объекту* класса Транзакция Банкомата.

8. Добавить *сообщение*: показать меню опций(), направленное от *объекта* класса Контроллер Банкомата к *объекту* класса Экран Банкомата.

9. Добавить *сообщение*: ввести тип транзакции(), направленное от *объекта* класса-актера Клиент Банкомата к *объекту* класса Клавиатура Банкомата.

10. Добавить *сообщение*: показать меню снятия суммы(), направленное от *объекта* класса Контроллер Банкомата к *объекту* класса Экран Банкомата.

11. Добавить *сообщение*: ввести сумму снятия наличных(), направленное от *объекта* класса-актера Клиент Банкомата к *объекту* класса Клавиатура Банкомата.

12. Последовательно добавить 3 *сообщения*: открыть счет клиента (Integer) , проверить баланс клиента (Integer, Currency) и уменьшить счет клиента(Integer, Currency), направленные от *объекта* класса Контроллер Банкомата к *объекту* класса IИнтерфейс Банка.

13. Добавить *сообщение*: распечатать чек(), направленное от *объекта* класса Контроллер Банкомата к *объекту* класса Принтер Банкомата.

14. Добавить *сообщение*: вернуть кредитную карточку(), направленное от *объекта* класса Контроллер Банкомата к *объекту* класса Устройство чтения карточки.

15. Добавить *сообщение*: выдать наличные(), направленное от *объекта* класса Контроллер Банкомата к *объекту* класса Устройство выдачи наличных.

16. Добавить *сообщение*: завершить транзакцию(), направленное от *объекта* класса Контроллер Банкомата к *объекту* класса Транзакция Банкомата.

Диаграмма кооперации, описывающая реализацию типичного хода событий варианта использования Снятие наличных по кредитной карточке для проекта системы управления банкоматом, показана на рис. 5.1.

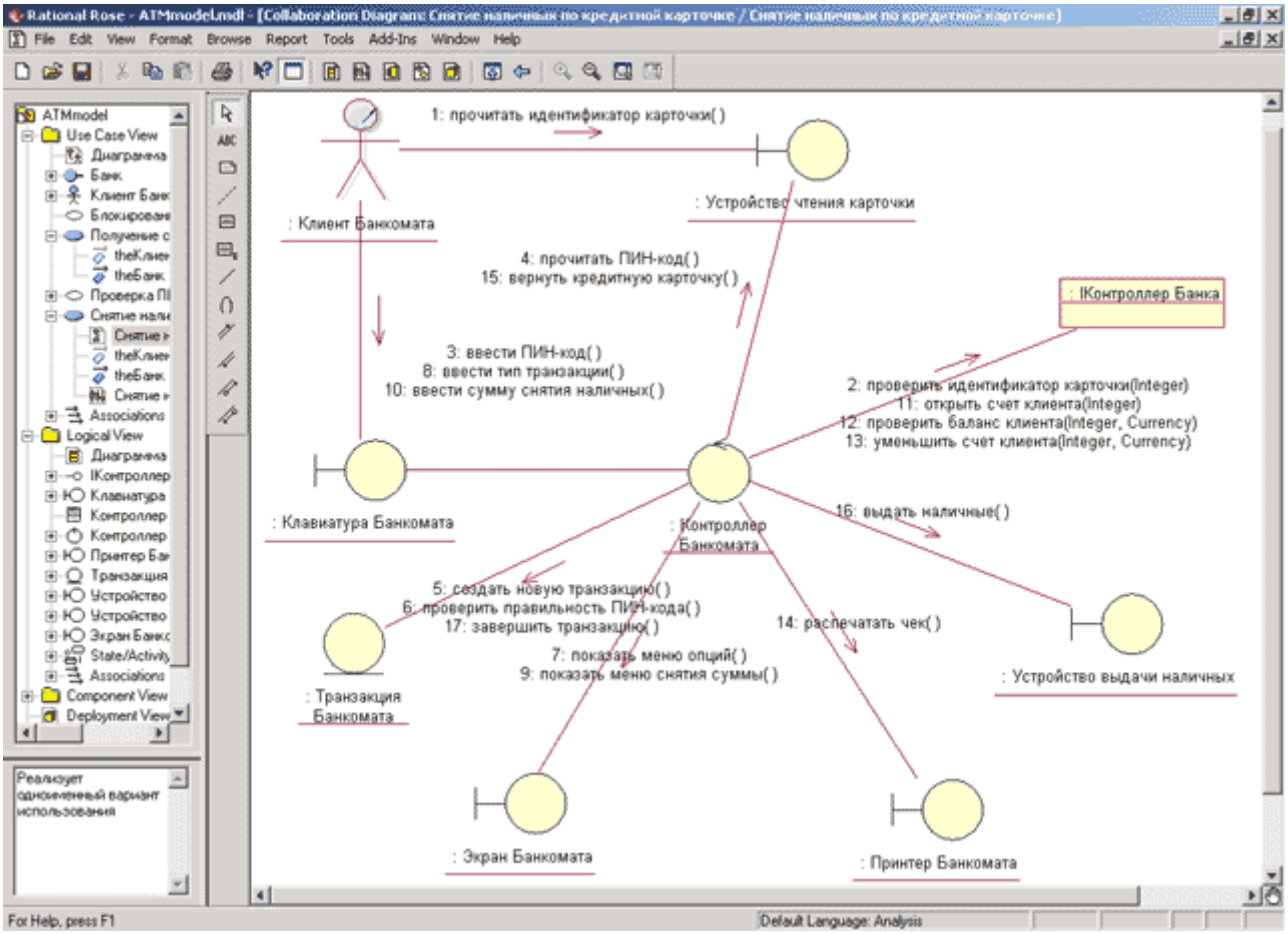


Рис. 5.1. Диаграмма кооперации, описывающая типичный ход событий варианта использования «Снятие наличных по кредитной карточке»

Контрольные вопросы

1. Как представляется имя объекта в диаграмме сотрудничества?

2. Поясните синтаксис представления свойства в диаграмме сотрудничества.

3. Какие стереотипы видимости используются в диаграмме сотрудничества? Поясните их смысл.

4. В какой форме записываются сообщения в языке UML? Поясните смысл сообщения.

5. В каком отношении находятся сообщения и действия? Перечислите разновидности действий.

6. Чем отличается процедурный поток от асинхронного потока сообщений?

7. Как указывается повторение сообщений?

8. Как показать ветвление сообщений?

9. Что общего в диаграмме последовательности и диаграмме сотрудничества? Чем они отличаются друг от друга?

Лабораторная работа №6

РАЗРАБОТКА ДИАГРАММ ПОВЕДЕНИЯ НА ЯЗЫКЕ UML

Цель работы: построение диаграмм поведения UML

Программа выполнения работы

1.Изучить методику представления поведения программных объектов с помощью диаграмм активности и состояний UML.

2.Изучить технологию построения диаграмм поведения UML в среде Rational Rose.

Методически указания

Начать построение диаграммы *состояний* для выбранного элемента модели или моделируемой системы в целом можно одним из следующих способов:

- Щелкнуть на кнопке с изображением диаграммы *состояний* на стандартной панели инструментов, после чего следует выбрать представление и тип разрабатываемой диаграммы - новая диаграмма *состояний*.

