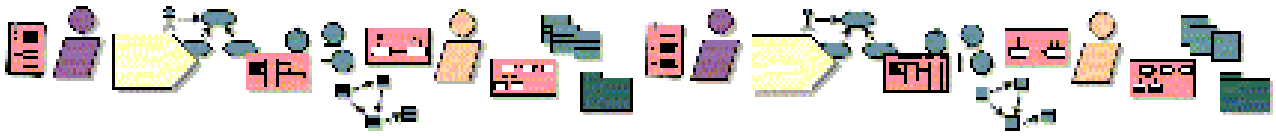


Особенности Rational Unified Process

Мы уже обсуждали кратко особенности технологии Rational Unified Process (см. стр. 2-1 – 2-3). Теперь рассмотрим их более подробно.

Рисунок внизу иллюстрирует основные идеи Rational Unified Process. Описание каждой из них поможет Вам понять основание для соответствующей концепции Rational Unified Process и способы ее реализации. Это, в свою очередь, должно помочь Вам впоследствии более эффективно использовать Rational Unified Process.

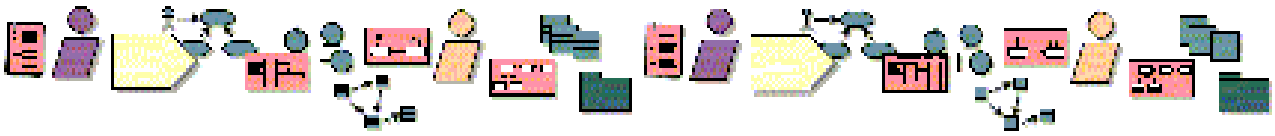
Каждый луч этой звезды представляет основополагающую идею Rational Unified Process.

Примечание: Не забывайте об интерактивных возможностях наших иллюстраций!

Визуальное моделирование

Множество проектов используют сегодня объектно-ориентированные языки программирования для получения перенастраиваемых, допускающих изменения и при этом устойчивых систем. Но чтобы получить эти преимущества, гораздо важнее использовать объектную технологию проектирования. Rational Unified Process производит объектно-ориентированную модель проекта, на которой базируется вся дальнейшая работа.

Объектно-ориентированная модель стремится к отражению мира, который мы наблюдаем в действительности. Поэтому объекты часто соответствуют вполне реальным вещам, которые должна обработать система. Например, объектами могут быть счет в банковской системе или



служащий в системе управления персоналом.

Модель, правильно разработанная с использованием объектной технологии:

- проста для понимания. Она ясно отражает действительность.
- проста для изменения. Изменение в конкретном явлении касается только объекта, представляющего это явление.

В этом разделе Вы познакомитесь с понятиями субъектов и прецедентов. Мы обсудим также другие объектно-ориентированные понятия и общие идеи моделирования. Более детальные определения Вы найдете, если у Вас хватит терпения, в главах, посвященных основным потокам работ.

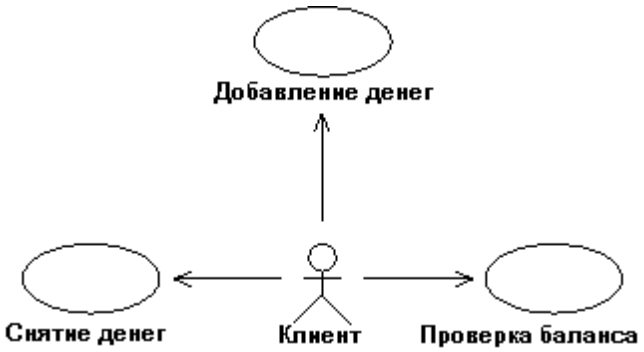
Субъекты

Для полного понимания назначения системы Вы должны знать**, для кого** предназначена эта система и **кто** будет ее использовать. В Rational Unified Process различные типы пользователей представлены **субъектами**. Субъектом представляется также и любая другая система, которая взаимодействует с нашей системой; таким образом, **субъекты определяют границы системы**.

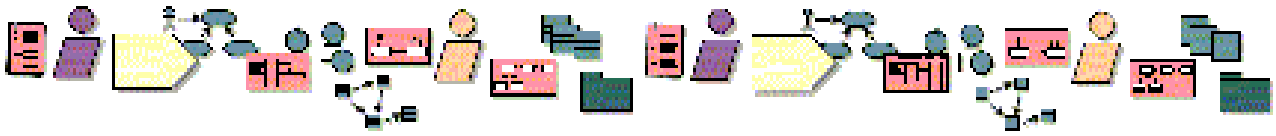
Прецеденты

Функциональные возможности системы определяются **прецедентами**, каждый из которых представляет определенный способ использования системы. Описание прецедента определяет то, что произойдет в системе, когда будет выполнен прецедент. Таким образом, каждый прецедент соответствует последовательности действий, выполняемых системой, которая выдает наблюдаемый результат, имеющий ценность для некоторого субъекта.

Действие – это атомарный набор операций, который выполняется полностью или частично.



Подойдя к банкомату, клиент может, например, снять деньги со счета, добавить деньги на счет или проверить состояние счета. Эти действия соответствуют потокам событий в системе, которые могут быть представлены прецедентами.



Объекты

Объект – это абстракция чего-либо в прикладной области или в выполняемой системе. Вы помните, что объектами могут быть, например, счет в банковской системе или служащий в системе управления персоналом.

Объект инкапсулирует данные и поведение. Данные объекта представляются **атрибутами**, а его поведение - **операциями**.

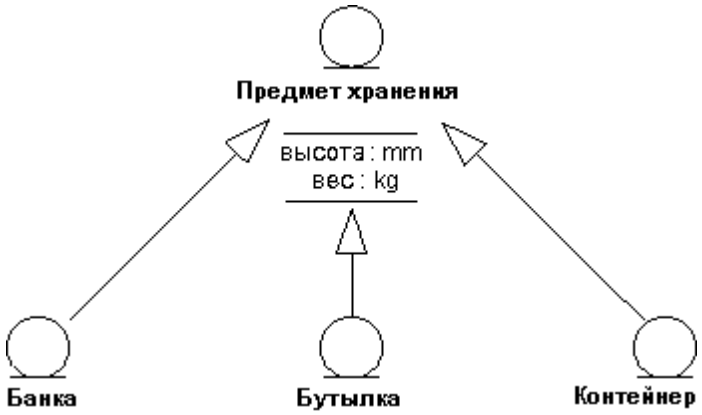
Объект определяется в **классе**.

Классы

Класс определяет шаблон структуры и поведение всех его объектов. Объекты, созданные в классе, называются также **экземплярами** класса.

Обобщения

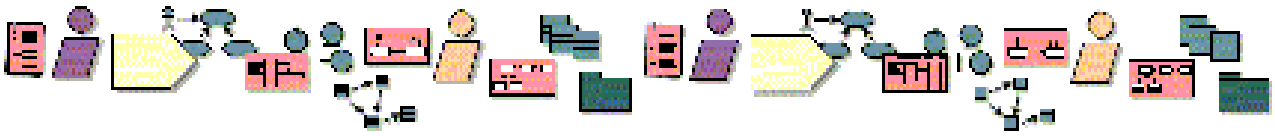
Обобщения показывают, как один класс наследуется от другого. Наследованный класс называется **потомком**. Класс, от которого происходит наследование, называется **предком**. Наследование означает, что определение предка - включая любые свойства типа атрибутов, связей или операций его объектов - является правильным и для объектов его потомка. Обобщение выводится от класса-потомка к его классу-предку.



