# Введение

Тема реферата по дисциплине «Проектирование интеллектуальных систем» «Основы объектно-ориентированного проектирования».

Целью изучения дисциплины является подготовка специалистов в области автоматизации сложноформализуемых задач. Задачей изучения дисциплины является приобретение знаний о фундаментальных алгоритмах, применяемых при построении систем искусственного интеллекта, а также методов разработки программных приложений, реализующих эти системы.

Принципиальное отличие интеллектуальных систем от любых других систем автоматизации заключается в наличии базы знаний о предметной среде, в которой решается задача. Неинтеллектуальная система при отсутствии каких-либо входных данных прекращает решение задачи, интеллектуальная же система недостающие данные извлекает из базы знаний и решение выполняет.

По А.Н. Колмогорову, любая материальная система, с которой можно достаточно долго обсуждать проблемы науки, литературы и искусства, обладает интеллектом. Такое определение показывает, что данная дисциплина находится во взаимосвязи практически со всеми учебными дисциплинами. Тем не менее, следует подчеркнуть связи со следующими дисциплинами: «Программирование», «Математический анализ», «Линейная алгебра и аналитическая геометрия», «Дискретная математика», «Логическое программирование», «Экспертные системы», «Интерфейсы интеллектуальных систем».

**1. Понятие о классах и объектах**

Проникновение классов и объектов в программирование произошло на основе известных принципов классификации, применяемых в различных отраслях науки – зоологии, биологии, химии и т.д. Со школьной скамьи известна систематизация живых организмов, предложенная К. Линнеем. Основой распределения животных по видам, подвидам, группам является наличие ряда общих особенностей. Например, общей чертой всех животных, относимых к млекопитающим, является кормление детенышей молоком; для членистоногих характерна общая черта строения конечностей и т.д. Следует, подчеркнуть отличие между видом и представителем вида: вид – один, представителей (экземпляров) вида – множество.

Таким образом, основным принципом отнесения какого-либо предмета реального мира к тому или иному классу является следующий: есть или нет у этого предмета характеристика (или совокупность характеристик), такая же, как и у остальных предметов, входящих в класс. Например, изображенные на рисунке 2 объекты могут быть отнесены к классу компьютеров, если они позволяют вводить описания, выполнять и отображать результаты некоторых последовательностей действий. На рисунке 2 представлены четыре экземпляра класса (понятия) «компьютер». Рисунок 2 отображает также и другую особенность – классы могут содержать в качестве составных частей другие классы. Используя информацию рисунка, можно выделить класс «Клавиатура», класс «Дисплей», класс «Системный блок».

Термин «общая черта», использованный при определении понятия «класс», предполагает наличие у предметов реального мира не только одинакового набора характеристик, но и одинакового поведения в одинаковых ситуациях. Таким образом, определение класса должно включать определение характеристики (совокупности характеристик), общей для всех представителей класса, и описаний действий, выполняемых любым представителем класса в различных ситуациях.

Объекты реального или виртуального мира находятся во взаимосвязях друг с другом. Связи могут воплощаться в виде реального объекта, как, например, соединительные кабели между клавиатурой и системным блоком, между дисплеем и системным блоком и т.д. Связи могут иметь духовную природу (например, брак между людьми) и оформляться документами. Очевидно, что при программой реализации связи между классами самый простой способ «документальный», т.е. когда одной из характеристик класса будет ссылка на класс (классы), связанный с данным.

Класс – это абстракция множества предметов реального мира, которые соответствуют следующим требованиям:

* все предметы в этом множестве – объекты (экземпляры) – имеют один и тот же набор характеристик(атрибутов) (значения характеристик могут быть разными);
* все объекты подчинены и согласовываются с одним и тем же набором правил и линий поведения;
* между объектами одного класса нет связей.

Каждый класс должен иметь уникальное и значимое имя.

Атрибут – это характеристика, которой обладают все объекты (экземпляры) класса. Каждый атрибут обеспечивается именем, уникальным в пределах класса.

Операция (метод) – это процедура (или функция), задающая последовательность действий любого представителя класса в некоторой ситуации.