- Выделить логическое представление (Logical View) или представление вариантов использования (Use Case View) в браузере проекта и выполнить операцию контекстного меню: New→Statechart Diagram (Новая→Диаграмма *состояний*).

- Раскрыть логическое представление (Logical View) в браузере проекта и выделить рассматриваемый класс, операцию класса, пакет, или раскрыть представление вариантов использования (Use Case View) и выбрать вариант использования, после чего выполнить операцию контекстного меню: New→Statechart Diagram (Новая→Диаграмма *состояний*).

- Выполнить операцию главного меню: Browse→State Machine Diagram (Обзор→Диаграмма *состояний*), после чего следует выбрать представление и тип разрабатываемой диаграммы.

В результате выполнения этих действий появляется новое окно с чистым рабочим листом диаграммы *состояний* и специальная панель инструментов, содержащая кнопки с изображением графических элементов модели, необходимых для разработки диаграммы *состояний* (табл. 6.1). Назначение отдельных кнопок панели можно узнать из всплывающих подсказок.

Таблица 6.1.

Назначение кнопок специальной панели инструментов диаграммы состояний

Графическое изображение	Всплывающая подсказка	Назначение кнопки
	**Selection Tool**	Превращает изображение курсора в форму стрелки для последующего выделения элементов на диаграмме
	**Text Box**	Добавляет на диаграмму текстовую область
	**Note**	Добавляет на диаграмму примечание
	**Anchor Note to Item**	Добавляет на диаграмму связь примечания с соответствующим графическим элементом диаграммы
	**State**	Добавляет на диаграмму *состояние*
	**Start State**	Добавляет на диаграмму начальное *состояние*
	**End State**	Добавляет на диаграмму конечное *состояние*
	**State Transition**	Добавляет на диаграмму *переход*
	**Transition to Self**	Добавляет на диаграмму рефлексивный *переход*
	**Horizontal Synchronization**	Добавляет на диаграмму горизонтально расположенный символ синхронизации (по

		умолчанию отсутствует)
	**Vertical Synchronization**	Добавляет на диаграмму вертикально расположенный символ синхронизации (по умолчанию отсутствует)
	**Decision**	Добавляет на диаграмму символ принятия решения для альтернативных *переходов* (по умолчанию отсутствует)

Для построения диаграммы *состояний* следует выполнить следующие действия:

1. Добавить *состояния* с именами: Ожидание ввода ПИН-кода, Проверка ПИН-кода, Ожидание выбора клиента, Обработка запроса на снятие наличных, Обработка запроса на получение справки, Выдача наличных, Печать, Возврат карточки, Завершение транзакции и финальное *состояние*.
2. Добавить *переход*: карточка вставлена, направленный от *состояния* Ожидание карточки к *состоянию* Ожидание ввода ПИН-кода.
3. Добавить *переход*: ПИН-код введен, направленный от *состояния* Ожидание ввода ПИН-кода к *состоянию* Проверка ПИН-кода.
4. Добавить *переход*: отмена транзакции, направленный от *состояния* Ожидание ввода ПИН-кода к *состоянию* Возврат карточки.
5. Добавить *переход* со *сторожевым условием*: [ПИН-код верный], направленный от *состояния* Проверка ПИН-кода к *состоянию* Ожидание выбора клиента.
6. Добавить *переход* со *сторожевым условием*: [ПИН-код неверный], направленный от *состояния* Проверка ПИН-кода к *состоянию* Ожидание ввода ПИН-кода.
7. Добавить *переход*: три неудачи с *действием на переходе* конфискация карточки, направленный от *состояния* Проверка ПИН-кода к *состоянию* Завершение транзакции. Для задания *действия на данном переходе* следует ввести текст конфискация карточки в поле ввода Action (Действие) на вкладке Detail (Подробно) окна спецификации свойств данного *перехода*.

8. Добавить *переход*: выбор суммы со *сторожевым условием*: [сумма введена], направленный от *состояния* Ожидание выбора клиента к *состоянию* Обработка запроса на снятие наличных.

9. Добавить *переход*: выбор справки, направленный от *состояния* Ожидание выбора клиента к *состоянию* Обработка запроса на получение справки.

10. Добавить *переход*: отмена транзакции, направленный от *состояния* Ожидание выбора клиента к *состоянию* Возврат карточки.

11. Добавить *переход* со *сторожевым условием*: [кредит не превышен], направленный от *состояния* Обработка запроса на снятие наличных к *состоянию* Выдача наличных.

12. Добавить *переход* со *сторожевым условием*: [кредит превышен] с *действием на переходе* сообщение, направленный от *состояния* Обработка запроса на снятие наличных к *состоянию* Возврат карточки.

13. Добавить *переход*: наличные выданы со *сторожевым условием*: [выбрана печать чека], направленный от *состояния* Выдача наличных к *состоянию* Печать.

14. Добавить *переход*: наличные выданы со *сторожевым условием*: [печать чека не выбрана], направленный от *состояния* Выдача наличных к *состоянию* Возврат карточки.

15. Добавить *переход*: справка сформирована, направленный от *состояния* Обработка запроса на получение справки к *состоянию* Печать.

16. Добавить *переход*: печать закончена, направленный от *состояния* Печать к *состоянию* Возврат карточки.

17. Добавить *переход*: карточка возвращена, направленный от *состояния* Возврат карточки к *состоянию* Завершение транзакции.

18. Добавить *переход*: транзакция завершена, направленный от *состояния* Завершение транзакции к *состоянию* Ожидание карточки.

19. Добавить *переход*, направленный от *состояния* Ожидание карточки к финальному *состоянию*.

Диаграмма *состояний* для рассматриваемой модели банкомата будет иметь следующий вид (рис. 6.1).

