МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Лекция № 16

Методология Rational Unified Process

Методы и средства проектирования информационно-аналитических систем

(наименование дисциплины (модуля) в соответствии с учебным планом)

Уровень: специалитет

(бакалавриат, магистратура, специалитет)

Форма обучения: очная

(очная, очно-заочная, заочная)

Направление(-я) подготовки: 10.05.04 «Информационно-аналитические системы безопасности»

(код(-ы) и наименование(-я))

Институт: Институт кибербезопасности и цифровых технологий (ИКБ)

(полное и краткое наименование)

Кафедра: Информационно-аналитические системы кибербезопасности (КБ-2)

(полное и краткое наименование кафедры, реализующей дисциплину (модуль))

Используются в данной редакции с учебного года 2023/24

(учебный год цифрами)

Проверено и согласовано «____» _____20___г.

(подпись директора Института/Филиала с расшифровкой)

Москва 2024 г.

Учебные вопросы:

1. **Методология Rational Unified Process**
2. **Структура процесса**
3. **Моделирование требований к информационной системе. Диаграмма требований**
4. **Анализ требования в Requisite Pro**

1. Методология Rational Unified Process

Rational Unified Process (RUP)- это процесс разработки программного обеспечения. Его цель состоит в том, чтобы гарантировать высокое качество программного продукта, отвечающего потребностям конечных пользователей, в пределах предсказуемого графика и бюджета выполнения. *RUP* обеспечивает строгий подход к решению задач проектирования и ответственности разработчиков.

Rational Unified Process – это итеративный процесс (так называемая спиральная модель жизненного цикла разработки). Каждая итерация может приводить к созданию фрагмента разрабатываемой системы или новой версии и включает этапы выработки требований, анализа, проектирования, реализации и тестирования. Поскольку тестирование проводится на каждой итерации, риск снижается уже на начальных этапах жизненного цикла разработки.

Итеративный подход позволяет увеличивать понимание проблемы через последовательные усовершенствования и повышать эффективность проектных решений. Этот подход обеспечивает лучшую гибкость в учете новых требований или тактических изменений в деловых целях, и позволяет проектировщику заранее идентифицировать и разрешать риски.

Rational Unified Process – это управляемый процесс. Итеративный подход предполагает управление требованиями и управление изменениями, чтобы своевременно и по всем пунктам обеспечивать общее понимание ожидаемых функциональных возможностей, ожидаемый уровень качества, и гарантировать наилучшее управление связанными затратами и графиками выполнения.

Rational Unified Process заключается в создании и обслуживании моделей. *RUP* фокусирует внимание не на создании большого количества бумажных документов, а на развитии и эксплуатациb моделей - семантически богатых представлений программной системы при ее разработке.

Rational Unified Process концентрирует внимание на первоначальной разработке и компоновке устойчивой архитектуры программы, которая облегчает параллельную разработку, минимизирует переделки, увеличивает возможность многократного использования и надежность эксплуатации. Эта архитектура используется для планирования использования и управления развитием программных компонентов.

Действия при выполнении *Rational Unified Process* управляются прецедентами. Определение прецедентов и сценария управляет технологическим маршрутом от делового моделирования и требований до

испытаний, и обеспечивают связанные и доступные для анализа маршруты разработки и поставки системы.

Rational Unified Process поддерживает объектно-ориентированную технологию. Объектно-ориентированные модели базируются на понятиях объектов, классов и зависимостей между ними. Эти модели, подобно многим другим техническим искусственным объектам (артефактам), используют унифицированный язык моделирования (*UML*) как общую систему обозначений.

Rational Unified Process поддерживает компонентно-ориентированное программирование. Компоненты - это нетривиальные модули или подсистемы, которые выполняют конкретную функцию и могут быть смонтированы в строго очерченной архитектуре, специальной или некоторой общедоступной инфраструктуре компонентов, типа *Internet, CORBA, COM/DCOM*, для которых появляется индустрия многократно используемых компонентов.

Rational Unified Process – это процесс с перестраиваемой конфигурацией. *Rational Unified Process* удовлетворяет и маленькие группы разработчиков, и большие организации. *Rational Unified Process* основан на простой и корректной архитектуре, которая обеспечивает общность для семейства процессов, и все же может быть адаптирована к конкретным ситуациям. *RUP* содержит рекомендации о том, как сконфигурировать процесс, чтобы удовлетворить потребности данной организации.