## **2. Унифицированный язык моделирования**

История развития унифицированного языка моделирования – Unified Modeling Language (UML), – берет начало с октября 1994 года, когда Гради Буч и Джеймс Румбах из Rational Software Corporation начали работу по унификации методов Booch и Object Modeling Technique (ОМТ). Хотя сами по себе эти методы были достаточно популярны, совместная работа была направлена на изучение всех известных объектно-ориентированных методов с целью объединения их достоинств. При этом Г. Буч и Дж. Румбах сосредоточили усилия на полной унификации результатов своей работы.

Компания Rational Software вместе с несколькими организациями, изъявившими желание выделить ресурсы для разработки строгого определения версии 1.0 языка UML, учредила консорциум партнеров UML, в который первоначально вошли такие компании, как Digital Equipment Corp., HP, i-Logix, Intellicorp, IBM, ICON Computing, MCI Systemhouse, Microsoft, Oracle, Rational Software, TI и Unisys. Эти компании обеспечили поддержку последующей работы по более точному и строгому определению нотации, что привело к появлению версии 1.0 языка UML. В январе 1997 года был опубликован документ с описанием языка UML 1.0.

Очередной этап развития данного языка закончился в марте 1999 года, когда консорциумом OMG было опубликовано описание языка UML 1.3 (alpha R5). Статус языка UML определен как открытый для всех предложений по его доработке и совершенствованию. Сам язык UML не является чьей-либо собственностью и не запатентован кем-либо, хотя указанный выше документ защищен законом об авторском праве. В то же время аббревиатура UML, как и некоторые другие (OMG, CORBA, ORB), является торговой маркой их законных владельцев, о чем следует упомянуть в данном контексте.

Следует отметить внимание гиганта компьютерной индустрии компании Microsoft к технологии UML. Еще в октябре 1996 г. Microsoft и Rational Software Corporation объявили о своем стратегическом партнерстве, которое, по их общему мнению, способно оказать решающее влияние на рынок средств объектно-ориентированной разработки программ. При этом Microsoft лицензировала у Rational Software некоторые технологии визуального моделирования, в результате чего был разработан Microsoft Visual Modeler for Visual Basic. В свою очередь Rational Software лицензировала у Microsoft Visual Basic и Microsoft Repository, разрабатываемые вместе с Texas Instruments. При создании языка UML Microsoft внесла свой вклад в интеграцию UML со своими стандартами типа ActiveX и СОМ и в использование языка UML со своей технологией Microsoft Repository.

В настоящее время Rational Software Corporation объединила свои усилия по совершенствованию технологии UML с компанией IBM.

## **3. Представление класса**

Графически класс изображается в виде прямоугольника, который дополнительно может быть разделен горизонтальными линиями на разделы или секции (см. рисунок 3). В этих разделах могут указываться имя класса, атрибуты (переменные) и операции (методы).

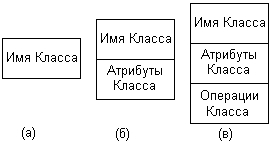


Рисунок 3 – Графическое изображение класса на диаграмме классов

Обязательным элементов обозначения класса является его имя. На начальных этапах разработки диаграммы отдельные классы могут обозначаться простым прямоугольником с указанием только имени соответствующего класса (рисунок 3, а). По мере проработки отдельных компонентов диаграммы, описания классов дополняются атрибутами (рисунок 3, б) и операциями (рисунок 3, в).

Предполагается, что окончательный вариант диаграммы содержит наиболее полное описание классов, которые состоят из трех разделов или секций. Иногда в обозначениях классов используется дополнительный четвертый раздел, в котором приводится семантическая информация справочного характера или явно указываются исключительные ситуации.

Даже если секция атрибутов и операций является пустой, в обозначении класса она выделяется горизонтальной линией, чтобы сразу отличить класс от других элементов языка UML. Примеры графического изображения классов на диаграмме классов приведены на рисунке 4. В первом случае для класса «Прямоугольник» (рисунке 4, а) указаны только его атрибуты – точки на координатной плоскости, которые определяют его расположение. Для класса «Окно» (рисунке 4, б) указаны только его операции, секция атрибутов оставлена пустой. Для класса «Счет» (рисунке 4, в) дополнительно изображена четвертая секция, в которой указано исключение – отказ от обработки просроченной кредитной карточки.