Банки, бутылки и контейнеры имеют общие свойства высоты и веса. Для работника склада каждое из понятий является уточнением обобщённого понятия «предмет хранения».

Модели

Разработка моделей широко принята во всех технических дисциплинах в значительной степени потому, что она позволяет представить себе предмет моделирования прежде, чем он будет создан фактически. Моделирование позволяет в самом начале увидеть проблемы, установить и устранить которые позже было бы чрезвычайно дорого или вообще невозможно.



В Rational Unified Process используется несколько моделей системы. Это связано с тем, что в процессе жизненного цикла системы модели используются разными специалистами, каждый из которых имеет свои интересы. Когда Вы обсуждаете систему и требования к ней, например, с конечным пользователем или заказчиком, модель должна описывать **то, что** система должна делать, и абстрагироваться от подробностей выполнения. Позже, при проектировании, модель должна фокусироваться на потребностях проектировщиков, которые должны иметь возможность обсуждать проблемы декомпозиции, абстракции и иерархии на уровне выше уровня языка программирования.

Выполнение и тестирование также требуют различных характеристик моделей, приемлемых для разработчиков и тестировщиков. Таким образом, двигаясь через жизненный цикл приложения, Вы должны будете разработать несколько моделей, которые описывают различные аспекты разрабатываемой системы.

Инструментальная поддержка

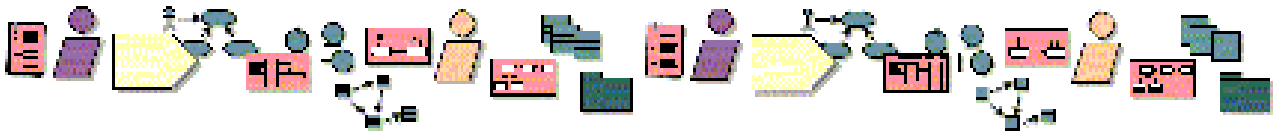
Раньше уже говорилось о том, что модели в Rational Unified Process используют UML как общую систему обозначений. Но при разработке больших систем этого мало. Нужно иметь инструмент, который поддерживал бы не только рисование составляющих модели диаграмм, но и контролировал бы их непротиворечивость и предоставлял возможность прямой и обратной разработки. Это означает, что Вы должны иметь возможность выполнять прямое проектирование и перепроектирование кода без того, чтобы отменять изменения, которые были сделаны в моделях или в коде, начиная с предыдущей генерации.

Rational Unified Process в рамках Rational Suite интегрирован с Rational Rose, который представляет собой именно такой инструмент моделирования.

Управление прецедентами

Положа руку на сердце, скажу - меня поразила бессмысленность использования прецедентов, когда я услышал о них впервые. Тем более, что прецеденты не являются частью «традиционного» объектного программирования. Как же я был не прав!

В традиционной объектно-ориентированной модели системы часто бывает трудно указать, как система будет делать то, что она должна делать. Эта трудность возникает из-за отсутствия «красной нити», объединяющей систему при выполнении отдельных задач. В Rational Unified Process такой «красной нитью» являются **прецеденты**, которые определяют поведение системы. Другие объектно-ориентированные методы также используют прецеденты, применяя для них разные названия.



Что такое прецедент?

Понятие прецедента было введено на стр. 4-2 при обсуждении основных концепций моделирования. Приведем определения:

Экземпляр прецедента – это последовательность действий, выполняемых системой; эта последовательность имеет наблюдаемый результат, представляющий ценность для конкретного субъекта.

Прецедент – это набор экземпляров прецедента.

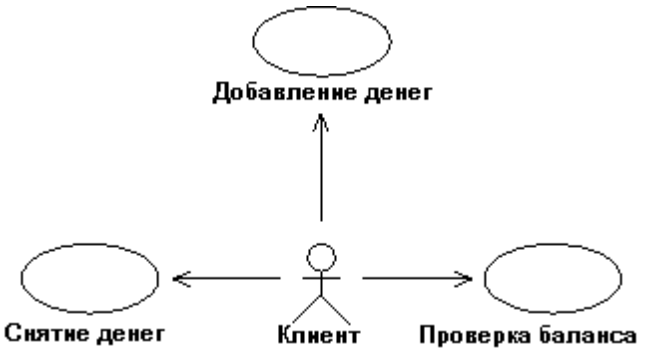
В этом определении есть несколько ключевых слов:

- **Экземпляр прецедента**. Последовательность, упомянутая в определении, действительно является конкретным потоком событий в системе, или экземпляром. Возможно множество потоков событий, и многие из них могут быть очень похожи. Чтобы сделать модель прецедентов понятной, можно сгруппировать подобные потоки событий в один прецедент. Идентификация и описание прецедента в действительности означает идентификацию и описание группы связанных потоков событий.
- **Система выполняет**. Это означает, что система обеспечивает прецедент. Субъект связывается с экземпляром прецедента системы.
- **Наблюдаемый результат, представляющий ценность.** Вы можете оценивать успешность выполнения прецедента. Прецедент следует производить, если в результате субъект сможет решить задачу, которая имеет реальный результат. Это очень важно для определения правильного уровня и глубины детализации прецедента. Правильным является стремление производить не слишком маленькие прецеденты. В некоторых обстоятельствах Вы можете использовать прецедент как элемент планирования.
- **Действия.** Действие – это вычислительная или алгоритмическая процедура; оно выполняется, когда субъект дает системе соответствующий сигнал или когда происходит временное событие. Действие может подразумевать передачу сигнала вызывавшему субъекту или другим субъектам. Действие атомарно; это означает, что оно или выполняется полностью, или не выполняется вовсе.
- **Конкретный субъект**. Субъект является ключевой фигурой при поиске правильного прецедента, особенно потому, что субъект помогает Вам избегать прецедентов, которые слишком велики. Например, рассмотрите инструмент визуального моделирования. В этом приложении имеются два реальных субъекта: разработчик - кто-то, кто разрабатывает системы, используя инструмент для поддержки, и администратор системы - кто-то, кто управляет этим инструментом. Каждый из этих субъектов имеет свои собственные запросы к системе, и ему будут нужны свои наборы прецедентов.

Функциональные возможности системы определяются набором прецедентов, каждый из которых представляет некоторый поток событий. Описание прецедента определяет то, что произойдет в

системе, когда прецедент будет выполнен.

Каждый прецедент имеет собственную задачу, которую он должен выполнить. Набор прецедентов устанавливает все возможные пути использования системы. Вы можете понять задачу прецедента, просто посмотрев на его название.

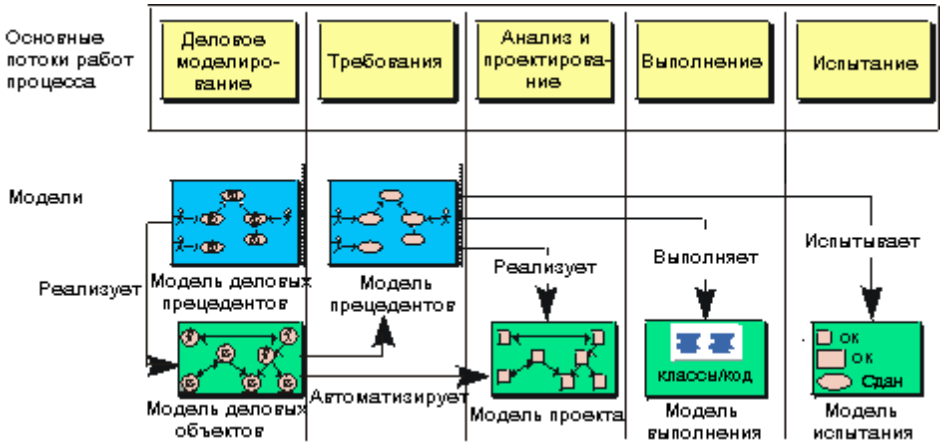


Вам уже знаком этот рисунок. Здесь указаны все прецеденты, которые выполняет банкомат по требованию клиента.