Rational Unified Process поощряет объективно осуществляемое управление **качеством**. Оценка качества всех действий и их участников, формируемая в процессе, использует объективные измерения и критерии.

Rational Unified Process поддерживается инструментальными средствами, которые автоматизируют большинство действий процесса. Инструментальные средства используются для создания и обслуживания различных артефактов - моделей в частности - процесса разработки программного обеспечения: визуального моделирования, программирования, испытаний и так далее. Инструментальные средства осуществляют информационную поддержку управления изменениями, которыми сопровождается каждая итерация.

Рассмотрим, как понимается в RUP процесс разработки программы. ***Процесс*** - частично упорядоченный набор шагов, которые нужно проделать для достижения цели; при разработке программного обеспечения цель состоит в формировании или расширении существующего программного изделия; при разработке процессов цель состоит в том, чтобы разработать или расширить процесс. Таким образом, процесс программирования - деловой процесс. *Rational Unified Process* представляет собой универсальный деловой процесс объектно-ориентированной разработки программного обеспечения. Он описывает семейство связанных процессов разработки программного обеспечения, совместно использующих общую структуру и имеющих общую архитектуру.

Когда программная система разрабатывается заново, ее разработка – это процесс создания системы из требований. Но как только система приняла

какую-то форму (или в наших терминах, когда система прошла начальный цикл развития), то любая дополнительная разработка – это процесс приспособления системы к новым или изменившимся требованиям. Это называют жизненным циклом системы.

Процесс разработки программного обеспечения - процесс разработки системы из требований, новых (начальный цикл развития) или измененных (цикл эволюции).

Чтобы понять *Rational Unified Process*, рассмотрим два следующих измерения:

- По основным потокам работ (какова по характеру логика действий групп разработчиков).
- По времени (аспекты жизненного цикла процесса, как он разворачивается).

Процесс организован во времени (стадии) и по содержанию (основные потоки работ).

Первое измерение представляет **статический** аспект процесса: оно описано в терминах основных потоков работ (действия, потоки работ и так далее).

Второе измерение представляет **динамический** аспект процесса, поскольку оно выражено в терминах циклов, стадий, итераций и этапов.

Описание процесса часто выполняется с двух различных точек зрения:

- **Техническая** точка зрения, фокусирующаяся на артефактах и представлениях, связанных с разрабатываемым изделием.
- **Организационная** точка зрения, фокусирующаяся на времени, бюджете, людях, процессах и других экономических соображениях.

Если целью объектного моделирования является формирование новых прикладных программ, то деловое моделирование есть часть процесса разработки программного обеспечения. Но объектное моделирование может иметь самостоятельную ценность, если его целью является совершенствование процессов организационного управления.

2. Структура процесса

Статическое содержание процесса организовано в основных потоках работ, которые описаны в терминах действий, работников и артефактов. *Rational Unified Process* включает девять основных потоков работ; шесть потоков работ процесса разработки (1-6) и три потока работ поддержки (7-9):

1. Деловое моделирование
2. Требования
3. Анализ и проектирование
4. Выполнение
5. Испытание
6. Развертывание
7. Управление конфигурацией и изменением
8. Руководство проектом
9. Среда

Структура жизненного цикла

Когда мы рассматриваем динамическую организацию процесса во времени, жизненный цикл программы разбивается на циклы, каждый из которых работает над новым поколением изделия. Rational Unified Process делит один цикл развития на четыре последовательных стадии:

- Начало
- Уточнение
- Конструирование
- Переход

Каждая стадия заканчивается четко определенной вехой – временной точкой, в которой должны быть принятыми некоторые проблемные решения, а поэтому ключевые цели должны быть достигнуты.

Стадии и главные вехи процесса.

Первый цикл выполнения этих четырех стадий для данного изделия называют начальным циклом разработки. Если жизненный цикл изделия на этом не завершается, существующее изделие разовьется в своем следующем поколении, повторив ту же последовательность стадий начала, уточнения, конструирования и перехода. Этот период называется "эволюцией". Циклы развития, которые следуют за начальным циклом развития, называются "эволюционными циклами".

Начальная стадия

На начальной стадии разработчик устанавливает деловые применения системы и определяет рамки проекта. Чтобы сделать это, нужно идентифицировать все внешние объекты, с которыми взаимодействует система (субъекты), и определить характер этого взаимодействия на высоком уровне. Эта работа включает идентификацию всех прецедентов и описание нескольких наиболее существенных. Деловое применение включает критерии успеха, оценку риска, оценку необходимых ресурсов и укрупненный план с указанием дат завершения главных этапов.