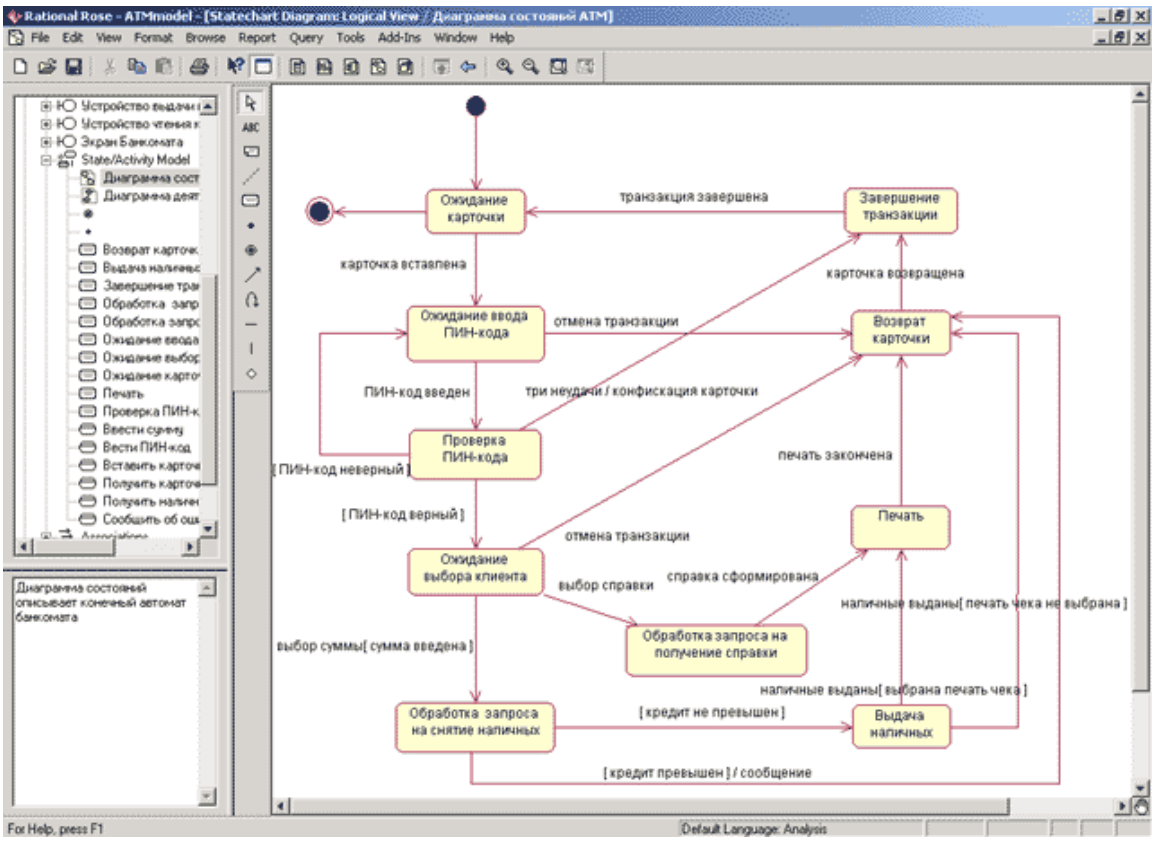


Рис. 6.1. Диаграмма состояний для моделирования поведения банкомата

Контрольные вопросы

1. Поясните два подхода к моделированию поведения системы. Объясните достоинства и недостатки каждого из этих подходов.
2. Охарактеризуйте вершины и дуги диаграммы схем состояний. В чем состоит назначение этой диаграммы?
3. Как отображаются действия в состояниях диаграммы схем состояний?
4. Как показываются условные переходы между состояниями?
5. Как задаются вложенные состояния в диаграммах схем состояний?
6. Поясните понятие исторического подсостояния.

Лабораторная работа №7

РЕАЛИЗАЦИЯ КОМПОНЕНТОВ ПРОГРАММНЫХ СРЕДСТВ

Программа выполнения работы

1.Изучить методику представления на физическом уровне элементов программных систем с помощью диаграмм компонентов UML.

2.Изучить технологию построения диаграмм компонентов UML в среде Rational Rose.

Методически указания

Диаграмма *компонентов* служит частью физического представления модели, играет важную роль в процессе ООАП и является необходимой для генерации программного кода. Общие рекомендации по построению диаграммы *компонентов* были рассмотрены в лекции 12 курса «Основы объектно-ориентированного моделирования в нотации UML». Для разработки диаграмм *компонентов* в браузере проекта предназначено отдельное представление *компонентов* (Component View), в котором уже содержится диаграмма *компонентов* с пустым содержанием и именем по умолчанию Main (Главная).

Активизация диаграммы *компонентов* может быть выполнена одним из следующих способов:

- Щелкнуть на кнопке с изображением диаграммы *компонентов* на стандартной панели инструментов.
- Раскрыть представление *компонентов* в браузере (Component View) и дважды щелкнуть на пиктограмме Main (Главная).
- Через пункт меню Browse→Component Diagram (Браузер→ Диаграмма *компонентов*).

В результате выполнения этих действий появляется новое окно с чистым рабочим листом диаграммы *компонентов* и специальная панель инструментов, содержащая кнопки с изображением графических примитивов, необходимых для разработки диаграммы *компонентов* (табл. 7.1).

Таблица 7.1.

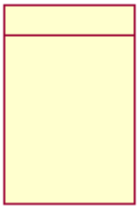
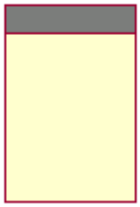
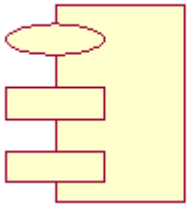
Назначение кнопок специальной панели инструментов диаграммы компонентов

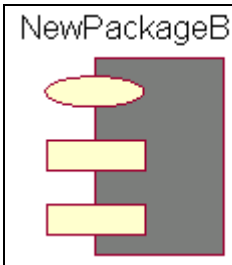
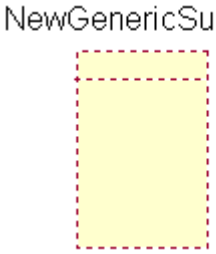
Графическое изображение	Всплывающая подсказка	Назначение кнопки
	**Selection Tool**	Превращает изображение курсора в форму стрелки для последующего выделения элементов на диаграмме
	**Text Box**	Добавляет на диаграмму текстовую область
	**Note**	Добавляет на диаграмму примечание
	**Anchor Note to Item**	Добавляет на диаграмму связь примечания с соответствующим графическим элементом диаграммы
	**Component**	Добавляет на диаграмму *компонент*
	**Package**	Добавляет на диаграмму пакет
	**Dependency**	Добавляет на диаграмму отношение зависимости
	**Subprogram Specification**	Добавляет на диаграмму спецификацию подпрограммы
	**Subprogram Body**	Добавляет на диаграмму тело подпрограммы
	**Main Program**	Добавляет на диаграмму главную программу
	**Package Specification**	Добавляет на диаграмму спецификацию пакета
	**Package Body**	Добавляет на диаграмму тело пакета
	**Task Specification**	Добавляет на диаграмму спецификацию задачи
	**Task Body**	Добавляет на диаграмму тело задачи
	**Generic Subprogram**	Добавляет на диаграмму типовую подпрограммы(по умолчанию отсутствует)
	**Generic Package**	Добавляет на диаграмму типовой пакет (по умолчанию отсутствует)
	**Database**	Добавляет на диаграмму базу данных (по умолчанию отсутствует)

Как видно из этой таблицы, по умолчанию на панели инструментов отсутствуют только три графических элемента из рассмотренных ранее элементов диаграммы *компонентов*, а именно - кнопки с пиктограммами типовой подпрограммы, типового пакета и базы данных. При необходимости их можно добавить на специальную панель диаграммы *компонента* стандартным способом.

Таблица 7.2.

Графическое изображение стереотипов компонентов и их характеристика

Графическое изображение и имя по умолчанию	Название стереотипа	Характеристика стереотипа *компонента*
NewSubprogSpec 	**Subprogram Specification**	Спецификация подпрограммы. Содержит описание переменных, процедур и функций и не содержит определений классов
NewSubprogBody 	**Subprogram Body**	Тело подпрограммы. Содержит реализацию процедур и функций, не относящихся к каким-то классам, при этом не содержит определений классов или реализаций операций других классов
NewMainSubprog 	**Main Program**	Главная программа. Реализует базовую логику работы программного приложения и содержит ссылки на другие *компоненты* модели
NewPackageSpec 	**Package Specification**	Спецификация пакета. Содержит определение класса, его атрибутов и операций. В языке программирования C++ спецификации пакета соответствует отдельный файл с расширением «h»

	**Package Body**	Тело пакета. Содержит код реализации операций класса. В языке программирования C++ спецификации пакета соответствует отдельный файл с расширением «cpp»
	**Task Specification**	Спецификация задачи. Может содержать определение класса, его атрибутов и операций, которые предполагается использовать в независимом потоке управления
	**Task Body**	Тело задачи. Может содержать реализацию операций класса, которые имеют независимый поток управления.
	**Generic Subprogram**	Типовая подпрограмма. Содержит описание переменных, процедур и функций, которые могут быть использованы в нескольких программных приложениях. При этом типовая подпрограмма не содержит определений классов
	**Generic Package**	Типовой пакет. Содержит определение класса, его атрибутов и операций, которое может быть использовано в нескольких программных приложениях
	**Database**	База данных. Содержит определение одного или нескольких классов, их атрибутов и, возможно, операций. При этом соответствующие классы могут быть реализованы в форме одной или нескольких таблиц базы данных

Использование рассмотренных *стереотипов* существенно увеличивают наглядность графического представления диаграммы *компонентов* и позволяют архитектору уточнить характер реализации модели программистом на выбранном языке программирования.