Использование прецедентов при разработке

Итак, в Rational Unified Process определенный для системы прецедент является основанием для всего процесса разработки.

Прецеденты играют роль в каждом из пяти основных потоков работ процесса: Деловое моделирование, Требования, Анализ и проектирование, Выполнение и Испытание.

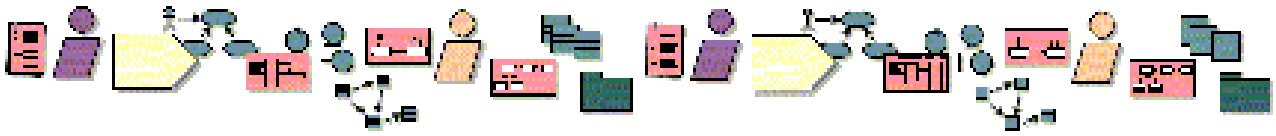


«Поток» прецедентов через различные модели

- **Деловые прецеденты определяются в ходе делового моделирования.**

  В управляемом прецедентами проекте Вы разрабатываете два представления деловой сферы.

  Сам деловой прецедент отражает внешнее представление деловой сферы, которое определяет,



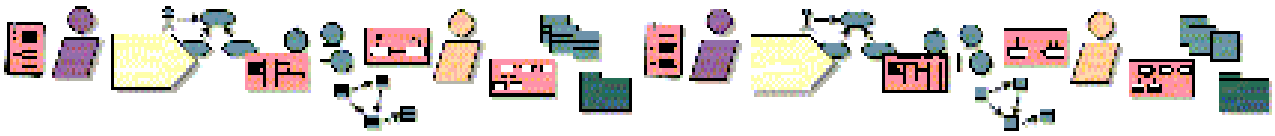
что существенно исполнить в деловой сфере, чтобы обеспечить субъекту нужные результаты. Оно определяет также, **какое** взаимодействие должны иметь деловая сфера и субъекты при выполнении делового прецедента. Такое представление должно быть разработано, когда Вы договариваетесь о том, что должно быть сделано в каждом деловом прецеденте. Совокупность деловых прецедентов **устанавливает границы системы**.

Реализация делового прецедента, с другой стороны, дает внутреннее представление делового прецедента, которое определяет, **как** должна быть организована работа, чтобы достичь нужных результатов. Реализация охватывает деловых работников и деловые сущности, которые участвуют в выполнение делового прецедента, и связи между ними, необходимые для выполнения задания. Такие представления необходимо разработать, чтобы решить, как должна быть организована работа в каждом деловом прецеденте для достижения нужных результатов.

- **Модель прецедентов** создается в потоке работ Требования. В этом потоке работ Вы нуждаетесь в прецеденте для моделирования того, что должна делать система с точки зрения пользователей. Таким образом, прецедент представляет важную фундаментальную концепцию, которая должна быть приемлема и для заказчика и для разработчиков системы.
- В потоке работ Анализ и проектирование прецеденты реализуются в модели проекта. В этой модели Вы создаете **реализации прецедента**, которые в терминах взаимодействующих объектов описывают, как прецедент выполняется. В терминах объектов проекта эта модель описывает различные части реализуемой системы и то, как должны взаимодействовать эти части, чтобы выполнить прецеденты.
- В потоке работ Выполнение **модель проекта** реализует технические требования. Поскольку прецеденты являются фундаментом модели проекта, они реализуются в терминах классов проекта.
- В потоке работ Испытания прецеденты составляют основу для идентификации **контрольных примеров** и **методов испытаний**. То есть система проверяется при выполнении каждого прецедента.

Прецеденты исполняют также и другие роли:

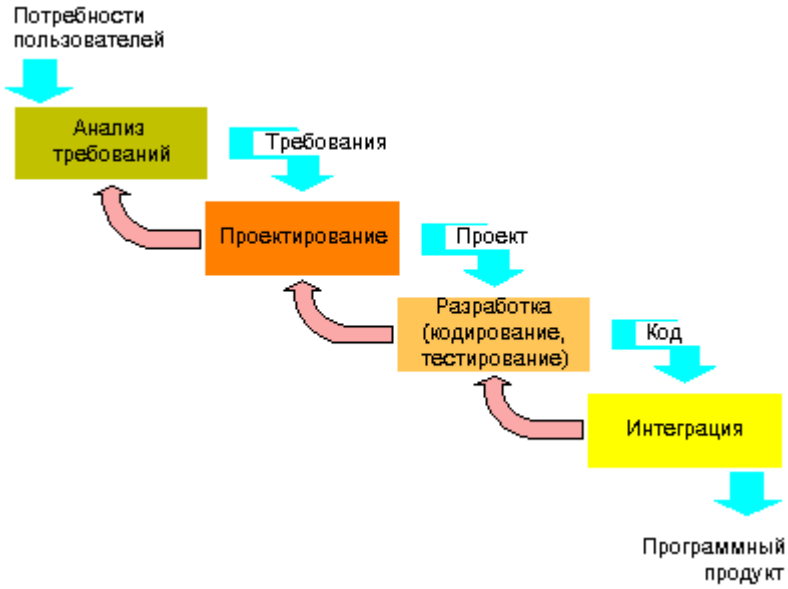
- Они используются для планирования итерационной разработки.
- Они составляют основу при написании руководств пользователей.
- Они могут использоваться как единицы упорядочения. Например, заказчик может конфигурировать систему с индивидуальным набором прецедентов.



Итеративная разработка

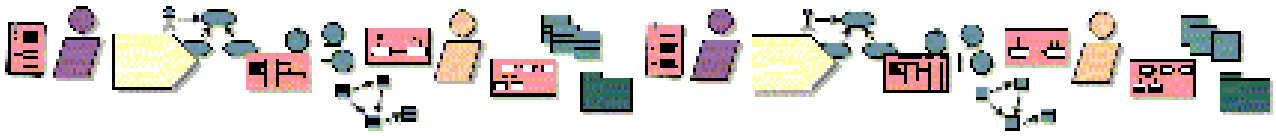
Процесс «водопада»

Сегодня стало модным винить во всех проблемах и неудачах программных проектов последовательный процесс разработки (или «водопад»). Это может показаться довольно странным. Ведь, на первый взгляд, этот подход кажется очень разумным! Вот шаги, которым Вы должны следовать:



В процессе водопада разработка ведется сверху вниз. В случае неудачи возможно возвращение к предыдущему этапу.

1. Исследуйте проблему, которая должна быть решена, все требования и их связи. Зафиксируйте их в виде задания на разработку и заставьте все заинтересованные стороны согласиться, что это именно то, что они хотят получить.
2. Разработайте проектные решения, удовлетворяющие всем требованиям и связям. Тщательно исследуйте эту конструкцию и удостоверьтесь, что все заинтересованные стороны согласны с правильностью Ваших решений.
3. Осуществите проектные решения, используя лучшие методы кодирования.
4. Убедитесь, что все заявленные требования выполнены и ваша работа удовлетворяет заказчика.
5. Передайте заказчику готовый продукт. Проблема решена!