В конце начальной стадии разработчик исследует требования жизненного цикла проекта и решает, следует ли продолжать разработку.

Стадия уточнения

Цели стадии уточнения состоят в том, чтобы проанализировать прикладную область, создать нормальную архитектурную основу, разработать план проекта и устранить самые высокие элементы риска проекта. Архитектурные решения должны приниматься с пониманием целостной системы. Это подразумевает описание большинства прецедентов и рассмотрение дополнительных требований. Чтобы проверить архитектуру, нужно разработать систему, которая демонстрирует архитектурные решения и выполняет существенный прецедент.

В конце стадии уточнения разработчик детально исследует цели и контекст системы, архитектурные решения и способы разрешения главных рисков.

Стадия конструирования

В ходе стадии конструирования происходит итеративная разработка законченного изделия, которое готово к передаче его пользователям. Это

подразумевает описание оставшихся прецедентов, изложение деталей конструкции, завершение выполнения и проверку программного обеспечения.

В конце стадии конструирования разработчик решает, все ли программное обеспечение, рабочие места и пользователи готовы и работоспособны.

Переходная стадия

В процессе переходной стадии разработчик передает программное обеспечение его пользователям. Как только изделие попадает в руки пользователей, часто возникают новые проблемы, которые требуют дополнительной разработки для корректировки системы, исправлению не обнаруженных ранее проблем или завершению реализации некоторых возможностей, которые, возможно, были отложены. Эта стадия обычно начинается с выпуска "бета-версии" системы.

В конце переходной стадии разработчик решает, были ли достигнуты цели жизненного цикла, и возможно, запускает другой цикл разработки. Это также та точка, в которой разработчик может проанализировать результаты и сделать выводы из некоторых уроков, полученных в процессе разработки проекта.

Итерации

Каждая стадия Rational Unified Process может быть в свою очередь разбита на итерации. 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.

Каждая итерация в пределах стадии приводит к выпуску выполнимой системы. Каждая итерация содержит все аспекты программирования и повторяет все основные потоки работ, хотя акценты на основных потоках работ устанавливаются различные, в зависимости от стадии. Каждый из основных потоков работ ответственен за набор артефактов.

Это положение иллюстрируется диаграммой на рис.*, где показана трудоемкость каждого из основных потоков работ по мере продвижения разработки от итерации к итерации через все четыре стадии. Высота кривых отражает количество ресурсов.

Главное последствие такого итерационного подхода – артефакты, описанные ранее, обогащаются и через какое-то время становятся полностью оформленными, как это показано на следующей диаграмме.

Обзор процесса

Основные потоки работ описаны в терминах работников, действий и потоков работ.

- **Работник** определяет поведение и ответственности индивидуума или нескольких индивидуумов, работающих вместе как группа. Это - важное отличие, потому что обычно о работнике думают как о конкретном индивидууме или группе. В *Rational Unified Process*, работник - больше роль, которая определяет, как индивидуумы должны выполнять работу.

- **Действие** – это самая маленькая часть работы, которая относится к делу; его можно интерпретировать как "техническую операцию" работника. Далее, невозможно выполнить только часть действия, хотя в пределах действия может существовать некоторая необязательная операция. Такое разделение работы облегчает возможность контролировать разработку. Гораздо лучше (проще) знать, что в проекте реализованы три из пяти действий, чем то, что выполнено 60 % проекта.
- **Искусственные объекты (артефакты)** - конструкции моделирования и документы, которые действия выделяют, поддерживают или используют как ввод.

С каждым работником ассоциируется набор "связных" действий; связность означает, что действия будут лучше выполнены этим индивидуумом. Ответственности каждого работника обычно определяются относительно некоторых артефактов, например документов. Примеры работников - аналитик делового процесса, проектировщик, архитектор или инженер-технолог.

Каждый работник имеет свой собственный набор действий и артефактов

Через связанный набор действий работник неявно определяет также и умения, необходимые для исполнения работ.

Для каждого основного потока работ представляется диаграмма **краткого обзора действий**. Эта диаграмма показывает все действия и всех работников, включенных в поток работ.

Для каждого основного потока работ представлена диаграмма **краткого обзора артефактов**. Эта диаграмма показывает все артефакты и всех работников, включенных в поток работ.

