**1. Особенности CASE-технологии. История развития CASE-средств.**

На пути к достижению комплексного подхода при разработке программных средств (ПС) широкое применение получили CASE-средства (Computer Aided Software Engineering), обеспечивающие поддержку многочисленных технологий проектирования информационных систем, охватывая всевозможные средства автоматизации и весь жизненный цикл программного обеспечения (ПО). Диапазон CASE-средств очень велик, и сегодня практически каждое из них располагает мощной инструментальной базой.

Основная цель CASE-подхода – разделить и максимально автоматизировать все этапы разработки ПС. Большинство CASE-средств основано на парадигме методология / метод / нотация / средство. Методология определяет шаги работы и их последовательность, а также правила распределения и назначения методов. Метод – это систематическая процедура генерации описаний компонентов ПО. Нотация предназначена для описания структур данных, порождающих систем и метасистем. Средства – это инструментарий для поддержки методов на основе принятой нотации.

Основные преимущества применения CASE-средств:

- улучшение качества ПО за счет автоматического контроля проекта;

- возможность быстрого создания прототипа будущей системы, что позволяет уже на ранних стадиях разработки оценить результат;

- ускорение процессов проектирования и программирования;

- освобождение разработчиков от выполнения рутинных операций;

- возможность повторного использования ранее созданных компонентов.

Основной причиной, по которой возникла необходимость в появлении систем автоматизированного проектирования в области информационных технологий, явился дисбаланс между производительностями труда в сфере производства и в сфере обработки информации. Причем разница была не в пользу последней.

Изначально проблему пытались решить переводом людей из одной области труда в другую, однако это не помогло решить проблему. В связи с этим получили распространение CASE-средства. Комплексный характер CASE-средств заключается в том, что в них выполняются и сложные вычисления, и обработка большого объема данных. Это определяет направление дальнейшего развития CASE-средств как интегрированных интеллектуальных систем.

ТЕМА 1

**2. Системная модель CASE-средств.**

Любая техническая система, включая CASE-средство, может быть представлена следующим набором характеристик:

S={Ind, P, Atr, Inp, Out, Str}, где

Ind – обозначение и наименование системы;

Обозначение и наименование системы. Каждая коммерческая система должна иметь зарегистрированный товарный знак, который в совокупности с обозначением версии и модификации системы представляет собой обозначение системы. Наименование включает в себя ее функциональное описание.

P – цели системы;

Цели системы достигаются за счет ее технических функций, которые характеризуют способность преобразовывать входную информацию в выходную. В качестве целей системы чаще всего выступают такие характеристики, как: трудоемкость, себестоимость, длительность цикла процесса, качество продукта.

Atr – общесистемные характеристики;

Общесистемные характеристики участвуют в классификации CASEсредств по следующим признакам: - прикладная область объектов проектирования, - сложность проектируемых объектов, - уровень автоматизации, - комплексность автоматизации проектирования, - возможность работы в сетевом режиме и в Internet.

Inp – входы системы;

Out – выходы системы;

Входы и выходы системы зависят от ее функционального назначения и описываются в техническом задании на разработку.

Str – структура системы;

Структура системы включает в себя ее функциональные составные части и связи между ними, а также зависит от комплексности CASE-средств.

Str={E, R}, где

E – компоненты системы,

R – связи между компонентами.

ТЕМА 1

**3. Критерии развития CASE-средств.**

Каждая техническая система, в том числе и CASE-система, характеризуется группой свойств, которые определяют меру совершенства и прогрессивности данной системы. Такие свойства называют критериями развития.

Критерии, важные для CASE-средств, можно разделить на четыре группы: функциональные, технологические, экономические, эргономические.

Функциональный критерий рассматривается как интегральный показатель, зависящий от ряда частных функциональных критериев: скорости обработки информации, интенсивности обработки информации, степени автоматизации труда, непрерывности процесса проектирования.

Технологические критерии связаны с настоящими и будущими затратами на стадиях создания, развития и адаптации системы к решению конкретных задач.