Именно так были построены небоскребы и космические корабли. Этот способ очень хорош, но только потому, что прикладная область давно известна: за плечами инженеров сотни лет проектирования и конструирования.

Специалисты по программному обеспечению не имели такой возможности. Не имея своего, они не могли не воспользоваться положительным опытом специалистов из смежных областей.

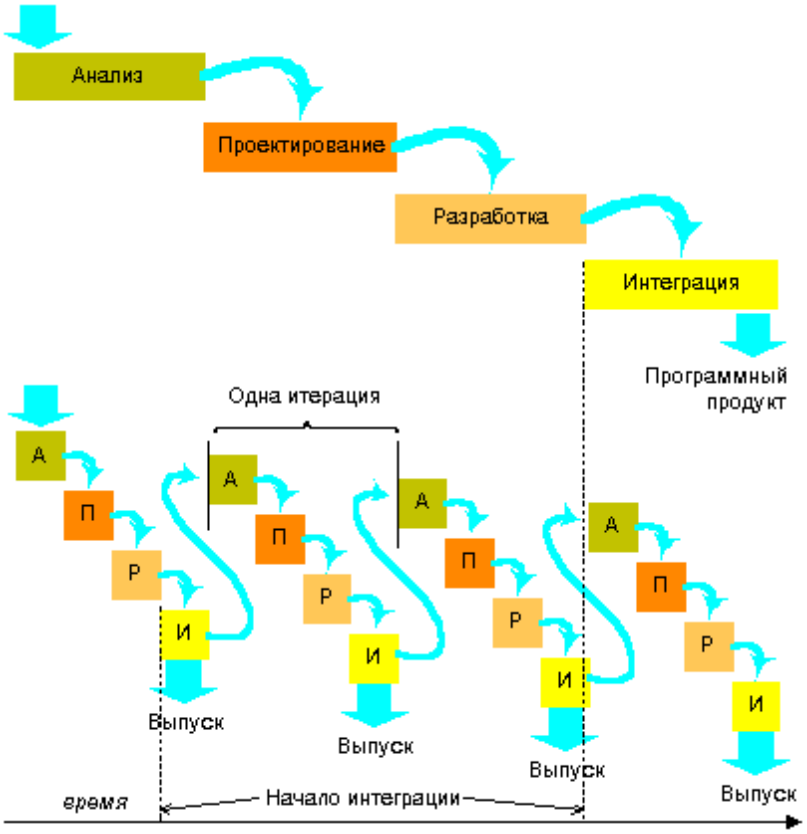
Несколько лет назад многие из нас и представить себе не могли, что процесс «водопада» не является единственным разумным подходом. А если бы и смогли, это не изменило бы ситуацию: метод «водопада» заложен во все ГОСТы и методики проектирования программных систем.

Если процесс «водопада» идеален, тогда почему большинство проектов, особенно проектов крупных систем, заканчиваются неудачей? Тому есть несколько причин.

- Контекст программирования существенно отличается от такового других технических дисциплин.
- Мы не сумели учесть некоторые факторы, связанные с человеком.
- Мы пытаемся использовать подход, четко работающий в определенных обстоятельствах, вне этих обстоятельств.
- Исторически мы все еще находимся в периоде исследований способов разработки программного обеспечения. Мы не имеем сотен лет опыта экспериментов и опыта ошибок, который превратил создание космического корабля в механический процесс. И в этом состоит главная причина неудач.

Преимущества итераций

Определение итерации, как ее представляет Rational Unified Process, приводилось раньше, на стр. 3-8. Вспомним его еще раз: «Итерация – это законченный цикл разработки, приводящий к выпуску **выполнимого изделия** (внутренней или внешней версии) или **подмножества конечного продукта**, которое возрастает с приращением от итерации к итерации, чтобы стать законченной системой.

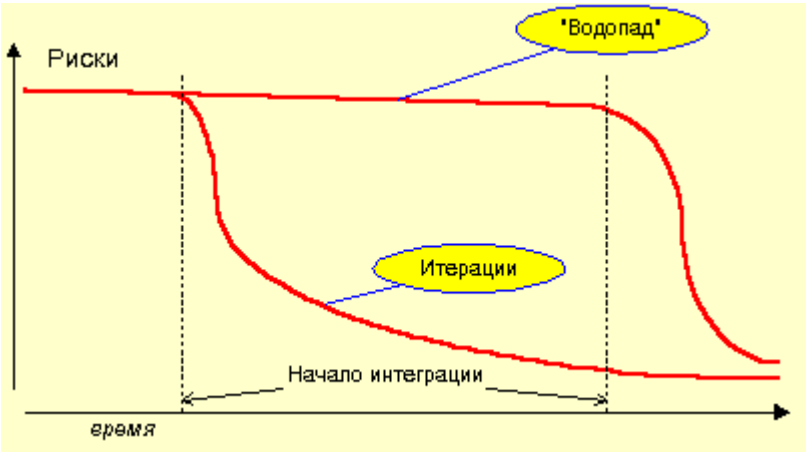


Сравните «водопад» (вверху) с итерационным подходом. Во втором случае каждая итерация начинается анализом и заканчивается выпуском изделия

Итерационный подход превосходит «водопад» по следующим причинам:

- Он позволяет принимать во внимание изменяющиеся требования. Действительность состоит в том, что требования изменяются. Изменение требований и их «ползучесть» всегда были первоисточниками неудач проектов; они ведут к опозданиям поставок, нарушениям планов, неудовлетворенности заказчиков и бесполезности разработки. Как писал Fred Brooks еще 25 лет назад: «Планируйте бросать так далеко, куда Вы добежите наверняка».
- Элементы интегрируются постепенно: интеграция - не один «большой восклицательный знак» в конце. Итерационный подход - почти непрерывная интеграция. То, что всегда было продолжительным, сомнительным и неприятным, занимая до 40 % всех усилий в конце проекта, теперь разбито на в шесть-девять раз меньшие интеграции, которые начинаются с меньшего количества элементов.
- Он позволяет Вам раньше смягчать риски, потому что интеграция, как правило, есть единственное время обнаружения или адресации рисков. Когда Вы выполняете начальные итерации, Вы проходите все основные потоки работ процесса, выявляя многие аспекты проекта: пригодность инструментальных средств и имеющегося в наличии программного

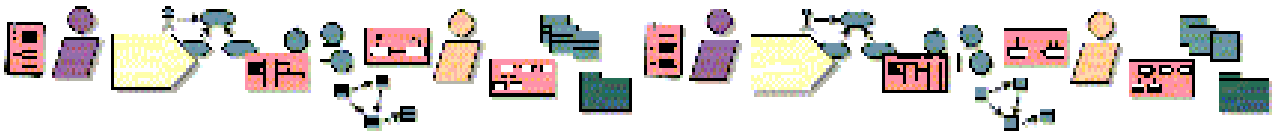
обеспечения, навыки людей и т.д. Некоторые предполагаемые риски могут не проявиться, и могут появляться новые, не предполагавшиеся ранее.



Больше всего мы рискуем, начиная проект. По мере движения вперед ожидаемый риск уменьшается. В процессе интеграции это происходит с наибольшей скоростью. В отличие от «водопада», при итеративном подходе интеграция выполняется уже на первой итерации.