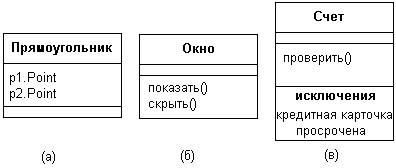


Рисунок 4 – Примеры графического изображения классов на диаграмме

Класс может не иметь экземпляров или объектов. В этом случае он называется абстрактным классом, а для обозначения его имени используется наклонный шрифт (курсив). В языке UML принято общее соглашение о том, что любой текст, относящийся к абстрактному элементу, записывается курсивом. Данное обстоятельство является семантическим аспектом описания соответствующих элементов языка UML.

### **Атрибуты**

Во второй сверху секции прямоугольника класса записываются его атрибуты (*attributes*) или свойства*.* В языке UML принята определенная стандартизация записи атрибутов класса, которая подчиняется некоторым синтаксическим правилам. Каждому атрибуту класса соответствует отдельная строка текста, структура которой показана на рисунке 5:



Рисунок 5 – Структура задания атрибута

Квантор видимости может принимать одно из трех возможных значений и отображается при помощи специальных символов:

* «+» обозначает атрибут с областью видимости типа общедоступный (public). Атрибут с этой областью видимости доступен или виден из любого другого класса пакета, в котором определена диаграмма;
* «#» обозначает атрибут с областью видимости типа защищенный (protected). Атрибут с этой областью видимости недоступен или невиден для всех классов, за исключением подклассов данного класса;
* «–» обозначает атрибут с областью видимости типа закрытый (private). Атрибут с этой областью видимости недоступен или невиден для всех классов без исключения.

Квантор видимости может быть опущен. Отсутствие квантора видимости обычно трактуется как private. Вместо условных графических обозначений можно записывать соответствующее ключевое слово: public, protected, private.

Имя атрибута должно быть уникальным в пределах данного класса. Имя атрибута является обязательным элементом синтаксического обозначения атрибута.

Кратность атрибута характеризует общее количество конкретных атрибутов данного типа, входящих в состав отдельного класса. В общем случае кратность записывается в форме строки текста в квадратных скобках после имени соответствующего атрибута:

[нижняя\_граница1. верхняя\_граница1, нижняя\_граница2. верхняя\_грашца2,…, нuжняя\_гpaнuцa k. верхняя\_граница k],

где нижняя\_граница и верхняя\_граница являются положительными целыми числами, каждая пара которых служит для обозначения отдельного замкнутого интервала целых чисел, у которого нижняя (верхняя) граница равна значению нижняя\_граница (верхняя\_граница). В качестве верхней\_границы может использоваться специальный символ «\*», который означает произвольное положительное целое число.

Значения кратности из интервала следуют в монотонно возрастающем порядке без пропуска отдельных чисел, лежащих между нижней и верхней границами. При этом соответствующие нижние и верхние границы интервалов включаются в значение кратности.

Тип атрибута представляет собой выражение, семантика которого определяется языком спецификации соответствующей модели. В нотации UML тип атрибута обычно определяется в зависимости от языка программирования, который предполагается использовать для реализации данной модели.

Исходное значение служит для задания некоторого начального значения для соответствующего атрибута в момент создания отдельного экземпляра класса. Здесь необходимо придерживаться правила принадлежности значения типу конкретного атрибута. Если исходное значение не указано, то значение соответствующего атрибута не определено на момент создания нового экземпляра класса.

### **Операция**

В третьей сверху секции прямоугольника записываются операции или методы класса. Операция (operation) представляет собой некоторый сервис, предоставляющий каждый экземпляр класса по определенному требованию. Совокупность операций характеризует функциональный аспект поведения класса. Запись операций класса в языке UML также стандартизована и подчиняется определенным синтаксическим правилам. При этом каждой операции класса соответствует отдельная строка, структура которой приведена на рисунке 6:



Рисунок 6 – Структура задания операции

Список параметров является перечнем разделенных запятой формальных параметров, каждый из которых может быть представлен в следующем виде (см. рисунок 7):



Рисунок 7 – Структура задания параметра

Здесь вид параметра – есть одно из ключевых слов in, out или inout со значением in по умолчанию, в случае если вид параметра не указывается.

Операция, которая не может изменять состояние системы и, соответственно, не имеет никакого побочного эффекта, обозначается строкой-свойством «{запрос}» («{query}»). В противном случае операция может изменять состояние системы, хотя нет никаких гарантий, что она будет это делать. При этом придерживаются следующей семантики для данных значений:

последовательная (sequential) – для данной операции необходимо обеспечить ее единственное выполнение в системе, одновременное выполнение других операций может привести к ошибкам или нарушениям целостности объектов класса.