Для завершения построения диаграммы *компонентов* рассматриваемого примера следует выполнить следующие действия:

1. Добавить *компонент* с именем: Устройства Банкомата, для которого задать стереотип Task Specification.

2. Добавить *компоненты* с именами: Устройство чтения карточки, Клавиатура Банкомата, Принтер Банкомата, Экран Банкомата, Устройство выдачи наличных, для которых задать стереотип Task Body.

3. Добавить *зависимость* от *компонента* с именем MainATM.exe к *компоненту* с именем Устройства Банкомата.

4. Добавить *зависимость* от *компонента* с именем Устройство чтения карточки к *компоненту* с именем Устройства Банкомата.

5. Добавить *зависимость* от *компонента* с именем Клавиатура Банкомата к *компоненту* с именем Устройства Банкомата.

6. Добавить *зависимость* от *компонента* с именем Принтер Банкомата к *компоненту* с именем Устройства Банкомата.

7. Добавить *зависимость* от *компонента* с именем Экран Банкомата к *компоненту* с именем Устройства Банкомата.

8. Добавить *зависимость* от *компонента* с именем Устройство выдачи наличных к *компоненту* с именем Устройства Банкомата.

Построенная таким образом диаграмма *компонентов* будет иметь следующий вид (рис. 7.1).

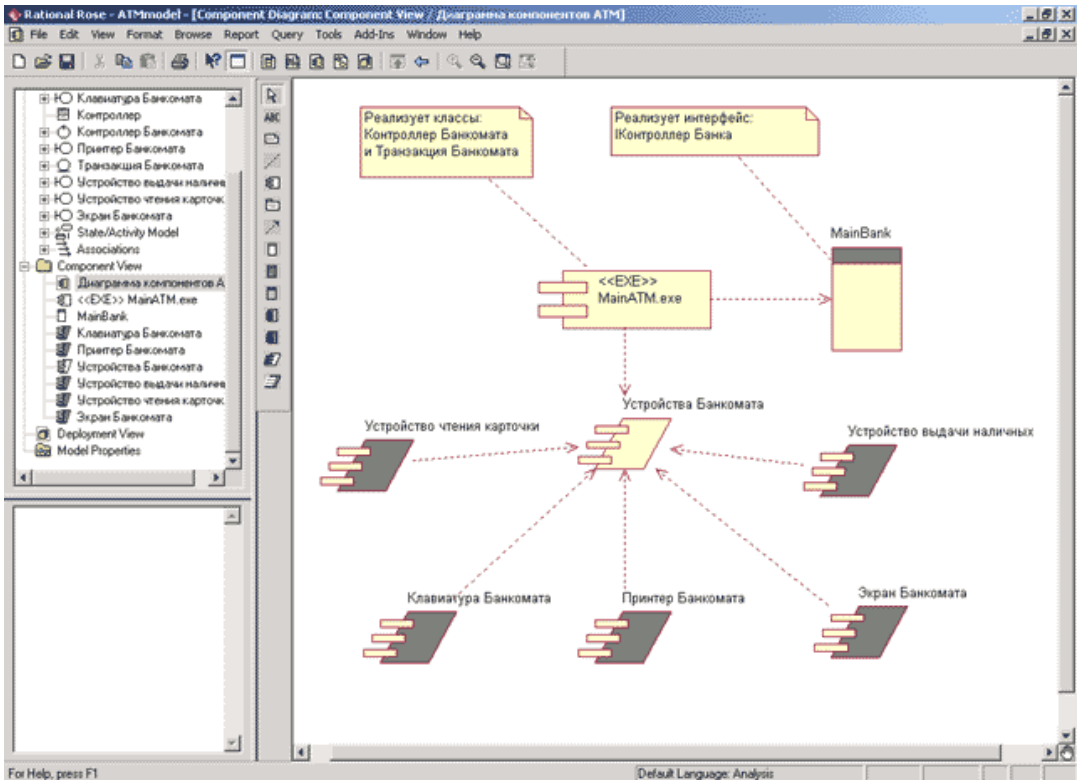


Рис. 7.1. Окончательный вид диаграммы компонентов разрабатываемой модели управления

Лабораторная работа №8

ТЕСТИРОВАНИЕ И ОТЛАДКА ПРОГРАММНЫХ СРЕДСТВ

Цель работы: получить навыки тестирования модулей в среде MS Visual Studio .Net

Программа выполнения работы

1. Изучить возможности модульного тестирования в MS Visual Studio средствами Nunit и Unit Testing Framework.
2. Выполнить тестирование заданного модуля программы на языке C#

Содержание отчета

1.Тестовые варианты.
2. Протокол тестирования заданных модулей (классов)

Методически указания

Unit-тестирование средствами .NET

Основная идея юнит (или модульного, как его еще называют) тестирования – тестирование отдельных компонентов программы, т.е. классов и их методов. При работе в среде Visual Studio .Net получили распространения следующие два средства модульного тестирования: Nunit и Unit Testing Framework.

Unit Testing Framework — это встроенная в Visual Studio система тестирования, разрабатываемая Майкрософт, постоянно развивающаяся. Ее особенность - хорошая интеграция в IDE и функция подсчета процента покрытия кода в программе.