Примечание: Два последних рисунка выполнены в одном временном масштабе.

- Он обеспечивает такое управление, при котором возможно делать тактические изменения в изделии; например, чтобы конкурировать с существующими изделиями. Вы можете принять решение выпустить изделие раньше с уменьшенными функциональными возможностями, чтобы противостоять продвижению конкурента, или Вы можете использовать продукт другого продавца для решения своих проблем.
- Он облегчает многократное использование компонентов - проще идентифицировать общие части, когда они частично разработаны или реализованы, чем идентифицировать все общие части с самого начала. Идентификация и разработка многократно используемых частей трудна. Критический анализ проектных решений на начальных итерациях позволяет архитекторам идентифицировать потенциальное многократное использование и разрабатывать и совершенствовать общий код в последующих итерациях.
- Он приводит к более устойчивой архитектуре - в нескольких итерациях Вы исправите больше ошибок. Слабые места обнаруживаются даже в ранних итерациях. Одновременно обнаруживаются и еще могут быть легко исправлены критические параметры эффективности, отпадает необходимость делать это накануне развертывания.
- Разработчики учатся по ходу и более полно используют различные навыки и специальные знания в течение всего жизненного цикла. Испытатели раньше запускают тесты, технические специалисты раньше делают свои записи и т.д. При не итерационной разработке те же самые люди ожидали бы начала своей работы. Это время можно было бы использовать для оттачивания своих навыков. Но потребности в обучении или в дополнительной (возможно внешней) помощи в это время, в процессе предварительной оценки, еще покрыты мраком.



- Процесс может быть улучшен и усовершенствован по ходу дела. Оценка в конце каждой итерации предусматривает не только анализ состояния изделия и перспектив выполнения графика продвижения проекта, но и того, что должно быть изменено в организации и непосредственно в процессе, чтобы улучшить его в следующей итерации.

При подходе «водопада» все выглядит высококачественным почти до конца проекта, иногда вплоть до середины интеграции. Затем все разваливается. При итерационном подходе очень трудно скрывать правду очень долго.

Организаторы проекта часто сопротивляются итерационному подходу, видя в нем своего рода бесконечное «перемалывание» сделанного. В Rational Unified Process интерактивный подход очень управляем; итерации планируются по числу, продолжительности и цели. Задачи и ответственности участников определены. Объективные критерии продвижения зафиксированы. Некоторая доработка от одной итерации до другой имеет место, но она также тщательно управляется.

Управление требованиями

Управление требованиями – это систематический подход к обнаружению, документированию, организации и сопровождению изменяющихся требований к системе.

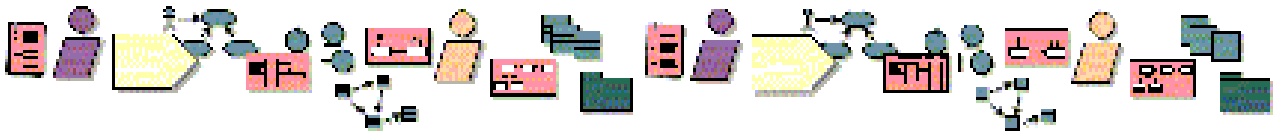
Требование – это условие или возможность, которой должна соответствовать система.

Наше формальное определение управления требованиями предопределяет систематический подход

- к обнаружению, организации и документированию требований к системе, и
- к установлению и поддержанию соглашений между заказчиком и проектной группой об изменяющихся требованиях к системе.

Сбор требований может показаться довольно тривиальной задачей. Однако, в реальных проектах Вы столкнетесь с существенными трудностями потому, что:

- Требования не всегда очевидны и могут исходить из разных источников.
- Требования не всегда удается ясно выразить словами.
- Имеется много различных типов требований на различных уровнях детализации.
- Число требований может стать неуправляемым, если ими не управлять.
- Требования связаны друг с другом, а также с другими представлениями процесса разработки программного обеспечения.
- Требования имеют уникальные свойства или значения свойств. Например, они не являются ни одинаково важными, ни одинаково простыми для согласования.
- Есть много заинтересованных сторон и это означает, что требования должны управляться



перекрестно-функциональными группами людей.

- Требования изменяются.

Какими навыками нужно обладать, чтобы справиться этими трудностями? Наиболее важные из них это умение:

- Анализировать проблемы
- Определять потребности совладельцев
- Определять систему
- Управлять контекстом проекта
- Уточнять определение системы
- Управлять изменением требований

Кто такие совладельцы?

На интуитивном уровне это понятие можно прокомментировать следующим образом:

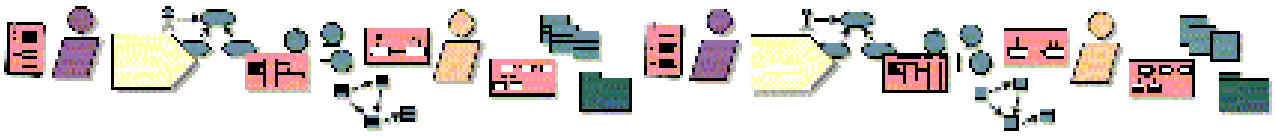
Эффективное решение любой сложной проблемы предполагает удовлетворение потребностей некоторой группы совладельцев. Совладельцы обычно имеют различные точки зрения на проблему и различные потребности, которые должны быть удовлетворены принимаемыми решениями. Многие совладельцы являются пользователями системы. Другие совладельцы - только косвенные пользователи системы или только используют результаты, на которые влияет система. К совладельцам могут быть отнесены покупатели или сторонники приобретения системы. Понимание того, кто является совладельцами и их специфических потребностей, является важным элементом при выборе эффективного решения. Примерами совладельцев являются:

- Представитель заказчика
- Представитель пользователя
- Инвестор
- Руководитель производства
- Покупатель

Анализ проблемы

Анализ проблемы выполняется для того, чтобы понять проблему, начальные потребности совладельцев и предложить решения высокого уровня.

Анализ проблемы - это акт рассуждения и анализа с целью найти «проблему позади проблемы». В ходе анализа достигается взаимное согласие по реальной проблеме и о том, кто заинтересован



в ее разрешении (является ее совладельцами). Кроме того, Вы определяете деловые ограничения решений. Вы должны также проанализировать деловые прецеденты проекта так, чтобы получить хорошее понимание того, какие выгоды ожидаются от инвестиций, вложенных в формируемую систему.

Определение потребностей совладельцев

Требования исходят из многих источников, примерами которых могут быть заказчики, партнеры, конечные пользователи и специалисты предметной области. Вы должны знать, какими должны быть эти источники и как получить к ним доступ, а также как лучше получить от них информацию. Людей, которые являются первичными источниками информации, мы назвали совладельцами проекта. Если Вы разрабатываете информационную систему для использования внутри вашей компании, Вы можете включить в вашу группу разработки людей с опытом конечного пользователя и знанием предметной области и проводить с ними обсуждения на уровне деловой модели. Если вы разрабатываете изделие на продажу, Вы можете привлечь к работе ваших специалистов по маркетингу, чтобы лучше понять потребности заказчиков на рынке.

Действия по выявлению информации могут использовать методы интервью, мозговой атаки, концептуального прототипирования, анкетных опросов и анализа конкурентоспособности. Результатом выявления потребностей может быть список запросов или потребностей, которые описаны в текстовом или графическом виде, с указанием приоритетности относительно друг друга.