Таким образом, *RUP* является энциклопедией (методологическим руководством) того, как нужно строить эффективное информационное производство. *RUP* также регламентирует этапы разработки программного обеспечения, документы, сопровождающие каждый этап. Методология постоянно обновляется, включая в себя все новые и новые возможности. К достоинством данной методологии стоит отнести чрезвычайную гибкость, то есть *RUP* не диктует, что необходимо сделать, а только рекомендует использовать то или иное средство.

Обобщим решения, которые предлагает *RUP* для разработки программного обеспечения:

1. Выпускать программное обеспечение, пользуясь принципом промышленного подхода. То есть так, как поступают любые заводы и фабрики: определяя стадии, потоки, уточняя обязанности каждого участника проекта. Именно промышленный подход позволит достаточно оперативно выпускать новые версии ПО, которые при этом будут надежными и качественными.
2. Расширять кругозор специалистов для снятия барьеров. Ведь в подавляющем большинстве случаев специалисты из разных отделов просто говорят на разных языках. Соответственно, снятие языкового барьера должно вести к ускорению работы над программным обеспечением.

3. Использовать итеративную разработку вместо каскадной, существующей в настоящее время. Принцип итерации заключается в повторяемости определенной последовательности процессов с целью доведения элемента до безошибочного состояния.

4. Обязательное управление требованиями. Всем известно, что по ходу разработки в систему вносятся изменения (самой группой разработчиков или заказчиком – неважно). *RUP* предлагает мощную систему контроля управления требованиями: их обнаружение и документирование, поддержку соглашений между разработчиками и заказчиками.

5. Полный контроль всего происходящего в проекте посредством создания специальных архивов.

6. Унифицированный документооборот, приведенный в соответствие со всеми известными стандартами. Это значит, что каждый этап в разработке (начало, работа и завершение) сопровождаются унифицированными документами, которыми должен пользоваться каждый участник проекта.

7. Использование визуального моделирования.

8. Применение не только механизмов объектно-ориентированного программирования, но и объектно-ориентированного мышления.

RUP представлен в виде “on-line” документации, оформленной как web-страницы, что позволяет размещать его описание на сервере внутренней сети предприятия с целью приобщения всех сотрудников к объектно-ориентированной методологии.

3. Моделирование требований к информационной системе. Диаграмма требований

Анализ требований к разрабатываемой информационной системе

В соответствии с методологией объектно-ориентированного анализа и проектирования первым этапом является анализ требований, который подразумевает выделение процессов и требований и их формулировку в виде прецедентов. По результатам анализа прецедентов на первом этапе моделирования предметной области создается диаграмма определения требований к системе Use Case (сценарии поведения). Данная диаграмма позволяет создавать диаграммы поведения объектов системы. Это представление работы системы с точки зрения акторов (actors), то есть объектов, выполняющих в системе определенные функции [5,6].

Элемент *Use Case* - это согласованный блок функциональности, которую представляет система, подсистема или класс. Этот блок описывает последовательности сообщений, которыми обменивается система и один или несколько внешних пользователей (актеров), а также действия, осуществляемые при этом системой [UML]. Прецеденты являются описанием или вариантами использования системы. С помощью прецедента описывается некоторый процесс. К взаимодействию относятся только отношения между системой и актерами; внутреннее поведение и реализация системы скрыты.

Разработка, управляемая прецедентами

В Rational Unified Process "красной нитью", объединяющей систему при выполнении отдельных задач, являются **прецеденты**, которые определяют поведение системы. **О**дним из преимуществ Rational Unified Process является его "управляемый прецедентами подход". Это предполагает, что определенный для системы прецедент является основанием для всего процесса разработки.

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

На диаграмме прецедентов иллюстрируется набор прецедентов системы и исполнители, а также взаимосвязи между ними. Прецеденты определяют, как исполнители взаимодействуют с программной системой. В процессе этого взаимодействия исполнителем генерируются события, передаваемые системе, которые представляют собой запросы на выполнение некоторой операции. Как правило, отдельные шаги или виды деятельности в виде прецедента не представляются.

При создании диаграмм использования вначале определяются исполнители (роли, пользователи). Исполнитель (*actor*) является внешним по отношению к системе понятием, которое определенным образом участвует в процессе, описываемом прецедентом. Актер представляет собой некую идеализацию намерения пользователя, а не самого этого пользователя. Один реальный пользователь может соответствовать нескольким актерам, а один актер может представлять одно и то же намерение сразу нескольких пользователей.

