Рабочие процессы RUP и диаграммы UML

http://www.caseclub.ru/articles/rup.html

Rational Unified Process (RUP) – одна из лучших методологий разработки программного обеспечения, созданная в компании Rational Software. Основываясь на опыте многих успешных программных проектов, Унифицированный процесс позволяет создавать сложные программные системы, основываясь на индустриальных методах разработки. Одним из основных столпов, на которые опирается RUP, является процесс создания моделей при помощи унифицированного языка моделирования (UML). Эта статья о применении диаграмм UML в рабочем процессе разработки программных систем по методологии Rational Software.

Ни для кого не секрет, что создание программного обеспечения – это сложный процесс, который, с одной стороны, имеет много общего с творчеством, а с другой, – хотя и высокодоходный, но и высокозатратный бизнес. Жестокая конкуренция на рынке вынуждает разработчиков к поиску более эффективных методов работы. Путей создания программных систем в еще более короткие сроки, с меньшими затратами и лучшим качеством. Сложность программ постоянно увеличивается. Еще недавно программные продукты могли быть созданы в обозримые сроки одиночками или, например, в IT-отделе автоматизируемого предприятия.

Сейчас же одиночкам, создающим программы «на коленке» остается область небольших утилит и различных модулей расширения «тяжелых» программных продуктов. Будущее за индустриальным подходом к созданию ПО. В 1913 году Генри Форд запустил первый автомобильный конвейер, а в 90-х аналогичный конвейер стал применяться в сфере IT-технологий. Командная разработка требует совсем другого подхода и другой методологии, которая рано или поздно должна была быть создана.

Корпорация Rational Software (http://www.rational.com) выпустила на рынок структурированную базу знаний под названием Rational Unified Process (RUP)[1,2], которая представляет собой набор исчерпывающих рекомендаций для создания практически любых программных продуктов. Вобрав в себя опыт лучших разработок, RUP подробно рассказывает когда, кто и что должен делать в проекте, чтобы в результате получить программную систему с установленные сроки, с определенной функциональностью и в рамках отведенного бюджета.

Унифицированный процесс можно представить как сумму различных видов деятельности компании-разработчика, необходимых для перевода требований заказчика в программную систему. Систему, которая давала бы «значимый результат»[2] пользователям и выполняла бы именно то, что они от системы ожидают. Поэтому процесс управляется вариантами использования (Use Case) системы, или иначе – прецедентами.

Для реализации требований заказчика в установленные сроки, Унифицированный процесс разделяется на фазы, которые состоят из итераций, поэтому процесс еще называют итеративным и инкрементным. Каждая итерация проходит цикл основных работ и подводит разработчиков к конечной цели: созданию программной системы. В ходе итераций создаются промежуточные артефакты, которые требуются для успешного завершения проекта и вариант программной системы, который реализует некоторый набор функций, увеличивающийся от итерации к итерации. Фазы и основные потоки работ процесса показаны на рис. 1, там же даны примерные трудозатраты работ по фазам.

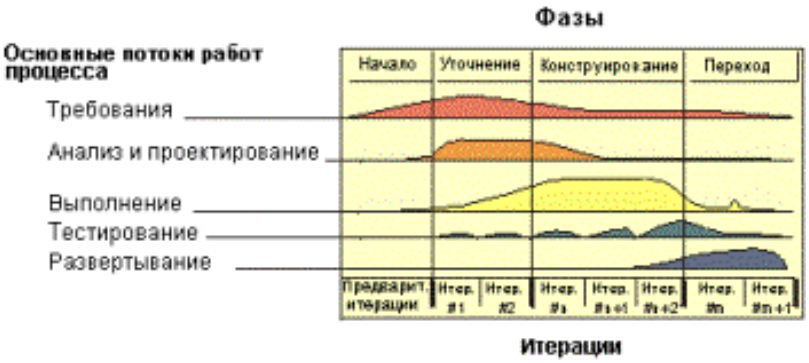
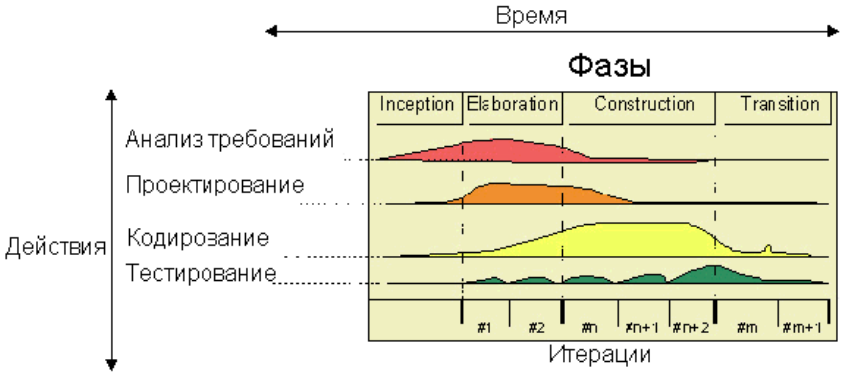