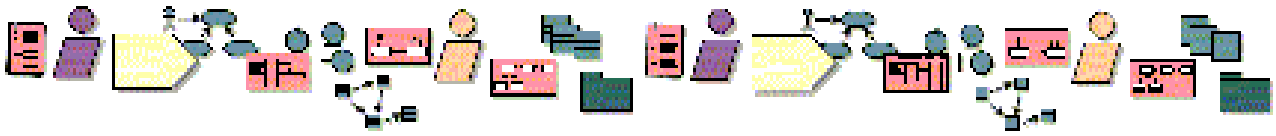
Определение системы

Определение системы формируется путем обработки и реорганизации потребностей совладельцев. Сначала собираются требования, затем выполняется их высокоуровневый анализ и формализация. При этом уточняется специфика требований (что должно быть сделано, с какой степенью детализации, каковы приоритеты, оценки трудоемкости, технические и управленческие риски и начальный контекст). Часть этих требований, непосредственно связанных с наиболее важными запросами совладельцев, может быть представлена в виде черновых прототипов и моделей проекта.

Результатом определения системы является ее описание с использованием естественного языка и графического представления.

Управление контекстом проекта

Для эффективного выполнения проекта Вы должны тщательно отсортировать поступившие от всех совладельцев требования по приоритетам и взять на себя управление их контекстом. Очень много проектов страдают от разработчиков, изготавливающих так называемые «пасхальные яички» (излишества, которые кажутся разработчику интересными и зрелищными), вместо того, чтобы сосредоточиться на задачах, которые смягчают риски или стабилизируют архитектуру



приложения. Чтобы быть уверенным, что Вы разрешили или смягчили риски в проекте как можно раньше, Вы должны разрабатывать вашу систему с приращениями, тщательно выбирая к каждому приращению требования, смягчающие известные риски в проекте. Для этого Вы должны согласовывать контекст каждой итерации с совладельцами проекта. Обычно это требует хороших навыков в управлении предполагаемыми результатами в различных стадиях разработки проекта. Вы должны также контролировать требования (как внешнее представление проекта) и непосредственно процесс разработки.

Уточнение определения системы

Детальное определение системы должно быть представлено таким образом, чтобы ваши совладельцы могли понимать его и соглашаться или не соглашаться с ним. Оно должно фиксировать не только функциональные возможности, но также и соответствие всем юридическим или нормативным требованиям, применимость, достоверность, эффективность, пригодность и надежность в эксплуатации.

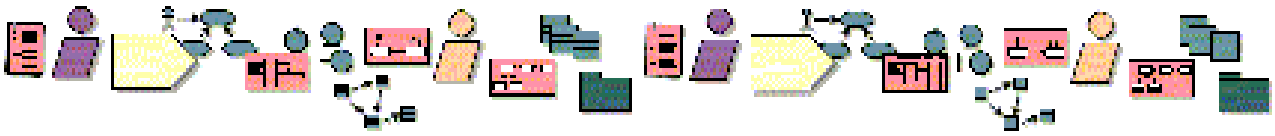
Часто совершаемая ошибка состоит в том, что Ваше ощущение комплексности разработки толкает Вас на формирование комплексного определения. Это ведет к трудностям в объяснении цели проекта и системы. Исполнители могут быть искренне увлечены работой, но они не дадут хорошей отдачи, так как у них нет полного понимания, что за продукт они производят. Вы должны приложить максимум усилий для понимания аудиторией тех документов, которые Вы производите для описания системы. Зачастую возникает потребность производить различные виды описания применительно к аудитории.

Мы видели, что методология прецедента, часто в комбинации с простыми визуальными прототипами, является очень эффективным способом указания цели системы и определения ее подробностей. Прецеденты помогают помещать требования в контекст, они сообщают историю того, как будет использоваться система.

Другой компонент детального определения системы - это констатация того, как система должна быть испытана. Планы проведения и определение предметов испытания говорят о том, какие возможности системы будут проверены.

Управление изменением требований

Независимо от того, насколько Вы осторожны при определении ваших требований, всегда будут вещи, которые потребуют изменения. Что делает изменяющиеся требования сложными для управления? Изменение требования означает не только то, что должно быть потрачено больше или меньше времени на реализацию новой конкретной возможности, но также и то, что изменение в одном требовании может вступить в конфликт с другими требованиями. Вы должны удостовериться, что Вы придаете вашим требованиям структуру, которая гибка к изменениям, и что Вы используете ссылки трассируемости для представления зависимостей между требованиями и другими артефактами (производимыми Вами искуственными объектами)



разработки. Управление изменением включает действия подобные базированию, определению зависимостей, которые важно проследить, устанавливая трассируемость между связанными предметами и управлением изменением.

Инструментальная поддержка

Rational Unified Process в рамках Rational Suite интегрирован с Rational Requisite Pro, который представляет собой инструмент для **управления требованиями.** С его помощью выполняются фиксация, организация, расположение по приоритетам и прослеживание всех требований.

Акцент на архитектуре

Rational Unified Process от начала до конца жизненного цикла управляется прецедентами, но действия проектирования сосредоточены вокруг понятия **архитектуры** – архитектуры системы, или для преимущественно программных систем, архитектуры приложения. Основной упор на ранних итерациях процесса - главным образом в стадии уточнения – делается на производство и проверку правильности **архитектуры приложения.** В начальном цикле развития архитектурные решения материализуются в форме выполняемого архитектурного прототипа, который постепенно развивается, чтобы на более поздних итерациях стать законченной системой.

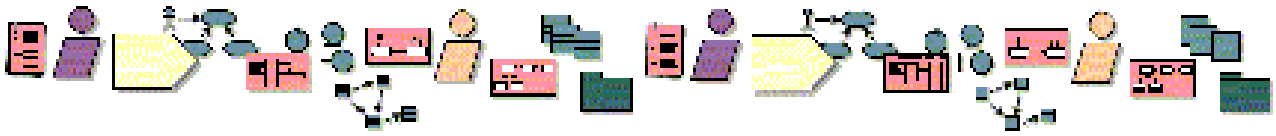
Что такое архитектура приложения?

Архитектура приложения – это понятие, которое интуитивно чувствует большинство проектировщиков, особенно опытных, и которое достаточно сложно точно определить. В частности, трудно провести четкую границу между проектом и архитектурой. Архитектура - один из аспектов проекта, который концентрируется на некоторых определенных особенностях.

Во «Введении в архитектуру приложений» David Garlan и Mary Shaw определяют архитектуру приложения как уровень проекта, отражающий «… проблемы проектирования и определения структуры комплексной системы вне алгоритмов вычислений и структур данных. Структурные проблемы включают общую организацию и структуру глобального управления, протоколы связи, синхронизации и доступа к данным, распределение функциональных возможностей между модулями, физическое распределение, композицию элементов проекта и т.п.»

Другие считают, что архитектура – это больше, чем только структура. Рабочая группа IEEE по архитектуре определяет ее как «высокоуровневую концепцию системы **в ее среде**». Это определение связывает «пригодность» с системной целостностью, экономическими связями, эстетическим восприятием и со стилем. Но оно не ограничено взглядом внутрь, а рассматривает систему в целом в ее операционной среде и среде разработки – взгляд наружу.

В Rational Unified Process архитектура системы программного обеспечения (в данном случае) – это организация или структура существенных компонентов системы, взаимодействующих через интерфейсы с компонентами, составленными из последовательно меньших компонентов и



интерфейсов.