Затем идентифицируются прецеденты.

При разработке систем искусственного интеллекта предлагается создавать типовые альтернативные прецеденты для классов проблемных ситуаций, выделенных, например, в результате кластерного анализа.

На основе набора Use Case диаграмм создается список требований к системе и определяется множество выполняемых системой функций.

Модель деловых прецедентов состоит из **деловых субъектов** и **деловых прецедентов**. Деловые субъекты представляют деловых пользователей, например, заказчиков, продавцов и внешние системы, с которыми взаимодействует организация. Деловые субъекты помогают ограничить рамки описания организации. Деловые прецеденты отображают деловые процессы.

Прецеденты представляют поведение системы. Каждый прецедент подробно описывается. Поток событий прецедентов показывает, как система шаг за шагом взаимодействует с субъектами и что делает система.

Объединяющая функция прецедентов прослеживается на всем протяжении цикла развития системы. Одна и та же модель прецедентов используется в потоках работ Требования, Анализ и проектирование и Испытания.

Создание прецедентов на диаграмме использования системы (

Поведение разрабатываемой системы (то есть функциональность, обеспечиваемая системой) описывается с помощью модели прецедентов, которая отображает системные прецеденты (use cases), системное окружение

(действующих лиц или актеров –actors) и связи между прецедентами и актерами (диаграммы прецедентов - use cases diagrams). Основная задача модели прецедентов – представлять собой единое средство, дающее возможность заказчику, конечному пользователю и разработчику совместно обсуждать функциональность и поведение системы.

Этот тип диаграмм служит для проведения итерационного цикла общей постановки задачи вместе с заказчиком (рис.3.1). Поскольку заказчик "... и раньше не знал, и теперь не знает, и в обозримом будущем не будет точно знать, что ему нужно. И это не "злой умысел", а объективная реальность", то диаграммы прецедентов как раз и служат основой для достижения взаимопонимания между программистами-профессионалами, разрабатывающими проект, и "бизнесменами" - заказчиками проекта. Эти диаграммы описывают функциональность ИС, которая будет видна пользователям системы, основные функции, которые должны быть включены в систему (use case), их окружение (actors). "Каждая функциональность" изображается в виде "прецедентов использования" (use case) или просто прецедентов. Прецедент - это типичное взаимодействия пользователя с системой, которое при этом:

- описывает видимую пользователем функцию,
- может представлять различные уровни детализации,
- обеспечивает достижение конкретной цели, важной для пользователя.

Актеры не являются частью системы, они используют систему (или используются системой) в данном прецеденте.

Актеры могут:

- Только снабжать информацией систему;
- Только получать информацию из системы;
- Снабжать информацией и получать информацию из системы.

Актер, представляющий человека-пользователя, характеризуется *ролью* в данном прецеденте. На диаграмме изображается только один актер, однако, реальных пользователей, выступающих в данной роли по отношению к ИС, может быть много. Обычно актеры определяются из описания задачи или путем переговоров с заказчиками и экспертами. Для выявления актеров может быть использована следующая группа вопросов:

- Кто заинтересован в определенном системном требовании?
- Какую роль система будет выполнять в организации?
- Кто получит преимущества от использования системы?
- Кто будет снабжать систему информацией, использовать информацию и получать информацию от системы?
- Кто будет осуществлять поддержку и обслуживание системы?
- Использует ли система внешние ресурсы?
- Выступает ли какой-либо участник системы в нескольких ролях?
- Выступают ли различные участники в одной роли?
- Будет ли новая система взаимодействовать со старой?

Определение актеров для системы обычно происходит итеративным путем. В модель желательно включить краткое описание каждого актера, в котором нужно указать роль актера при взаимодействии с системой.

С помощью прецедентов моделируется диалог между актером и системой, другими словами, они определяют возможности, обеспечиваемые системой для актера. Можно сказать, что прецедент – это последовательность транзакций, выполняемых системой, которая приводит к значимому результату для определенного актера.

Чтобы выделить прецеденты для системы, можно использовать следующую серию вопросов:

Каковы задачи каждого актера?

Будет ли актер создавать, хранить, изменять, удалять или получать информацию?

Какой прецедент буде создавать, хранить, изменять, удалять или получать эту информацию?

Должен ли актер информировать систему о внезапных изменениях внешней среды?

Какие прецеденты будут поддерживать и обслуживать систему?