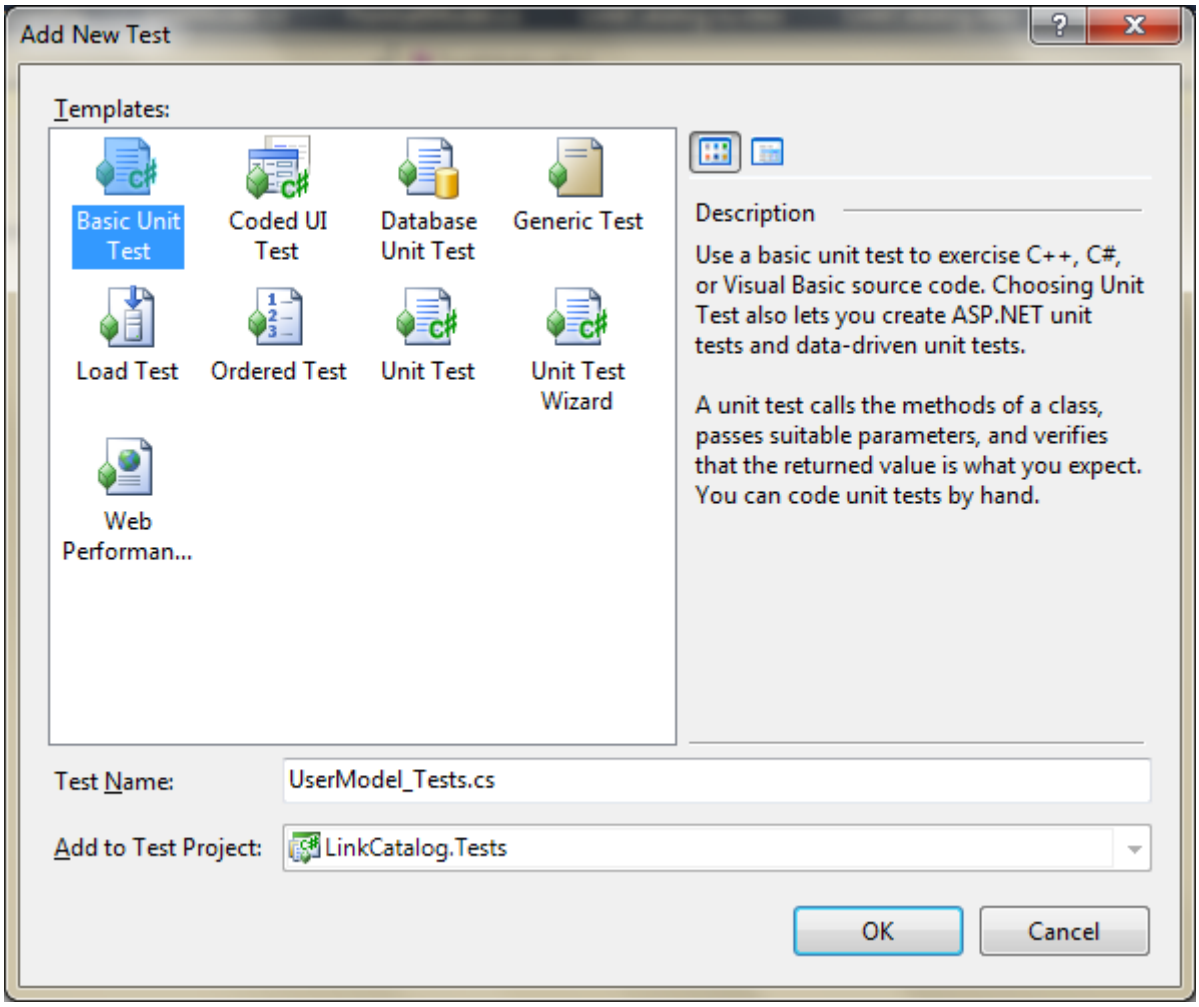
Юнит-тесты кода

Рассмотрим тестированию кода на примере небольшого ASP.NET приложение с реализацией шаблона MVC. Модель представляет собой отдельную сборку под названием DAL и включает в себя враппер доступа к БД. Одним из требований было использование именно DataReader в ADO.NET. Настройки, как это принято, хранятся в web.config.

Код для тестирования

```
namespace LinkCatalog.DAL
{
  public class UserModel
  {
...........
public static int GetUserIdByName(string username)
    {
      string query = "SELECT ID FROM Users WHERE Login = @login;";
      DB.get().CommandParameters.Add(new SqlParameter("@login", username));

      int id = -1;

      int.TryParse(DB.get().GetOneCell(query).ToString(), out id);

      return id;
    }
  }
  public class DB
  {
...........
private static DB instance;

    public static DB get()
    {
      if (instance == null)
        instance = new DB();

      return instance;
    }

    private SqlConnection connection;
    private SqlDataReader reader;

    public List CommandParameters;
    private DB()
    {
      this.connection = newSqlConnection(WebConfigurationManager.ConnectionS
trings["DBConnectionString"].ConnectionString);
      this.CommandParameters = new List();
    }

   public object GetOneCell(string query)
    {
      SqlCommand sc = new SqlCommand(query, this.connection);

      if (this.CommandParameters.Count != 0)
        sc.Parameters.AddRange(this.CommandParameters.ToArray());

      this.connection.Open();

      object res = sc.ExecuteScalar();
      this.CommandParameters.Clear();

     this.Close();
      return res;
    }
  }
}

* This source code was highlighted with Source Code Highlighter.
```

Добавляем в проект новый юнит-тест



и получаем файл такого содержания:

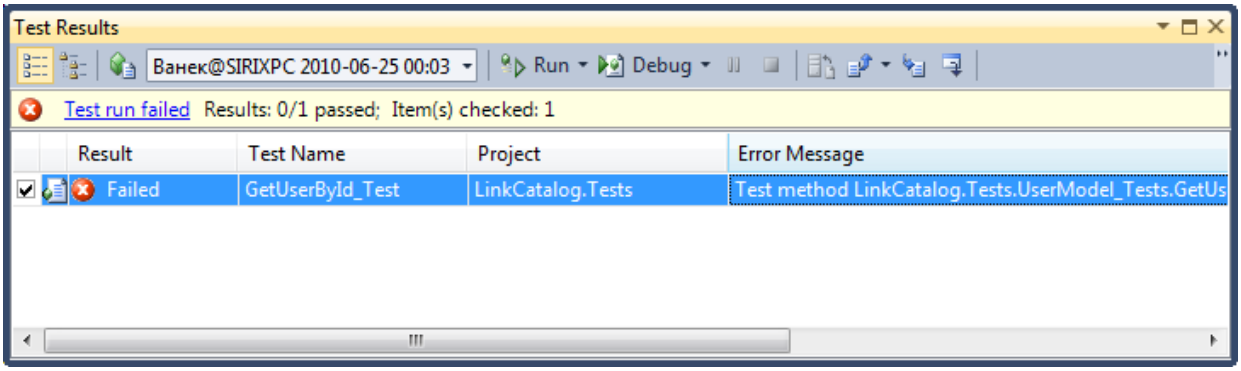
```
using System;
using System.Text;
using System.Collections.Generic;
using System.Linq;
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace LinkCatalog.Tests
{
  [TestClass]
  public class UserModel_Tests
  {
    [TestMethod]
    public void TestMethod1()
    {
    }
  }
}

* This source code was highlighted with Source Code Highlighter.
```

Атрибут [TestClass] означает, что этот класс содержит тестовые методы, а [TestMethod] – что такой метод представляет собой конкретный метод.