рис. 1 Фазы и потоки работ RUP

Унифицированный процесс разработки



Нужно отметить, что на рис. 1 показаны только основные работы Унифицированного процесса. Например, работы по управлению деятельностью здесь не показаны, чтобы не загромождать диаграмму.

Вся разработка ПО рассматривается в RUP как процесс создания артефактов. Любой результат работы проекта, будь то исходные тексты, объектные модули, документы, передаваемые пользователю, модели – это подклассы всех артефактов проекта. Каждый член проектной группы создает свои артефакты и несет за них ответственность. Программист создает программу, руководитель — проектный план, а аналитик — модели системы. RUP позволяет определить когда, кому и какой артефакт необходимо создать, доработать или использовать.

Одним из интереснейших классов артефактов проекта являются модели, которые позволяют разработчикам определять, визуализировать, конструировать и документировать артефакты программных систем. Каждая модель является самодостаточным взглядом на разрабатываемую систему и предназначена как для очерчивания проблем, так и для предложения решения. Самодостаточность моделей

означает, что аналитик или разработчик может из конкретной модели почерпнуть всю необходимую ему информацию, не обращаясь к другим источникам.

Модели позволяют рассмотреть будущую систему, ее объекты и их взаимодействие еще до вкладывания значительных средств в разработку, позволяют увидеть ее глазами будущих пользователей снаружи и разработчиков изнутри еще до создания первой строки исходного кода. Большинство моделей представляются UML диаграммами, подробнее об UML можно прочитать, например в [3].

Унифицированный язык моделирования (Unified Modeling Language) появился в конце 80-х в начале 90-х годов в основном благодаря усилиям «трех друзей» Гради Буча, Джима Рамбо и Ивара Якобсона. В настоящее время консорциум OMG принял этот язык как стандартный язык моделирования, который предоставляет разработчикам четкую нотацию, позволяющую отображать модели общепринятыми и понятными каждому члену проекта графическими элементами.

Однако не следует забывать, что язык моделирования дает только нотацию – инструмент описания и моделирования системы, а унифицированный процесс определяет методику использования этого инструмента, как впрочем, и других инструментов поддержки методологии от компании Rational. UML можно использовать и без конкретной методологии, поскольку он не зависит от процесса и какой бы вариант процесса не был бы применен, вы можете использовать диаграммы для документирования принятых в ходе разработки решений и отображения создаваемых моделей.

Программная система создается для решения определенных проблем пользователя, а не для опробования новых технологий программистами и получения опыта руководителем проекта. По большому счету, пользователю не важно используете ли вы в процессе разработки объектно-ориентированный подход, UML, RUP или создаете систему по методу XP (экстремального программирования) [4]. Применение тех или иных методик, технологий, создание оптимальной внутренней структуры проекта остается на совести разработчиков, которые принимают решения исходя из предыдущего опыта и собственных предпочтений. Однако пользователь не простит вам игнорирование его требований. Будь программная система хоть десять раз разработана при помощи суперсовременных методов и технологий, если пользователь не получит от нее то, что называется «значимым результатом», ваш программный проект с треском провалится.

Из этого следует, что бездумное применение UML, просто потому что это модно, не только не приведет разработку к успеху, но и может вызвать недовольство сотрудников, которым необходимо изучать большое количество дополнительной литературы и руководителей проекта, когда окажется, что трудозатраты на проекте возрастают, а отдача не повышается. Нужно четко представлять себе, что вы хотите получить от внедрения этой технологии и следовать этой цели. Применение UML экономит ресурсы разработки, поскольку позволяет получить представление о системе быстрее, чем при создании макетов и прототипов, затратив на это несравнимо меньше ресурсов.