параллельная (concurrent) – данная операция в силу своих особенностей может выполняться параллельно с другими операциями в системе, при этом параллельность должна поддерживаться на уровне реализации модели.

охраняемая (guarded) – все обращения к данной операции должны быть строго упорядочены во времени с целью сохранения целостности объектов данного класса, при этом могут быть приняты дополнительные меры по контролю исключительных ситуаций на этапе ее выполнения.

На рисунке 8 приведен пример задания класса, содержащего атрибуты и операции.



Рисунок 8 – Пример задания класса

## 

## **4. Представление отношения между классами**

Если класс оказывает влияние на другой класс, то на диаграмме эта взаимосвязь (отношение) между классами изображается соединительной линией. Анализ разновидностей взаимосвязей (в дальнейшем отношений) позволяет выделить следующие:

* отношение зависимости (dependency relationship);
* отношение ассоциации (association relationship);
* отношение обобщения (generalization relationship);
* отношение реализации (realization relationship).

Каждое из этих отношений имеет собственное графическое представление на диаграмме.

### **Отношение зависимости**

Отношение зависимости используется в такой ситуации, когда некоторое изменение одного элемента модели может потребовать изменения другого зависимого от него элемента модели. Отношение зависимости графически изображается пунктирной линией между соответствующими элементами со стрелкой на одном из ее концов. На диаграмме классов данное отношение связывает отдельные классы между собой, при этом стрелка направлена от класса-клиента зависимости к классу-серверу (рисунок 9).

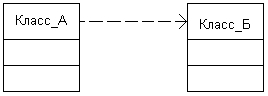


Рисунок 9 – Графическое изображение отношения зависимости

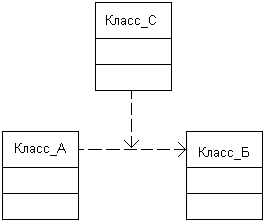


Рисунок 10 – Графическое представление зависимости между классом-клиентом (Класс\_С) и классами-источниками (Класс\_A и Класс\_Б)

Стрелка может помечаться необязательным, но стандартным ключевым словом в кавычках и необязательным индивидуальным именем. Для отношения зависимости предопределены ключевые слова, которые обозначают некоторые специальные виды зависимостей. Эти ключевые слова (стереотипы) записываются в кавычках рядом со стрелкой, которая соответствует данной зависимости. Примеры стереотипов для отношения зависимости представлены ниже:

«access» – служит для обозначения доступности открытых атрибутов и операций класса-источника для классов-клиентов;

«bind» – класс-клиент может использовать некоторый шаблон для своей последующей параметризации;

«derive» – атрибуты класса-клиента могут быть вычислены по атрибутам класса-источника;

«import» – открытые атрибуты и операции класса-источника становятся частью класса-клиента, как если бы они были объявлены непосредственно в нем;

«refine» – указывает, что класс-клиент служит уточнением класса-источника в силу причин исторического характера, когда появляется дополнительная информация в ходе работы над проектом.

Примечание

Отношение зависимости является наиболее общей формой отношения в языке UML. Все другие типы рассматриваемых отношений можно считать частным случаем данного отношения. Однако важность выделения специфических семантических свойств и дополнительных характеристик для других типов отношений обусловливают их самостоятельное рассмотрение при построении диаграмм.

### **Отношение ассоциации**

Отношение ассоциации соответствует наличию некоторого отношения между классами. Данное отношение обозначается сплошной линией с дополнительными специальными символами, которые характеризуют отдельные свойства конкретной ассоциации. В качестве дополнительных специальных символов могут использоваться имя ассоциации, а также имена и кратность классов-ролей ассоциации. Имя ассоциации является необязательным элементом ее обозначения. Если оно задано, то записывается с заглавной (большой) буквы рядом с линией соответствующей ассоциации.

Наиболее простой случай данного отношения – бинарная ассоциация. Она связывает в точности два класса. Простым примером отношения бинарной ассоциации является отношение между двумя классами – классом «Компания» и классом «Сотрудник» (см. рисунок 11). Они связаны между собой бинарной ассоциацией *Работа*, имя которой указано на рисунке рядом с линией ассоциации.