Добавляем в проект ссылку на сборку DAL и импортируем пространство имен LinkCatalog.DAL. Подготовительные работы закончены, настало время писать тесты.

```
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace LinkCatalog.Tests
{
  using LinkCatalog.DAL;
  [TestClass]
  public class UserModel_Tests
  {
    [TestMethod]
    public void GetUserById_Test()
    {
      Assert.AreNotEqual(UserModel.GetUserIdByName("Admin"), -1);
    }
  }
}

* This source code was highlighted with Source Code Highlighter.
```

У нас всегда есть администратор с таким логином и потому его идентификатор не может быть -1. Запускаем тест – и ошибка:



Выскочило исключение:

```
Test method LinkCatalog.Tests.UserModel_Tests.GetUserById_Test threw excep-
tion:
System.NullReferenceException: Object reference not set to an instance of
an object.
LinkCatalog.DAL.DB..ctor() in C:\Users\Ванек\Documents\Visual Studio
2010\Projects\Practice\DAL\Database.cs: line 35
LinkCatalog.DAL.DB.get() in C:\Users\Ванек\Documents\Visual Studio
2010\Projects\Practice\DAL\Database.cs: line 17
LinkCatalog.DAL.UserModel.GetUserIdByName(String username) in C:\Users\Ванек
\Documents\Visual Studio 2010\Projects\Practice\DAL\Models\UserModel.cs:
line 63
LinkCatalog.Tests.UserModel_Tests.GetUserById_Test() in C:\Users\Ванек\Docum
```

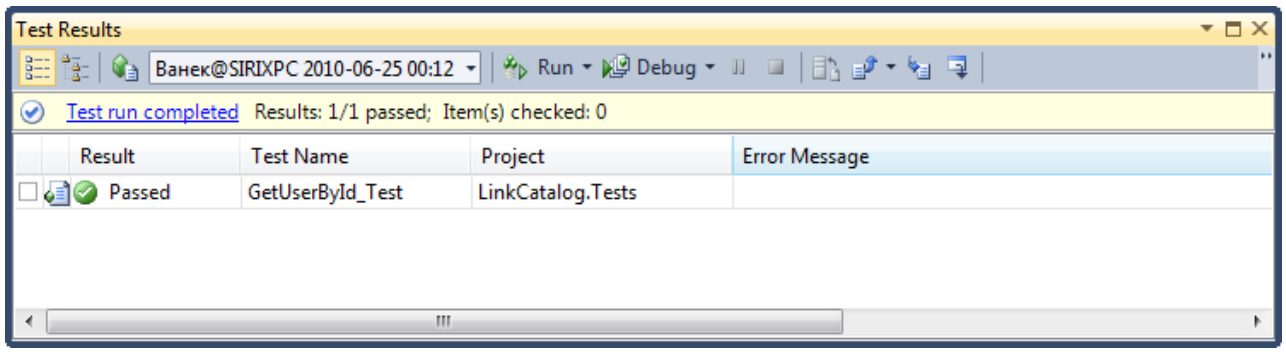
```
ents\Visual Studio
2010\Projects\Practice\LinkCatalog.Tests\UserModel_Tests.cs: line 12

* This source code was highlighted with Source Code Highlighter.
```

Как видно, ошибка заключается в конструкторе класса DB – он не может найти файл конфигурации и, вследствие этого, тест завершается не только неудачно, но и с ошибкой.

Решение проблемы конфигурационных файлов довольно простое. Среда тестирования не подключит web.config автоматически. Вместо этого каждый тест-проект создает свой файл конфигурации app.config, и все, что требуется – дописать в него необходимые настройки.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>

 <configSections>
  <sec-
tion name="DatabaseUnitTesting"type="Microsoft.Data.Schema.UnitTesting.Confi
guration.DatabaseUnitTestingSection, Microsoft.Data.Schema.UnitTesting, Ver-
sion=10.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />
 </configSections>
 <DatabaseUnitTesting>
  <DataGeneration ClearDatabase="true" />
  <ExecutionContext Provider="System.Data.SqlClient" ConnectionString="Data
Source=SIRIXPC\sqlexpress;Initial Catalog=TestDB;Integrated Securi-
ty=True;Pooling=False"
    CommandTimeout="30" />
  <PrivilegedContext Provider="System.Data.SqlClient" ConnectionString="Data
Source=SIRIXPC\sqlexpress;Initial Catalog=TestDB;Integrated Securi-
ty=True;Pooling=False"
    CommandTimeout="30" />
 </DatabaseUnitTesting>


 <connectionStrings>
  <add name="DBConnectionString" connectionString="Data
Source=SIRIXPC\SQLEXPRESS;Initial Catalog=tbd;Integrated Security=True"
    providerName="System.Data.SqlClient" />
 </connectionStrings>

</configuration>

* This source code was highlighted with Source Code Highlighter.
```

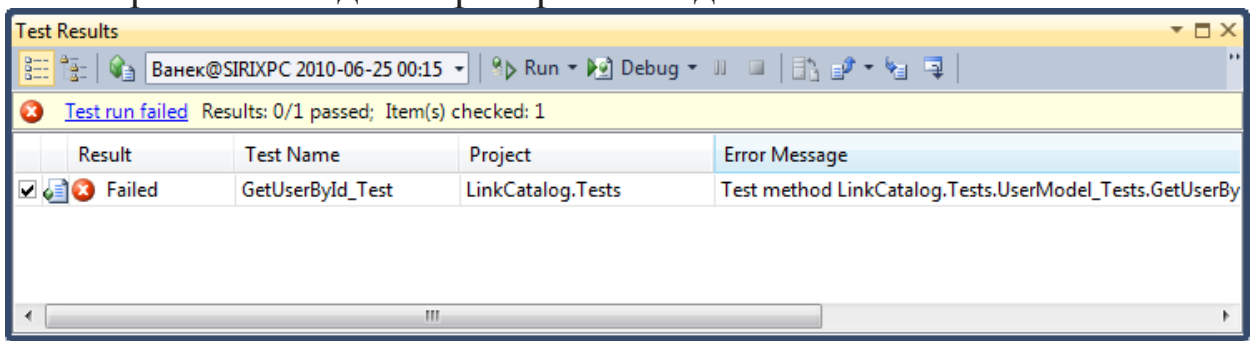
Теперь тест проходит успешно:



Немного изменим тест

```
    [TestMethod]
    public void GetUserById_Test()
    {
      Assert.AreNotEqual(UserModel.GetUserIdByName("Admin"), -1);

      Assert.AreEqual(UserModel.GetUserIdByName("0-934723  ### 12sdf s"), -
1);
    }

* This source code was highlighted with Source Code Highlighter.
```

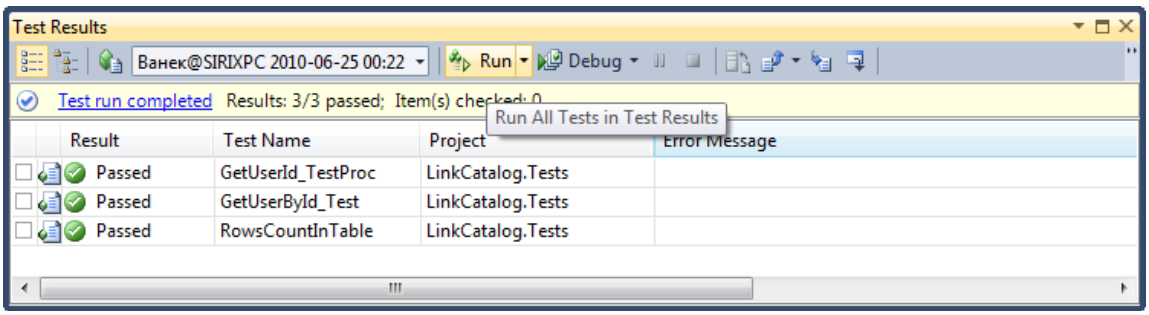
В базе гарантированно нет такого логина(логин должен быть без пробелов, за этим следят валидаторы в контроллере при регистрации), и это вторая и последняя проверка метода.



Тест опять не проходится – смотрим детализацию:

```
Test method LinkCatalog.Tests.UserModel_Tests.GetUserById_Test threw excep-
tion:
System.NullReferenceException: Object reference not set to an instance of
an object.