Диаграммы позволяет легче общаться членам проекта между собой, и, что особенно ценно, вовлекают в процесс конечных пользователей системы. Моделирование позволяет уменьшить риски проекта, поскольку программистам всегда легче делать то, что ясно и понятно, чем идти к неопределенному результату. Создание диаграмм

аналогично созданию проекта в строительстве – можно обойтись и без него, например, при строительстве сарая на дачном участке, однако, чем больше здание (в нашем случае программный продукт), тем труднее это делать и неопределеннее конечный результат.

Однажды я проводил семинар по RUP в компании-разработчике программного обеспечения, которая уже десять лет довольно успешно работает на рынке, но не использует в своем рабочем процессе моделирования вовсе, а основывается на прототипах. В зале собралось около двадцати молодых и умудренных опытом программистов, которые внимательно прослушали все, что я им рассказал о RUP и UML. И вот один из них, посмотрев на доску, испещренную примерами диаграмм, заметил: «Это все интересно и, наверное, хорошо подходит для других проектов», – сказал он, «но мы работаем довольно давно без всего этого, раз мы до сих пор обходились без UML, он, наверное, нам это просто не нужен».

Этот вопрос заставил меня задуматься о том, что изменение бизнес-процессов, которое неизбежно должно произойти в компании-разработчике программного обеспечения при внедрении RUP и UML может проходить также тяжело, как внедрение информационной системы на промышленном предприятии. Любое внедрение – это ломка стереотипов. Поскольку квалификация сотрудников в компании-разработчике ПО довольно высока, то и отказаться от своих взглядов таким людям труднее, чем «простым смертным», а возникающие трудности и неприятие можно сравнить с переходом от процедурного к объектно-ориентированному мышлению.

1.Определение требований

Унифицированный процесс – это процесс, управляемый прецедентами, которые отражают сценарии взаимодействия пользователей. Фактически, это взгляд пользователей на программную систему снаружи. Таким образом, одним из важнейших этапов разработки, согласно RUP, будет этап определения требований, который заключается в сборе всех возможных пожеланий к работе системы, которые только могут прийти в голову пользователям и аналитикам. Позднее эти данные должны будут систематизированы и структурированы, но на данном этапе в ходе интервью с пользователями и изучения документов, аналитики должны собрать как можно больше требований к будущей системе, что не так просто, как кажется на первый взгляд. Пользователи часто сами не представляют, что они должны получить в конечном итоге. Для облегчения этого процесса аналитики используют диаграммы прецедентов (рис. 2)

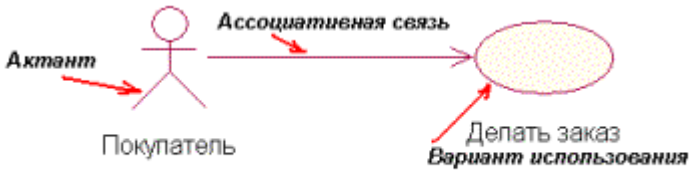


рис 2. Пример диаграммы Прецедентов

Диаграмма представляет собой отражение действующих лиц (актантов), которые взаимодействуют с системой, и реакцию программных объектов на их действия. Актантами могут быть как пользователи, так и внешние агенты, которым

необходимо передать или получить информацию. Значок варианта использования отражает реакцию системы на внешнее воздействие и показывает, что должно быть сделано для актанта.

Для детализации конкретного прецедента используется диаграмма Активности (Activity Diagram), пример которой дан на рис 3.

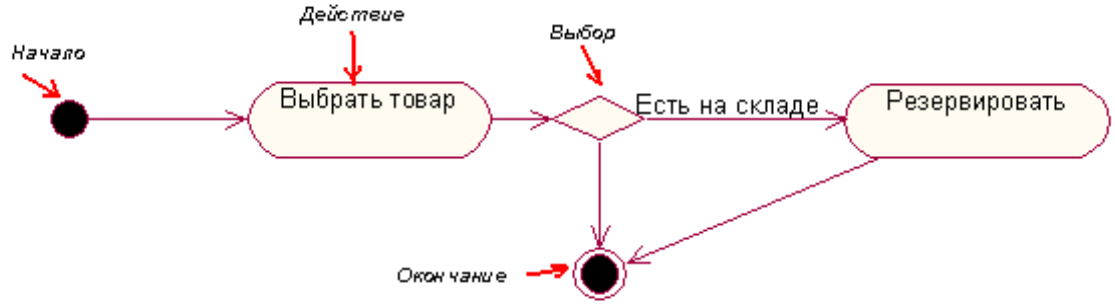


рис. 3 Пример диаграммы Активности

Простота диаграммы прецедентов позволяет аналитикам легко общаться с заказчиками в процессе определения требований, выявлять ограничения, налагаемые на систему и на выполнение отдельных требований, такие, например, как время реакции системы, которые в дальнейшем попадают в раздел нефункциональных требований.