Уровень детализации прецедента: прецедент обычно определяет основной элемент функциональности и совершается от начала до конца. Он должен приносить что-то значимое для актера.

Последовательность создания прецедентов.

Browser>Use case View>New>Use Case

Список всех прецедентов фактически определяет функциональные требования к ИС, с помощью которых может быть сформулировано техническое задание. Прецедент рисуется

как овал, связанный с типичными пользователями-актерами.

Диаграммы Use Cases включают отношения и ассоциации, показывающие взаимодействие между воздействующими объектами и функциями (изображаются в виде стрелок), и примечания (note), которые могут быть привязаны к любому объекту диаграммы Use Cases.

Поток событий для прецедента

Поток событий (flow of events) – это последовательность событий, необходимых для обеспечения требуемого поведения. Поток событий описывается в терминах предметной области.

Если типичный ход событий можно представить в виде одной последовательности, очевидно, что в процессах может быть множество альтернативных вариантов реализации действий. В таком случае возникает проблема определения альтернативных прецедентов. Для этого в прецедент добавляются потоки для исключительных ситуаций (альтернативные потоки).

Между актером и прецедентом может существовать ассоциативное отношение. Ассоциативная связь может быть односторонней или двусторонней. Направление связи показывает, кто является ее инициатором (актер или прецедент). Существует два типа отношений между прецедентами: включает (include) и дополняет (extend). Отношение «включает» изображается

как отношение зависимости , направленное от базового прецедента к используемому.

Отношение дополняет (extend relationship) применяется для отражения:

- дополнительных режимов;
- режимов, которые запускаются только при определенных условиях, например сигнала тревоги;
- альтернативных потоков, которые запускаются по выбору актера.

В каждой системе обычно есть главная диаграмма прецедентов, которая отображает границы системы (актеров) и основное функциональное поведение системы (прецеденты). Другие диаграммы прецедентов могут создаваться при необходимости:

- Диаграмма, показывающая все прецеденты для определенного актера;
- Диаграмма, показывающая все прецеденты, реализованные на данной итерации;
- Диаграмма, показывающая определенный прецедент и все его отношения.

По методологии Rational Unified Process реализации прецедентов отражаются в логическом представлении модели.

Порядок разработки Use Case диаграммы в Rational Rose:

Browse>Use Case View>New>Use Case Diagram

Рассмотрим пример разработки диаграммы Вариантов Использования для системы записи студентов на элективные курсы.. Требуемые для этого действия подробно перечислены далее. Готовая диаграмма Вариантов Использования должна выглядеть как на рисунке *.

Этапы разработки диаграммы

1. Дважды щелкните на Главной диаграмме Вариантов Использования (Main) в броузере, чтобы открыть ее.

2. С помощью кнопки Use Case (Вариант Использования) панели инструментов поместите на диаграмму новый вариант использования.

3. Назовите этот новый вариант использования «Запись на элективный курс»

4. Повторите этапы 2 и 3, чтобы поместить на диаграмму остальные варианты использования: Выбор курса для преподавания, Запрос расписания курса и так далее.

5. С помощью кнопки Actor (Действующее лицо) панели инструментов поместите на диаграмму новое действующее лицо.

6. Назовите его «Студент»

7. Повторите шаги 5 и 6, поместив на диаграмму остальных действующих лиц::Преподаватель, Методист.

Добавить ассоциации

1. С помощью кнопки Unidirectional Association (Однонаправленная ассоциация) панели инструментов нарисуйте ассоциацию между действующим лицом Студент и вариантом использования Запись на элективный курс.

2. Повторите этот этап, чтобы поместить на диаграмму остальные ассоциации.

Добавить связь расширения

1. С помощью кнопки Generalization панели инструментов нарисуйте связь между вариантом использования Завершение регистрации на курс и вариантом использования Запись на элективный курс. Стрелка должна протянуться от первого варианта использования ко второму. Связь расширения означает, что вариант использования "Завершение регистрации на курс при необходимости дополняет функциональные возможности варианта использования Запись на элективный курс.

2. Щелкните правой кнопкой мыши на новой связи между вариантами использования " Завершение регистрации на курс " и " Запись на элективный курс ".

3. В открывшемся меню выберите пункт Open Specification (Открыть спецификацию).

4. В раскрывающемся списке стереотипов введите слово extends (расширение), затем нажмите ОК.

5. Слово <<extends>> появится на линии данной связи.