На этот раз ошибка заключается в UserModel.GetUserIdByName, а именно вот
здесь не хватает проверки на null:
      int.TryParse(DB.get().GetOneCell(query).ToString(), out id);
Добавляем:
var res = DB.get().GetOneCell(query);
      if (res != null)
        int.TryParse(res.ToString(), out id);

* This source code was highlighted with Source Code Highlighter.
```

Теперь получим положительный результат:



Контрольные вопросы

1. Назовите средства автоматизации тестирования ПС на уровне модулей.
2. Дайте сравнительную характеристику возможностей Nunit и Unit Testing Framework.
3. Какие существуют виды тестирования?
4. Какие Вы знаете методики структурного тестирования?
5. Какие Вы знаете методики функционального тестирования?
6. Какие известны подходы тестирования интеграции программной системы?
7. Как оценить количество необходимых тестов для структурного тестирования модуля?
8. В чем особенности тестирования классов?
9. Способы тестирования циклов?
10. Способы тестирования условий?

Лабораторная работа №9

ВЫЧИСЛЕНИЕ МЕТРИК ПРОГРАММНЫХ СИСТЕМ

Цель работы: изучить метрики для количественной оценки программного кода и возможности их получения в современных инструментальных средах

Программа выполнения работы

1.Изучить системы метрик программных средств.
2. Выполнить расчет метрик классов для заданного примера.

Содержание отчета

1. Диаграмма UML классов для заданного программного проекта.
2. Метрики, рассчитанные по диаграмме классов.
3. Метрики, полученные автоматически в VisualStudio.

Методически указания

Метрика программного обеспечения (англ. Software metric) – это некая мера определенного свойства программного обеспечения или же его спецификаций. Как известно, мера – это числовое значение. Таким образом, метрика программного обеспечения будет показывать некое числовое значение определенного свойства ПО.

Метрики в Visual Studio

Использована версия Visual Studio 2013. Откроем проект для изучения.

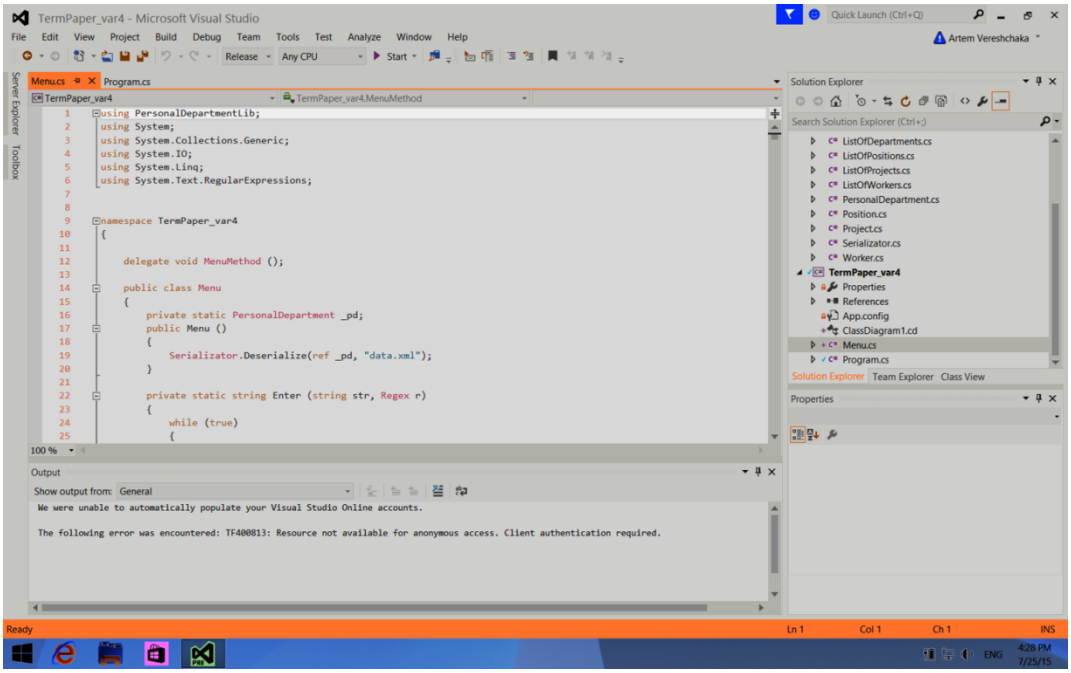


Рис.9.1.

Далее, мы можем видеть вкладку Analyze

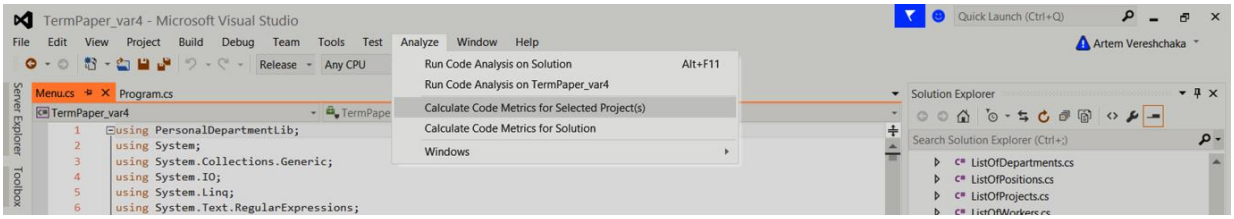


Рис.9.2.

В этой вкладке мы видим Calculate Code Metrics for ...

Это нам и нужно. Разница лишь в том, что будет анализироваться. Или же выбранные проекты в Solution Explorer, или же сразу весь Solution. После нажатия придется немного подождать. Время зависит от конфигурации Вашего компьютера. Когда анализ будет завершен, Вы увидите внизу окно

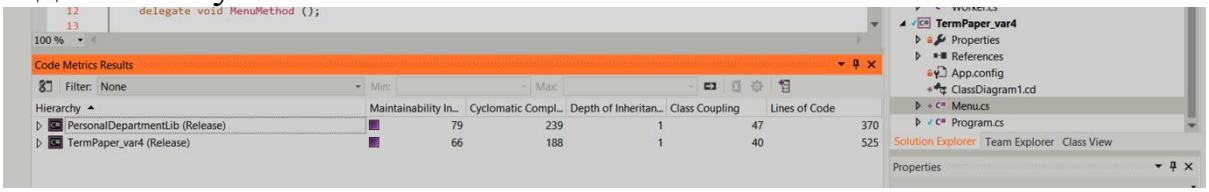


Рис.9.3.

Здесь будет видна иерархия всего Solution. В моем случае это отдельная dll библиотека и проект. Когда развернем библиотеку, мы увидим следующий уровень иерархии, и так далее

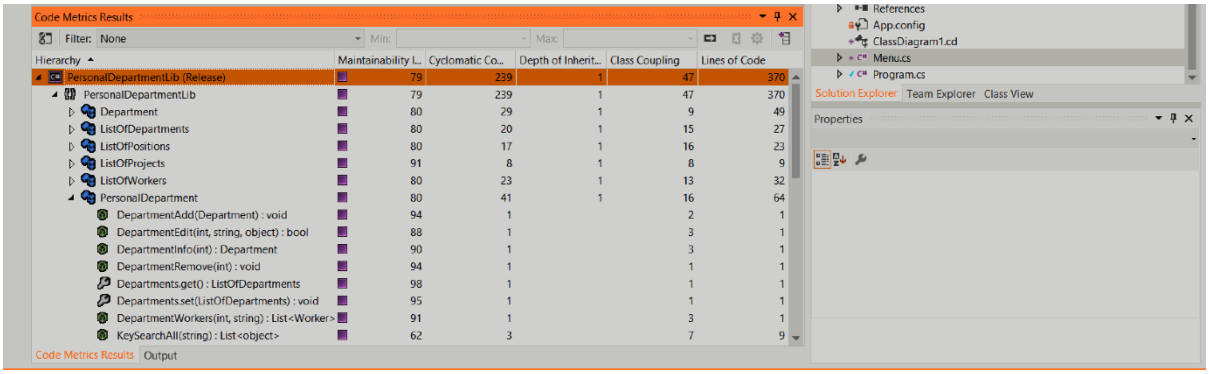


Рис.9.4.

Теперь давайте разберемся со столбцами дальше.

1. Maintainability Index – это комплексный показатель качества кода. Эта метрика рассчитывается по следующей формуле:

MI = MAX(0, (171 — 5.2 * ln(HV) — 0.23 * CC — 16.2 * ln(LoC)) * 100 / 171),

где HV – Halstead Volume, вычислительная сложность. Чем больше операторов, тем больше значение этой метрики; CC – Cyclomatic Complexity (Эта метрика описана ниже); LoC – количество строк кода (Эта метрика описана ниже).

2. Cyclomatic Complexity – показывает структурную сложность кода. Иными словами, количество различных ветвей кода. Считается на основе операторов в Вашем коде, строя графы переходов от одного оператора к другому. К примеру, оператор if-else увеличит эту метрику, потому что здесь будут разные ветви выполнения.

3. Depth of Inheritance – глубина наследования. Для каждого класса эта метрика показывает, насколько глубоко он в цепочке наследования.

4. Class Coupling – указывает на зависимость классов друг от друга. Проект с множеством зависимостей очень трудно и дорого поддерживать.

5. Lines of Code – количество строк кода. Напрямую используется редко. В наши дни, с множеством разнообразных как подходов к программированию, так и языков, эта метрика дает нам мало полезной информации. Если брать во внимание отдельный метод, то можно разбить его на несколько методов поменьше.

Использования метрик

Изначально стоит обращать внимание на Maintainability Index. Старайтесь придерживать его около 70-90. Это значительно облегчит сопровождения кода как Вами, так и другими программистами. Ино-