Экономический критерий CASE-средства служит для комплексного стоимостного учета положительного эффекта от автоматизации проектирования и основных затрат.

Критерий эргономичности CASE-средства равен отношению реализуемой эффективности системы к максимально возможной эффективности этой системы. Он представляет собой зависящую от времени функцию, стремящуюся к 1. Данный критерий можно трактовать как КПД человека в системе.

ТЕМА 1

**4. Понятие проекта. Масштаб проекта. Общие принципы управления проектом.**

Проект – это уникальный комплекс взаимосвязанных мероприятий, направленных на достижение конкретной цели при определенных требованиях к срокам, бюджету и характеристикам ожидаемых результатов.

В этом определении следует обратить внимание на следующее:

1) Каждый проект характеризуется конкретной целью, ради которой он затевается.

2) Каждый проект в чем-то уникален.

3) Любой проект ограничен по времени “жизни”.

4) Каждый проект характеризуется конкретными ресурсами, выделенными на его выполнение.

Масштаб проекта – это совокупность цели проекта и планируемых для ее достижения затрат времени и средств. Другими словами, это своеобразное трехмерное пространство (цель-время-деньги), в котором живут участники проекта и сам проект. Опираясь на введенное понятие масштаба, можно сказать, что управление проектом направлено на сохранение его исходного масштаба – содержания и границ.

Управление проектом – это процесс планирования, организации и контроля состояния задач и ресурсов проекта, направленный на своевременное достижение цели проекта.

В ходе управления любым проектам должно быть обеспечено решение следующих задач:

- соблюдение директивных сроков завершения проекта;

- рациональное распределение материальных ресурсов и исполнителей между задачами проекта, а также во времени;

Целенаправленное управление проектом предназначено для пропорционального распределения ресурсов между работами по созданию ПС на протяжении всего цикла проектирования вплоть до внедрения системы в серийное производство или ее массового использования.

ТЕМА 2

**5. Три составляющие программного проекта: система обозначений, процесс и инструмент. Их роль и значение для проекта.**

Успех любого программного проекта зависит от трех составляющих: системы обозначений (нотации, языка), процесса и инструмента. Одинаково нужны все три составляющие.

Система обозначений важна в любой модели. Это связующее звено между всеми составляющими процесса разработки проекта. Примером полной и надежной системы обозначений может служить унифицированный язык моделирования (Unified Modeling Language – UML). С его помощью можно описывать модели на любом этапе разработки ПС: от анализа требований до проектирования и реализации.

Современные проекты должны разрабатываться с помощью эффективных средств в соответствии со строгим графиком и обеспечивать возможность внесения изменений и адаптации к конкретным условиям и требованиям. ЖЦ проекта должен быть управляемым, что позволит гарантировать его непременное завершение. Примером процесса, подходящего для современных IT-проектов, является унифицированный процесс. Его методология основана на языке UML и находит поддержку в современных инструментах.

Ни одна из современных технологий разработки ПС не обходится без какого-либо инструмента. Сегодня на рынке представлен достаточно широкий спектр инструментов, использующих нотацию UML и объектноориентированный подход к разработке ПО.

Унифицированный процесс – это больше, чем единичный процесс. UP – это обобщенный каркас процесса, который может быть специализирован для широкого круга программных систем, различных областей применений, уровней компетенции и размеров проекта. Его основными принципами являются:

– итерационный и инкрементный подход к созданию ПО;

– управление вариантами использования;

– построение системы на базе архитектуры ПО.

ТЕМА 2

**6. Типы и особенности современных программных проектов.**

Проект для постоянного заказчика. Ситуация, при которой команда разработчиков в течение длительного времени обслуживает единственного заказчика.

Продукт под заказ. Ситуация, при которой команда разработчиков находит стороннего заказчика и договаривается с ним о разработки программного продукта, призванного решить те или иные проблемы заказчика.