Также диаграмма прецедентов может использоваться для создания сценариев тестирования, поскольку все взаимодействие пользователей и системы уже определено.

Для того чтобы верно определить требования, разработчики должны понимать контекст (часть предметной области) в котором будет работать будущая система. Для этого создаются модель предметной области и бизнес-модель, что является различными подходами к одному и тому же вопросу. Часто создается что-то одно: модель предметной области или бизнес-модель.

Отличия этих моделей в том, что модель предметной области описывает важные понятия, с которыми будет работать система и связи их между собой. Тогда как бизнес-модель описывает бизнес-процессы (существующие или будущие), которые должна поддерживать система. Поэтому кроме определения бизнес-объектов, вовлеченных в процесс, эта модель определяет работников, их обязанности и действия, которые они должны выполнять.

Для создания модели предметной области используется обычная диаграмма классов (рис 6), однако для создания бизнес-модели ее уже явно недостаточно. В этом случае применяется диаграмма прецедентов с использованием дополнительных значков, которые отражающие сущность бизнес-процессов – это бизнес-актант, бизнес-прецедент, бизнес-сущность и бизнес-управление. Эта модель намного ближе к следующей модели, создаваемой в процессе разработки – модели анализа.

2.Анализ

После определения требований и контекста, в котором будет работать система, наступает черед анализа полученных данных. В процессе анализа создается аналитическая модель, которая подводит разработчиков к архитектуре будущей системы. Аналитическая модель – это взгляд на систему изнутри, в отличие от модели прецедентов, которая показывает, как система будет выглядеть снаружи.

Эта модель позволяет понять, как система должна быть спроектирована, какие в ней должны быть классы и как они должны взаимодействовать между собой. Основное ее назначение - определить направление реализации функциональности, выявленной на этапе сбора требований и сделать набросок архитектуры системы.

В отличие от создаваемой в дальнейшем модели проектирования, модель анализа является в большей степени концептуальной моделью и только приближает разработчиков к классам реализации. Эта модель не должна иметь возможных противоречий, которые могут встретиться в модели прецедентов.

Для отображения модели анализа при помощи UML используется диаграмма классов со стереотипами (образцами поведения) «граничный класс», «сущность», «управление», а для детализации используются диаграммы сотрудничества (Collaboration) (рис 4). Стереотип «граничный класс» отображает класс, который взаимодействует с внешними актантами, «сущность» – отображает классы, которые являются хранилищами данных, а «управление» – классы, управляющие запросами к сущностям.