Добавить описания к вариантам использования

1. Выделите в броузере вариант использования Запись на элективный курс

2. В окне документации введите следующее описание к этому варианту использования: Этот вариант использования дает студенту возможность записаться на элективный курс.

3. С помощью окна документации введите описания ко всем остальным вариантам использования.

Добавить описания к действующему лицу

1. Выделите в броузере действующее лицо Преподаватель

2. В окне документации введите для этого действующего лица следующее описание: Преподаватель – человек, осуществляющий обучение студентов

3. С помощью окна документации введите описания к оставшимся действующим лицам.

4. Анализ требования в Requisite Pro

Создание требований в RequisitePro из модели прецедентов Rose

Цель

Как известно, средства Rational могут быть использованы в большинстве софтверныхпроектовпрограммных системв компаниях с собственными стандартами и подходами к созданию программного обеспечения.

Согласно RUP модель прецедентов является стержнем, который связывает воедино многие артефакты, возникающие в большинстве современных софтверных проектов: планы работ, архитектура системы, программные коды, скриптысценариидляавтоматизированного тестирования и т.д. Также в этот список можно добавить и функциональные требования на

создаваемую систему, речь о которых пойдет ниже. Интегрированное использование Rose и RequisitePro позволяет решить две очень важные задачи:

• облегчить процесс создания начального списка функциональных требований из модели прецедентов;

• облегчить процесс синхронизации функциональных требований с прецедентами модели прецедентов. Создание начального списка функциональных требований из модели прецедентов разбивается на следующие подзадачи:

• создание необходимых групп (типов) требований и типов документов в проекте RequisitePro;

• интеграция модели Rose и проекта RequisitePro;

• экспорт прецедентов из модели Rose в группу "Прецеденты" проекта RequisitePro;

• создание требований в группе "Функциональные требования" проекта RequisitePro на основании сценариев прецедентов;

• создание трассировочной матрицы для синхронизации требований в группе "Прецеденты" и "Функциональные требования".

Создание необходимых типов требований и типов документов

Прежде всего, в проекте RequisitePro необходимо создать тип требований, который будет объединять прецеденты, экспортированные из модели прецедентов Rose. Преимущества дополнительного хранения этих прецедентов в виде требований RequisitePro:

• возможность создания дополнительных метрик для прецедентов (например, дата ожидаемой реализации, приоритет, исполнитель и т.д.);

• возможность сортировки и фильтрации прецедентов по названиям или атрибутам;

• возможность связывания прецедентов с требованиями различных типов (с требованиями планирования, тестирования, нефункциональными и т.д.);

• сохранение истории изменений прецедентов;

• ведение дискуссий по отдельным прецедентам или по их группам.

Для создания указанного типа требований на дереве узлов открытого проекта RequisitePro необходимо активизировать самый верхний узел и в его контекстном меню выбрать пункт "Properties" (рис. 1).

В открывшемся окне "Project Properties" следует активизировать страницу "Requirement Types". Она позволяет добавить в проект новые типы требований. После добавления типов требований "Прецеденты" и "Функциональные требования" окно "Project Properties" примет вид согласно

На страничке "Attributes" окна "Project Properties" следует определить необходимые атрибуты для сформированных типов требований. Хотя, конечно же, можно использовать и те атрибуты, которые создаются по умолчанию для новых типов требований.

В нашем случае для каждого созданного типа требований удаляем существующие атрибуты и создаем следующие:

"Приоритет" (список значений; возможные значения: "Высочайший", "Высокий", "Средний", "Низкий") - приоритет реализации функционального требования или прецедента;

"Состояние" (список значений; возможные значения: "Предложено", "Одобрено", "Реализовано", "Протестировано") - состояние, в котором на данный момент находится процесс реализации функционального требования или прецедента;

"Трудность" (список значений; возможные значения: "Высокая", "Средняя", "Низкая") – оценочная сложность реализации функционального требования или прецедента;

"Ответственный" (строка) - указывается фамилия ответственного за реализацию функционального требования или прецедента.