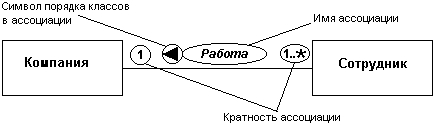


Рисунок 11 – Графическое изображение отношения бинарной ассоциации

Кратность отдельного класса обозначается в виде интервала целых чисел. Так, для рассмотренного примера кратность «1» для класса «Компания» означает, что каждый сотрудник может работать только в одной компании. Кратность «1..\*» для класса «Сотрудник» означает, что в каждой компании могут работать несколько сотрудников, общее число которых заранее неизвестно и ничем не ограничено. Заметим, что вместо кратности «1..\*» записать только символ «\*» нельзя, поскольку последний означает кратность «0..\*». Для данного примера это означало бы, что отдельные компании могут совсем не иметь сотрудников в своем штате.

Как уже упоминалось, отдельный класс ассоциации имеет собственную роль в отношении. Эта роль может быть изображена графически на диаграмме классов. С этой целью в языке UML вводится в рассмотрение специальный элемент – конец ассоциации (Association End), который графически соответствует точке соединения линии ассоциации с отдельным классом. Конец ассоциации является частью ассоциации, но не класса. Каждая ассоциация имеет два или больше концов ассоциации. Наиболее важные свойства ассоциации указываются на диаграмме рядом с этими элементами ассоциации и должны перемешаться вместе с ними. Одним из таких дополнительных обозначений является имя роли отдельного класса, входящего в ассоциацию. Имя роли представляет собой строку текста рядом с концом ассоциации для соответствующего класса. Она указывает специфическую роль, которую играет класс, являющийся концом рассматриваемой ассоциации. Имя роли не является обязательным элементом обозначений и может отсутствовать на диаграмме.

### **Отношение агрегации**

Отношение агрегации имеет место между несколькими классами в том случае, если один из классов представляет собой некоторую сущность, включающую в себя в качестве составных частей другие сущности. Данное отношение имеет фундаментальное значение для описания структуры сложных систем, поскольку применяется для представления системных взаимосвязей типа «часть-целое». Раскрывая внутреннюю структуру системы, отношение агрегации показывает, из каких компонентов состоит система и как они связаны между собой.

Графически отношение агрегации изображается сплошной линией, один из концов которой представляет собой незакрашенный внутри ромб. Этот ромб указывает на тот из классов, который представляет собой «целое». Остальные классы являются его «частями» (см. рисунок 12).



Рисунок 12 – Графическое изображение отношения агрегации

Еще одним примером отношения агрегации может служить деление персонального компьютера на составные части: системный блок, монитор, клавиатуру и мышь. Используя обозначения языка UML, компонентный состав ПК можно представить в виде соответствующей диаграммы классов (см. рисунок 13), которая в данном случае иллюстрирует отношение агрегации.

Примечание.

В связи с рассмотрением данного отношения вполне уместно вспомнить о специальном термине «агрегат», которое служит для обозначения технической системы, состоящей из взаимодействующих составных частей или подсистем.

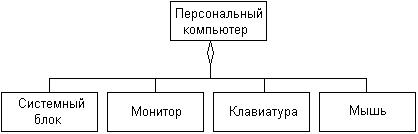


Рисунок 13 – Диаграмма классов для иллюстрации отношения агрегации

### **Отношение композиции**

Отношение композиции, как уже упоминалось ранее, является частным случаем отношения агрегации. Это отношение служит для выделения специальной формы отношения «часть-целое», при которой составляющие части в некотором смысле находятся внутри целого. Специфика взаимосвязи между ними заключается в том, что части не могут выступать в отрыве от целого, т.е. с уничтожением целого уничтожаются и все его составные части.

Пример этого отношения представляет собой живая клетка в биологии. Другой пример – окно интерфейса программы, которое может состоять из строки заголовка, кнопок управления размером, полос прокрутки, главного меню, рабочей области и строки состояния. Подобное окно представляет собой класс, а его компоненты являются как классами, так и атрибутами или свойствами окна. Последнее обстоятельство весьма характерно для отношения композиции, поскольку отражает различные способы представления данного отношения.

Графически отношение композиции изображается сплошной линией, один из концов которой представляет собой закрашенный внутри ромб. Этот ромб указывает на тот из классов, который представляет собой класс-композицию или «целое». Остальные классы являются его «частями (рисунок 14).