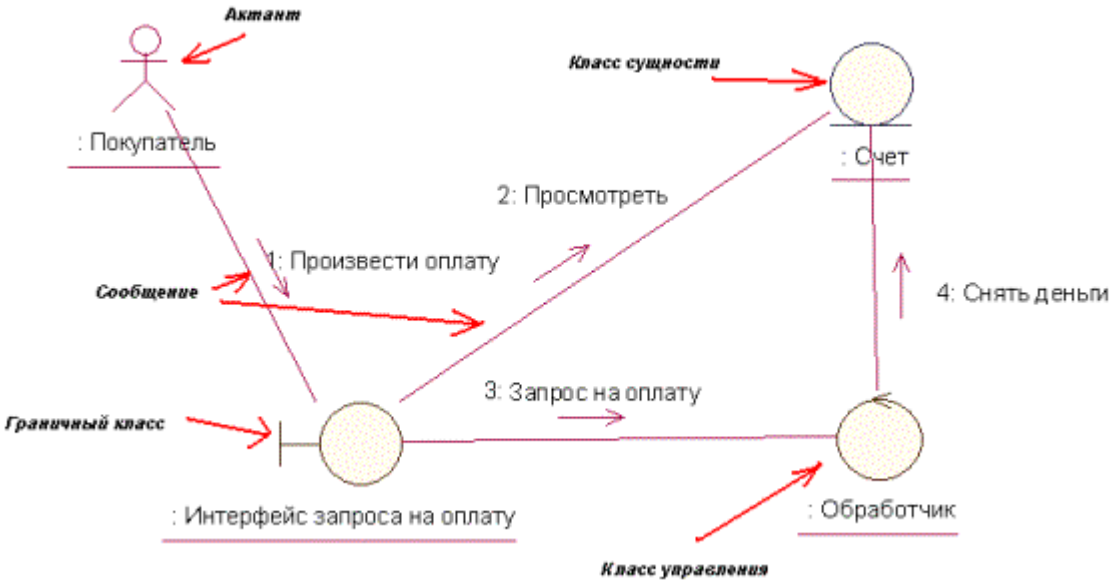


рис 4. Пример диаграммы сотрудничества

Нумерация сообщений показывает их порядок, однако назначение диаграммы не в том, чтобы рассмотреть порядок обмена сообщениями, а в том, чтобы наглядно показать связи классов друг с другом.

Если акцентировать внимание на порядке взаимодействия, то другим его представлением будет диаграмма последовательности (Sequence), показанная на рис 5. Эта диаграмма позволяет взглянуть на обмен сообщениями во времени, наглядно отобразить последовательность процесса. При использовании такого инструмента для создания моделей как Rational Rose, эти два вида диаграмм могут быть созданы друг из друга автоматически (подробнее о Rational Rose можно прочитать, например, в [5]).

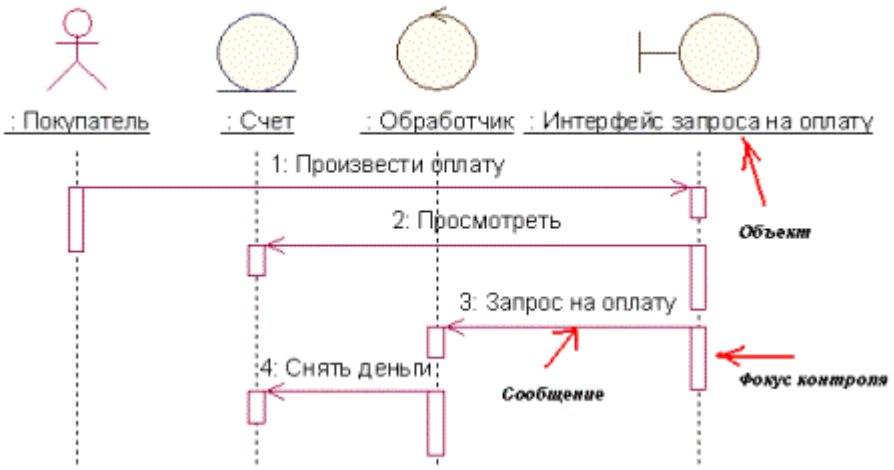


Рис. 5 Пример диаграммы последовательности действий

Решение о том, какую из двух диаграмм нужно создавать первой, зависит от предпочтений конкретного разработчика. Поскольку эти диаграммы являются отображением одного и того же процесса, то и та и другая позволяют отразить взаимодействие между объектами.

3.Проектирование

Следующим этапом в процессе создания системы будет проектирование, в ходе которого на основании моделей, созданных ранее, создается модель проектирования. Эта модель отражает физическую реализации системы и описывает создаваемый продукт на уровне классов и компонентов. В отличие от модели анализа, модель проектирования имеет явно выраженную зависимость от условий реализации, применяемых языков программирования и компонентов. Для максимально точного понимания архитектуры системы, эта модель должна быть максимально формализована, и поддерживаться в актуальном состоянии на протяжении всего жизненного цикла разработки системы.

Для создания модели проектирования используются целый набор UML диаграмм: диаграммы классов (рис. 5), диаграммы кооперации, диаграммы взаимодействия, диаграммы активности.