Тиражируемый продукт. Ситуация, при которой команда разработчиков либо вообще не имеет конкретных заказчиков, либо довольно большое количество заказчиков желают иметь один и тот же продукт.

Аутсорсинг. Это одна из наиболее новых моделей производства ПО. Суть ее состоит в том, что между крупной (обычно) фирмой по производству ПО и другой иностранной фирмой заключается договор о субподряде.

ТЕМА 2

**7. Задачи и категории современных методологий создания программных проектов.**

Методологии можно условно разбить на три категории: тяжелые, средние и легкие. Упрощенно каждая из них предназначена для работы в условиях больших, малых и средних проектов соответственно.

Тяжелая категория методологий появилась раньше других и служит неотъемлемой частью моделей качества ПО. Тяжелые методологии охватывают все аспекты деятельности компании, производящей ПО, – от управления требованиями и планирования процесса до регламентирования отношений с заказчиком. Все методологии данной категории нетерпимы к изменениям и рассматривают людей как ресурс. К этой категории относятся ISO9001, CMM, SPICE.

Средняя категория методологий включает в себя так называемые универсальные процессы. Наиболее популярным представителем этой категории является методология рационального унифицированного процесса – Rational Unified Process (RUP). Основная характеристика этой методологии – масштабируемость, т.е. процесс может быть настроен на работу как в малой команде над небольшим проектом, так и в большой команде над большим проектом.

Легкая категория методологий – это некоторая совокупность методов и практик, применявшихся небольшими командами разработчиков в небольших проектах. Все процессы данной категории предусматривают итерационный ЖЦ разработки ПО. Идея легких методологий – обеспечение максимальной скорости и качества разработки ПО при минимуме ограничений. Во всех легких методологиях предусмотрен лишь необходимый минимум документов. Примеры легких методологий - SCRUM, ICONIX, XP, Crystal Clear.

ТЕМА 2

**8. Взаимосвязь между методологией, размером задачи и командой разработчиков.**

Главными задачами современной методологии и основанного на ней процесса являются следующие:

- облегчить процедуру введения новых людей в курс процесса производства; - обеспечить взаимозаменяемость людей;

- распределить ответственности;

- демонстрировать видимый прозрачный процесс;

- создать учебную базу для сотрудников.

Для обеспечения выполнения данных задач выбирается тот или иной тип методологии в зависимости от размера задачи и команды разработчиков.

Существуют тяжелые, средние и легкие методологии.

Тяжелые методологии предназначены для больших проектов(сложных и больших задач соответственно). Все методологии данной категории нетерпимы к изменениям и рассматривают людей как ресурс, то есть методологии данной категории предназначены для больших команд.

Средние методологии обеспечивают гибкость. Основная характеристика этой методологии – масштабируемость, т.е. процесс может быть настроен на работу как в малой команде над небольшим проектом, так и в большой команде над большим проектом.

Легкие методологии обычно применяются небольшими командами разработчиков в небольших проектах. Все процессы данной категории предусматривают итерационный ЖЦ разработки ПО. Идея легких методологий – обеспечение максимальной скорости и качества разработки ПО при минимуме ограничений.

ТЕМА 2

**9. Целесообразность использования различных методологий для различных типов программных проектов.**

Методологии можно условно разбить на три категории: тяжелые, легкие и средние.

Проект для постоянного заказчика. Самый благоприятный тип проекта для внедрения легких методологий, поскольку заказчик всегда доступен и не предъявляет сверхтребований к ПО. Однако необходимо учитывать количество разработчиков и степень их распределенности. Как правило, у таких команд не бывает необходимости в сертификации.

Продукт под заказ. Самый уязвимый тип проекта. Фирма целиком зависит от количества заключенных договоров. Постоянно идет поиск новых заказчиков. В таких условиях, конечно же, желательно наличие сертификата ISO. Сертификацию целесообразно проводить лишь при достижении определенной численности персонала, которой будет достаточно для внедрения тяжелой или средней технологии. Алтернативный вариант – одна из легких методологий.