Почему так нужен проект архитектуры?

- Наличие проекта архитектуры позволяет Вам иметь интеллектуальный контроль над проектом, управлять его сложностью и поддерживать целостность системы.

  Сложная система – это нечто большее, чем сумма ее частей, чем последовательность маленьких независимых тактических решений. Она должна иметь некоторую объединяющую структуру, позволяющую организовать эти части и обеспечивать точные правила роста. Система, если не учитывать ее сложность, «взрывается» вне человеческого понимания.

  Архитектура закладывает основу для понимания всего проекта, устанавливая общий набор справочников, общий словарь и т.д.

- Архитектура является эффективной базой для крупномасштабного многократного использования.

  Ясно показывая крупные компоненты и критические интерфейсы между ними, архитектура позволяет Вам рассуждать о многократном использовании: о внутреннем многократном использовании - идентифицировать общие части, и о внешнем многократном использовании – объединять готовые изделия и имеющиеся в наличии компоненты. Она позволяет также многократное использование в большем масштабе: многократное использование самой архитектуры в потоке производства приложений, - т.е. в производстве, которое объединяет различные функциональные возможности в одной системе.

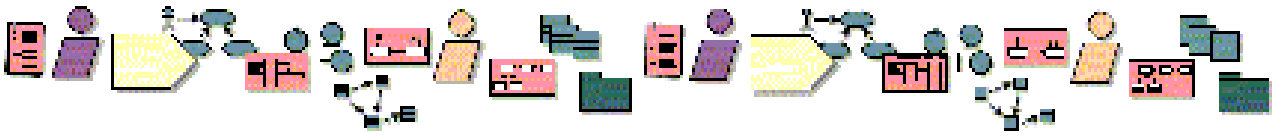
- Архитектура является базой для руководства проектом.

  Планирование и персонал организуются по линиям крупных компонентов. Фундаментальные структурные решения принимаются небольшой централизованной архитектурной группой. Разработка разбивается на разделы среди маленьких групп, каждая из которых отвечает за одну или несколько частей системы.

Описание архитектуры

Архитектура приложения описывается множеством ее **представлений**. Каждое представление архитектуры отражает некоторый аспект, интересующий группу совладельцев проекта: конечных пользователей, проектировщиков, администраторов, системотехников, эксплуатационщиков и т.д.

Представления фиксируют главные решения проектирования структуры, показывая, как приложение разделено на **компоненты**, и как связаны эти компоненты для получения полезной **конфигурации**. Эти решения должны вытекать из **требований** функциональности и других факторов. С другой стороны, эти решения устанавливают дальнейшие **ограничения** на требования и на будущие проектные решения нижнего уровня.

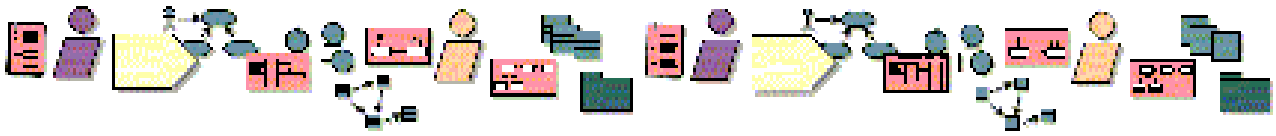


Представления архитектуры по существу являются выжимками, иллюстрирующими «архитектурно существенные» элементы моделей. В Rational Unified Process Вы начинаете с типичного набора представлений, называемого моделью представления «4+1».

Модель представления архитектуры «4+1»

Этот набор включает:

- **представление прецедентов**, которое содержит прецеденты и сценарии, охватывающие архитектурно существенное поведение, классы или технические риски. Оно является подмножеством модели прецедентов
- **логическое представление**, которое содержит наиболее важные классы проекта, их организацию в пакеты и подсистемы различных уровней. Здесь содержится уже некоторая реализация прецедента. Это представление является подмножеством модели проекта
- **представление выполнения**, которое содержит краткий обзор модели выполнения в терминах модулей пакетов и уровней. Здесь описывается также распределение пакетов и классов (из логического представления) в пакетах и модулях представления выполнения. Это представление является подмножеством модели выполнения
- **представление процесса**, которое содержит описание задач (процессов и нитей), их взаимодействия и конфигурации и распределения объектов и классов по задачам. Потребность этого представления возникает только, если система имеет существенную степень параллелизма. В Rational Unified Process 5.1 представление процесса - это подмножество модели проекта



- **представление развертывания**, которое содержит описание различных физических узлов для наиболее типичных конфигураций платформы, и расположение задач (из представления процесса) по физическим узлам. Потребность этого представления возникает только, если система распределена.

В Rational Unified Process представления архитектуры регистрируются в документе **Архитектура приложения**.

Вы можете предлагать дополнительные представления для выражения различных специальных потребностей: представление интерфейса пользователя, представление защиты, представление данных и так далее. Для простых систем Вы можете опустить некоторых из представлений, содержащихся в «модели вида 4+1».

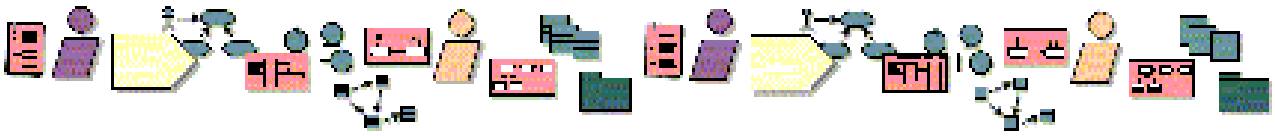
Круг интересов архитектуры

Перечисленные выше представления могут изобразить проект системы в целом. Но архитектура интересуется только некоторыми определенными аспектами:

- **Структура** модели - организационные элементы, например, иерархическое представление.
- **Существенные элементы** - критические прецеденты, главные классы, общие механизмы и так далее, а не все элементы, представленные в модели.
- Несколько **ключевых сценариев**, показывающих главный поток управления в системе.
- **Сервис**, чтобы зафиксировать модульность, необязательные особенности, аспекты производственной линии.

В сущности, архитектурные представления - это **абстракции** или упрощения полного проекта, в которых важные характеристики сделаны более яркими, подробности оставлены в стороне. Эти характеристики важны при рассуждении относительно

- развития системы – переходу к следующему циклу разработки
- многократного использования архитектуры или ее частей в контексте производственной линии
- оценки дополнительных качеств типа эффективности, доступности, взаимозаменяемости и безопасности
- распределения проектных работ между группами или субподрядчиками
- решения о включении имеющихся в наличии компонентов
- включения в более широкую систему.



Методическая поддержка

Rational Unified Process предлагает систематический способ проектирования, разработки и проверки правильности архитектуры. Он предоставляет шаблоны для описания архитектуры и для сбора данных, касающихся архитектуры. Поток работ проектирования содержит специальные действия, нацеленные на идентификацию архитектурных связей и существенных архитектурных элементов, а также рекомендации по принятию архитектурных решений. Поток работ управления проектом показывает, как планирование ранних итераций учитывает проектирование архитектуры и ликвидацию главных технических рисков.

Шаблоны архитектуры – это готовые формы, которые решают повторяющиеся архитектурные проблемы. **Архитектурный каркас** или **архитектурная инфраструктура** - набор компонентов, из которых Вы можете формировать некоторый вид архитектуры. Многие из архитектурных трудностей были преодолены путем использования каркасов или инфраструктур, обычно связанных с определенной областью: управление и контроль, административные информационные системы и т.д.