Рисунок 14 – Графическое изображение отношения композиции

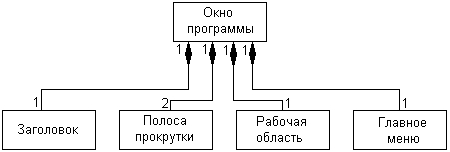


Рисунок 15 – Диаграмма классов для иллюстрации отношения композиции на примере класса окна программы

### **Отношение обобщения**

Отношение обобщения является отношением между более общим элементом (родителем или предком) и более частным или специальным элементом (дочерним или потомком). Данное отношение может использоваться для представления взаимосвязей между пакетами, классами, вариантами использования и другими элементами языка UML.

Применительно к диаграмме классов данное отношение описывает иерархическое строение классов и наследование их свойств и поведения. При этом предполагается, что класс-потомок обладает всеми свойствами и поведением класса-предка, а также имеет свои собственные свойства и поведение, которые отсутствуют у класса-предка. На диаграммах отношение обобщения обозначается сплошной линией с треугольной стрелкой на одном из концов (рисунок 16). Стрелка указывает на более общий класс (класс-предок или суперкласс).



Рисунок 16 – Графическое изображение отношения обобщения

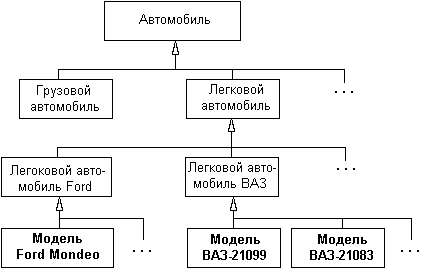


Рисунок 17 – Фрагмент диаграммы классов с отношением обобщения

Рядом со стрелкой обобщения может размещаться строка текста, указывающая на некоторые дополнительные свойства этого отношения. В версии MS UML строка задает стереотип отношения с помощью слов: *extends, inherits, private, protected, subclass, subtype, uses.*

Диаграмма, на которой отображаются классы и отношения между ними называется статической диаграммой классов (или диаграммой классов). В литературе используются также и другие наименования – «информационная модель»[7]. На рисунке 18 приведен фрагмент диаграммы классов, содержащей отношение обобщения и бинарной ассоциации.



Рисунок 18 – Пример диаграммы классов

**Литература**

1. Уоссермен Ф., Нейрокомпьютерная техника, – М., Мир, 1992.

2. Горбань А.Н. Обучение нейронных сетей. – М.: ПараГраф, 1990

3. Горбань А.Н., Россиев Д.А. Нейронные сети на персональном компьютере. – Новосибирск: Наука, 1996

4. Gilev S.E., Gorban A.N., Mirkes E.M. Several methods for accelerating the training process of neural networks in pattern recognition // Adv. Modelling & Analysis, A. AMSE Press. – 1992. – Vol.12, N4. – P.29–53

5. С. Короткий. Нейронные сети: алгоритм обратного распространения.

6. С. Короткий, Нейронные сети: обучение без учителя. Artificial Neural Networks: Concepts and Theory, IEEE Computer Society Press, 1992.

7. Заенцев И.В. Нейронные сети: основные модели. / Учебное пособие к курсу «Нейронные сети» для студентов 5 курса магистратуры к. электроники физического ф-та Воронежского Государственного университета – e-mail: ivz@ivz.vrn.ru

8. Лорьер Ж.Л. Системы искусственного интеллекта. – М.: Мир, 1991. – 568 с.

9. Искусственный интеллект. – В 3-х кн. Кн. 2. Модели и методы: Справочник/ Под ред. Поспелова Д.А. – М.: Радио и связь, 1990. – 304 с.

10. Бек Л. Введение в системное программирование. – М.: Мир, 1988.

11. Шлеер С., Меллор С. Объектно-ориентированный анализ: моделирование мира в состояниях. – К.: Диалектика, 1993. – 240 с.

12. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на С++. – http://www.nexus.odessa.ua/files/books/booch.

13. Аджиев В. MS: корпоративная культура разработки ПО – http:// www.osp.ru

14. Трофимов С.А. Case-технологии. Практическая работа в Rational Rose. – М.: ЗАО «Издательство БИНОМ», 2001.