гда стоит оставить его на уровне 50-60, так как переписать некоторые участки кода бывает очень затратным. Оценивайте здраво как код, так и Ваши возможности с затратами.

Стоит также уделить много внимания Class Coupling. Эта метрика должна быть как можно меньшей. Ведь она так же способствует поддержке кода. Для оптимизации возможно придется пересматривать дизайн проекта и некоторые архитектурные решения.

Теперь стоит уделить внимание Cyclomatic Complexity. Эта метрика показывает сложность кода, а это так же влияет на поддержку кода в будущем. Иногда приходится переписывать куски кода, которые писали до Вас другие люди, так как Вы просто не можете понять, что, как и зачем в этом методе. Конечно, этому еще способствует стиль кода и идея, но не забывайте о Cyclomatic Complexity при рефакторинге.

Контрольные вопросы

1. Какие факторы объектно-ориентированных систем влияют на метрики для их оценки и как проявляется это влияние?
2. Какое влияние оказывает наследование на связность классов?
3. Охарактеризуйте метрики связности классов по данным.
4. Охарактеризуйте метрики связности классов по методам.
5. Какие характеристики объектно-ориентированных систем ухудшают сцепление классов?
6. Объясните, как определить сцепление классов с помощью метрики «зависимость изменения между классами».
7. Поясните смысл метрики локальности данных.
8. Какие метрики входят в набор Чидамбера и Кемерера? Какие задачи они решают?
9. Как можно подсчитывать количество методов в классе?
10. Какие метрики Чидамбера и Кемерера оценивают сцепление классов? Поясните их смысл.
11. Какая метрика Чидамбера и Кемерера оценивает связность класса? Поясните ее смысл.
12. Как добиться независимости метрики WMC от реализации?
13. Дайте характеристику метрик для объектно-ориентированного тестирования.

СПИСОК ЛИТЕРАТУРЫ

1. Иванова Г. С. Технология программирования: учебник / Иванова Г. С.;. - 2-е изд., стер.-М.:МГТУ,2003.-320 с.
2. Орлов С.А. Технологии разработки программного обеспечения. Разработка сложных программных систем: учебник / Орлов С. А.; .- 3-е изд.. - СПб.: ПИТЕР, 2004.-527 с.
3. Вендров А.М. Проектирование программного обеспечения экономических информационных систем - М.: Финансы и статистика, 2003.-352с.
4. М. Фаулер, К. Скотт. UML в кратком изложении. Применение стандартного языка объектного моделирования: Пер. с англ. – М.: Мир, 1999. – 191 с.
5. ГОСТ Р ИСО/МЭК 9294-93. Информационная технология. Руководство по управлению документированием программного обеспечения. [Электронный ресурс].- Режим доступа: http://195.209.112.90:3000/texpert/
6. 10.ГОСТ Р ИСО/МЭК 9126-93. Информационная технология. Оценка программной продукции, характеристика качества и руководство по их применению. [Электронный ресурс].- Режим доступа: http://195.209.112.90:3000/texpert/
7. 11.ГОСТ Р ИСО/МЭК 12119:1994. Информационная технология. Пакеты программных средств. Требования к качеству и испытания. [Электронный ресурс].- Режим доступа: http://195.209.112.90:3000/texpert/
8. 12.ГОСТ Р ИСО/МЭК 12207-99. Процессы жизненного цикла программных средств. [Электронный ресурс].- Режим доступа: http://195.209.112.90:3000/texpert/
9. Орлик С. Основы программной инженерии по SWEBOK [Электронный ресурс] Режим доступа http://swebok.sorlik.ru
10. 20. Кознов Д.В., Бугайченко Д.Ю. Введение в программную инженерию – Учебный курс ИНТУИТ.РУ. [Электронный ресурс] Режим доступа http://www.intuit.ru/department/se/inprogeng/ .

Учебно-методическое издание

ПРОГРАММНАЯ ИНЖЕНЕРИЯ ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Учебно-методическое пособие
к лабораторным работам

Составитель - **Зуев** Владимир Алексеевич

Редактор *Н.А. Юшко*
Компьютерная верстка авторская
Подписано в печать 05.10. 2013 г.
Формат 60x84$^{1}/_{16}$. Бумага офсетная. Печать цифровая.
Усл.-печ.л.3,25. Уч.-изд. л. 3,31. Тираж 50 экз. Заказ _____

Южно-Российский государственный технический университет (НПИ)
им. М.И. Платова
Редакционно-издательский отдел ЮГРПУ(НПИ)
346428, Новочеркасск, ул. Просвещения, 132

Издательство ЛИК
346430, Новочеркасск, пр. Платовский, 82Е
Тел.: 8(8635) 226-442, 8-952-603-609, e-mail: center-op@mail.ru

Отпечатано в Издательско-полиграфическом комплексе «Колорит»
346430, Новочеркасск, пр. Платовский, 82Е
Тел.: 8(8635) 226-442, 8-952-603-609, e-mail: center-op@mail.ru