Архитектура приложения или только какое-то представление архитектуры может иметь атрибут под названием **архитектурный стиль**, который предоставляет набор возможных форм, что позволяет применить к архитектуре некоторую степень унификации. Стиль может быть определен набором образцов или выбором определенных компонентов или разъемов как базовых строительных блоков. Для данной системы некоторый стиль может быть зафиксирован как раздел описания архитектуры в **руководстве по архитектурному стилю** – отдельной части документации по руководящим принципам проектирования в Rational Unified Process. Стиль играет важную роль в понятности и целостности архитектуры.

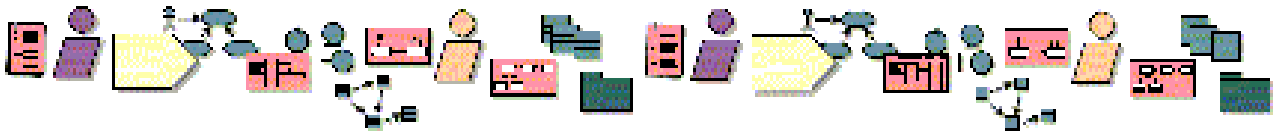
Архитектура в Rational Unified Process – это, прежде всего, результат потока работ анализа и проектирования. Так как проект повторяет этот поток работ, итерацию за итерацией, архитектура развивается, усовершенствуется и приглаживается. Поскольку каждая итерация включает интеграцию и тестирование, ко времени поставки архитектура будет весьма надежна. Архитектура является главным делом итераций стадии разработки, в конце которой архитектура обычно бывает сформирована.

Инструментальная поддержка

Выше уже упоминалось, что Rational Unified Process в рамках Rational Suite интегрирован с Rational Rose, который представляет собой инструмент моделирования с использованием системы обозначений UML.

Для графического описания поясненных выше представлений архитектуры используются следующие диаграммы UML:

- **Представление прецедентов**: диаграммы прецедентов, изображающие прецеденты, субъекты и обычные классы проекта; диаграммы последовательности, изображающие



объекты проекта и их взаимодействия.

- **Логическое представление**: диаграммы классов, диаграммы состояний и диаграммы объектов.
- **Представление выполнения**: диаграммы компонентов.
- **Представление процесса**: диаграммы классов и диаграммы объектов, затрагивающих задачу - процессы и нити.
- **Представление развертывания**: диаграммы развертывания.

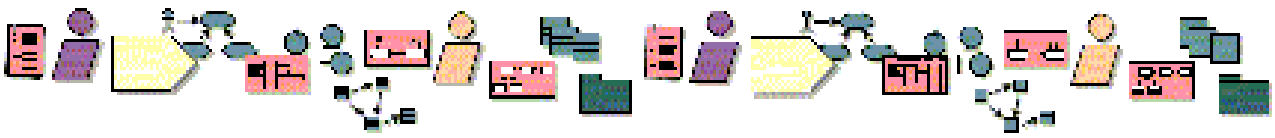
Разработка на базе компонентов

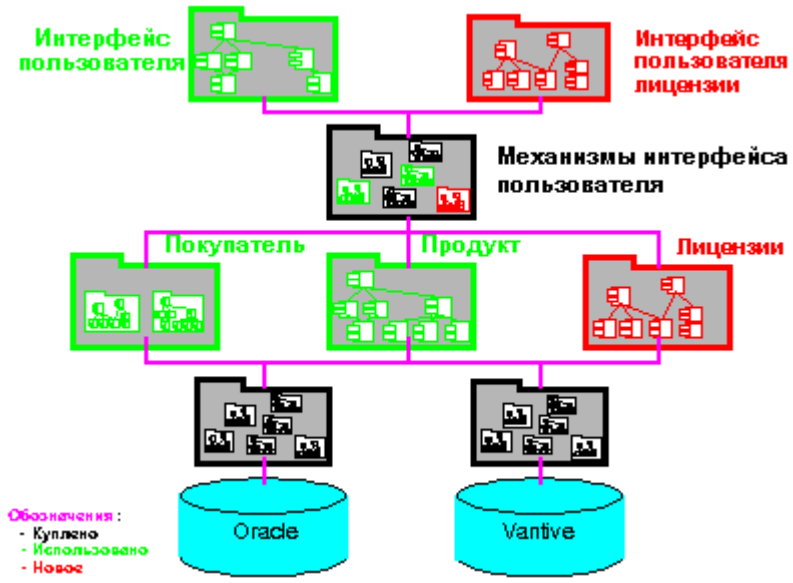
Программный компонент может быть определен как нетривиальная часть программного обеспечения, модуля, пакета или подсистемы, которая выполняет ясную функцию, имеет ясную границу и может быть интегрирована в четко определенной архитектуре. Это - физическая реализация абстракции в вашем проекте.

Компоненты появляются из различных источников:

- Внутри модульной архитектуры Вы идентифицируете, изолируете, проектируете, разрабатываете и тестируете правильно построенные компоненты. Эти компоненты могут быть индивидуально проверены и постепенно интегрированы в цельную систему.
- Кроме того, некоторые из этих компонентов могут быть разработаны так, чтобы их можно было перенастраивать, особенно компоненты, обеспечивающие общие решения в широких пределах общих проблем. Эти перенастраиваемые компоненты, которые могут быть больше, чем просто совокупностями утилит или библиотеками классов, образуют базу для многократного использования в пределах организации, увеличивая производительность и качество программирования.
- Недавнее появление большого числа коммерчески успешных компонентных инфраструктур, типа CORBA, Internet, ActiveX или JavaBeans, вызвало бурный рост индустрии производства компонентов для различных прикладных областей, позволяя Вам покупать и интегрировать компоненты вместо разрабатывания их внутри организации.

Первое появление эксплуатирует старые концепции модульности и инкапсуляции, на шаг далее продвигая концепции, лежащие в основе объектно-ориентированной технологии. Два последних переориентируют разработку программ от программирования к сборке программного обеспечения (трансляции компонентов).





В архитектуре этого приложения используются вновь разработанные, модифицированные старые и покупные компоненты (они обозначены разными цветами)

Rational Unified Process поддерживает компонентно-ориентированную разработку в нескольких направлениях:

- Итерационный подход позволяет Вам постепенно идентифицировать компоненты, решая какой из них разрабатывать, какой многократно использовать и который купить.
- Акцент на архитектуре программы позволяет Вам соединять структуру (компоненты) и способы их интеграции (фундаментальные механизмы и характер их взаимодействия).
- Концепции пакетов, подсистем и уровней используются в процессе анализа и проектирования для организации компонентов и определения их интерфейсов.
- Тестирование организуется сначала для одиночных компонентов, затем для все более крупных наборов интегрируемых компонентов.

Настраиваемый процесс

Rational Unified Process является достаточно общим и совершенным, чтобы использоваться широким кругом разработчиков программного обеспечения. Во многих случаях этот процесс должен быть изменен, откорректирован, расширен и приспособлен к определенным характеристикам, связям и традициям конкретной организации.

Элементы процесса, которые могут изменяться, настраиваться, добавляться или исключаться, включают: артефакты, действия, потоки работ и работников.

С возможностями настройки процесса к нуждам конкретного проекта и конкретной организации мы познакомимся более подробно при рассмотрении потока работ «Среда».