Совместное использование Rose и RequisitePro позволяет связывать документы проекта RequisitePro с любыми прецедентами модели Rose. Процесс интеграции модели Rose с проектом RequisitePro требует предварительного создания хотя бы одного типа документа, который будет связан с требованиями типа "Прецеденты". Сформируем некоторый обобщенный тип документов "Общие документы"

Для использования Rose совместно с RequisitePro необходимо, чтобы в первом был активизирован соответствующий "Add-In". Для этого необходимо выбрать пункт меню "Add-Ins/Add-In Manager...". Это приводит к появлению окна "Add-In Manager". Здесь следует проконтролировать, чтобы был активен пункт "RequisitePro". В результате появляются дополнительные пункты главного и различных контекстных меню, позволяющих работать с RequisitePro из Rose.

Теперь следует связать текущий файл модели Rose с проектом RequisitePro. Для этого требуется выбрать пункт меню "Tools/Rational RequisitePro/Associate Model To Project". В открывшемся окне "Associate Model To RequisitePro Project" следует указать проект RequisitePro, в который будут экспортированы прецеденты Кроме того, необходимо указать тип требований и тип документов по умолчанию, с которыми автоматически будут связаны экспортированные прецеденты. В дальнейшем указанные типы могут быть заменены на любые другие с помощью этого же окна.

Нажатие кнопки <OK> приведет к интеграции модели Rose и проекта RequisitePro. Экспорт прецедентов из Rose в RequisitePro Рассмотрим экспорт прецедента на простейшем примере. Пусть имеется некоторая модель прецедентов.

Для экспорта прецедентов следует для каждого из них выбрать пункт контекстного меню "Requirement Properties/New...". При этом появляется форма добавления требований "Requirement Properties..." из RequisitePro.

Данная форма позволяет установить значения дополнительных атрибутов прецедентов (страничка "Attributes"), создать связи с существующими требованиями любых типов ("Traceability") и сформировать иерархию прецедентов ("Hierarchy"). Нажатие <OK> приведет к физическому созданию требования типа "Прецеденты" в базе RequisitePro. Таким образом экспортируем все прецеденты в проект RequisitePro. Созданные требования в

RequisitePro следует перенести в папку, предназначенную для их хранения. После этого требуется создать матрицу атрибутов (Attribute Matrix") для работы с ними.

Функциональные требования создаются аналитиком на основе анализа сценариев прецедентов в модели Rose. Как и прежде, рассмотрим этот процесс на кратком примере.

Для добавления функциональных требований создадим матрицу атрибутов (Attribute Matrix) "Функциональные требования". После внимательной проработки указанного сценария прецедента сформируем список характеристик (Feachures), используя эту матрицу.

На приведенном выше рисунке показан внешний вид проекта RequisitePro после создания необходимых требований и характеристик. Для удобства работы требования и характеристики находятся в своих папках. Нами определены следующие требования для прецедента Запись на элективный курс.

Создание трассировочной матрицы и отслеживание изменений в модели прецедентов

Далее следует позаботиться об удобстве отслеживания характеристик в функциональных требованиях после возникновения изменений в модели прецедентов. Для этого в папке "Трассировочные матрицы" создадим трассировочную матрицу (Traceability Matrix) "Требования -Характеристики". С помощью нее будет легко определить, какие характеристики необходимо просмотреть на предмет возможных изменений, если изменились любые прецеденты модели прецедентов.

Перечеркнутый значок трассировки показывает, что нам необходимо внимательно просмотреть требование запись на элективный курс и все его дочерние требования. Очевидно, что после изменения названия прецедента, как минимум, требуется изменить формулировку родительского требования.

Для разработки тезауруса предложена иерархическая система понятий, применяемых в образовательном процессе.

Для создания термов в глоссарии в проекте Requisite Pro необходимо открыть в этом проекте вордовский файл *Glossary*:

На странице 4 этого документа (в разделе Glossary) нужно выбрать пункт 2.3 <aGroupOfTerms>. Перед знакосочетанием <aGroupOfTerms> отработатьEnteru вписать термины, которые мы собираемся включить в словарь проекта. Выделяем вписанный термин и отрабатываем на панели управления окнаMSWordиконку “RequisitePro”, с помощью модуляNewwizardотправляем наши термины в проект:

В процессе анализа и проектирования информационной системы необходимо поддерживать словарь общеупотребительных терминов и определений, которые собраны в глоссарии моделируемого процесса (Business Glossary). Необходимо также документировать процесс и выявлять правила ведения процесса.

Заключение

Совместное использование Rose и RequisitePro дает большое преимущество в поддержании модели прецедентов и списка функциональных

требований в актуальном состоянии. Любые изменения в модели прецедентов достаточно легко отображаются на функциональных требованиях проекта RequisitePro.