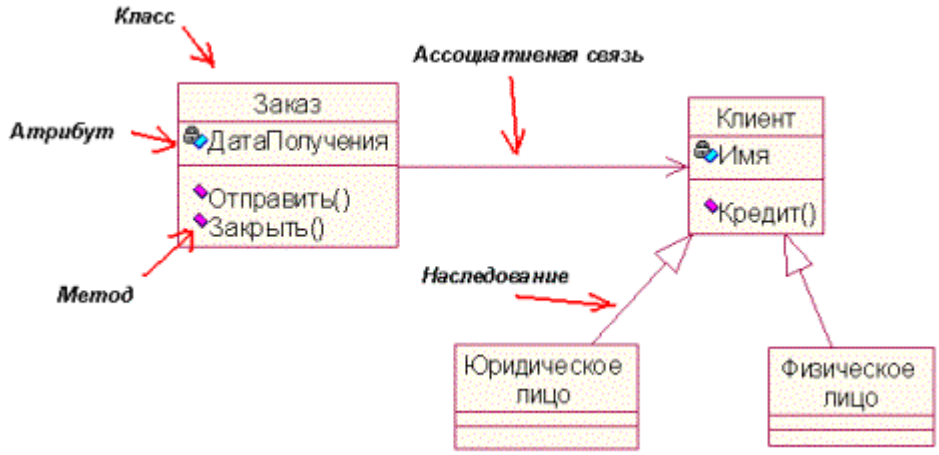


рис 6. Пример диаграммы классов

Дополнительно в этом рабочем процессе может создаваться модель развертывания, которая реализуется на основе диаграммы развертывания (Deployment Diagram). Это самый простой тип диаграмм, предназначенный для моделирования распределения устройств в сети. Для отображения используется всего два варианта значков процессор и устройство вместе со связями между ними.

4.Реализация

Основная задача процесса реализации – создание системы в виде компонентов – исходных текстов программ, сценариев, двоичных файлов, исполняемых модулей и т.д. На этом этапе создается модель реализации, которая описывает то, как реализуются элементы модели проектирования, какие классы будут включены в конкретные компоненты. Данная модель описывает способ организации этих компонентов в соответствии с механизмами структурирования и разбиения на модули, принятыми в выбранной среде программирования и представляется диаграммой компонентов (рис. 7).

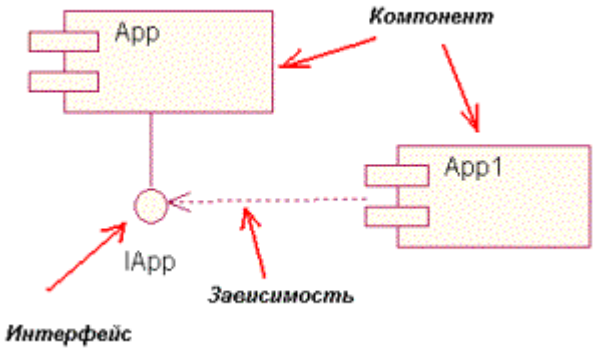


рис. 7 Пример диаграммы компонентов

5.Тестирование

В процессе тестирования проверяются результаты реализации. Для данного процесса создается модель тестирования, которая состоит из тестовых примеров, процедур тестирования, тестовых компонентов, однако не имеет отображения на UML диаграммы, поэтому не будем на ней останавливаться.

6.Заключение

Здесь были рассмотрены только основные процессы методологии Rational. RUP довольно обширен, и содержит рекомендации по ведению различных программных проектов, от создания программ группой разработчиков в несколько человек, до распределенных программных проектов, объединяющих тысячи человек на разных континентах. Однако, несмотря их колоссальную разницу, методы применения моделей, создаваемых при помощи UML будут одни и те же. Диаграммы UML, создаваемые на различных этапах разработки, неотделимы от остальных артефактов программного проекта и часто являются связующим звеном между отдельными процессами RUP.

Решение о применении конкретных диаграмм зависит от поставленного в компании процесса разработки, который, хотя и называется унифицированным, однако не есть нечто застывшее. Компания Rational не только предлагает его улучшать и дорабатывать, но и предоставляет специальные средства внесения изменений в базу данных RUP.

Но в любом случае, применение UML вместе с унифицированным процессом позволит получить предсказуемый результат, уложиться в отведенный бюджет, повысить отдачу от участников проекта и качество создаваемого программного продукта.

[1] Крачтен.Ф. Введение в Rational Unified Process. Изд. 2-е.- М.: Издательский дом "Вильямс", 2002. - 240 с.: ил.

[2] Якобсон А., Буч Г., Рамбо Дж. Унифицированный процесс разработки программного обеспечения - Спб.: Питер, 2002. - 496 с.: ил.

[3] Фаулер М., Скотт К. UML в кратком изложении. Применение стандартного языка объектного моделирования: Пер. с англ. – М.:Мир, 1999. – 191 с., ил.

[4] Бек. Е. Экстремальное программирование. – Спб.: Питер, 2002. – 224 с.: ил.

[5] Трофимов С. CASE-технологии: Практическая работа в Rational Rose. Изд. 2-е.- М.: Бином-Пресс, 2002 г. - 288 с.