Тиражируемый продукт. Самый устойчивый тип проекта. Выпуск такого проекта всегда характеризуется более низкими затратами на его производство по сравнению с выпуском единичных экземпляров. В данных условиях невозможно использование легкой методологии в чистом виде, т.к. нет возможности постоянно работать с заказчиком. В этом случае все зависит от способа управления командой, тактических и стратегических целей.

Аутсорсинг. Данный вид проектов характеризуется распределенной структурой и начальными предпосылками к утяжелению процесса, поскольку общение с заказчиком происходит в виде документов установленного образца. Если сторона фирмы-заказчика предоставляет команде свой технологический процесс, то у нее нет свободы выбора. В противном случае стоит остановить свой выбор на каком-либо из процессов средней тяжести.

ТЕМА 2

**10. Унифицированный процесс разработки программных средств.**

Прежде всего унифицированный процесс – это процесс разработки ПО, т.е. это множество различных видов деятельности, необходимых для преобразования требований пользователей в программную систему.

Однако унифицированный процесс – это больше, чем единичный процесс. UP – это обобщенный каркас процесса, который может быть специализирован для широкого круга программных систем, различных областей применений, уровней компетенции и размеров проекта. Его основными принципами являются:

– итерационный и инкрементный подход к созданию ПО;

– управление вариантами использования;

– построение системы на базе архитектуры ПО.

Первый принцип является определяющим. В соответствии с ним разработка системы выполняется в виде нескольких краткосрочных минипроектов фиксированной длительности (от 2 до 6 недель), называемых итерациями. Каждая итерация включает в себя свои собственные этапы анализа требований, проектирования, реализации, тестирования, интеграции и завершается созданием работающей системы. Итерационный цикл основывается на постоянном расширении и дополнении системы в процессе нескольких итераций с периодической обратной связью и адаптацией добавляемых модулей к существующему ядру системы. Система постоянно разрастается, поэтому такой подход называют итерационным и инкрементным.

Вариант использования – это часть функциональности системы, необходимая для получения пользователем значимого для него, ощутимого и измеримого результата. Варианты использования обеспечивают функциональные требования. Все варианты использования в совокупности составляют модель вариантов использования, которая описывает полную функциональность системы. Варианты использования в UP – это не только средство описания требований к системе. Они направляют далее весь процесс ее разработки. Основываясь на модели вариантов использования, разработчики создают все последующие модели. Поскольку варианты использования управляют процессом, они не выделяются изолированно, а разрабатываются совместно с созданием архитектуры системы. Следовательно, варианты использования управляют архитектурой системы, которая, в свою очередь, оказывает влияние на их выбор. И архитектура системы, и варианты использования развиваются по мере хода жизненного цикла.

Созданная архитектура является основой всей дальнейшей разработки. В будущем неизбежны незначительные изменения в деталях архитектуры, однако серьезные изменения маловероятны.

ТЕМА 2

**11. Основные и дополнительные элементы объектно-ориентированного подхода.**

Главное отличие объектного подхода от структурного заключается в объектной декомпозиции системы. При этом статическая структура системы описывается в терминах объектов и связей между ними, а поведение системы – в терминах обмена сообщениями между объектами. Концептуальной основой объектного подхода является объектная модель, главными элементами которой считаются: абстрагирование, инкапсуляция, модульность, иерархия. Кроме того, имеются три дополнительных элемента: типизация, параллелизм, устойчивость. Рассмотрим все элементы с точки зрения создания ПО с помощью CASE средств.

Абстрагирование – это выделение существенных характеристик некоторого объекта, которые отличают его от всех других видов объектов и, таким образом, четко определяют его концептуальные границы относительно дальнейшего рассмотрения и анализа. Абстрагирование позволяет отделить существенные особенности поведения объекта от деталей их реализации. Выбор правильного набора абстракций – это главная задача объектно-ориентированного подхода.

Инкапсуляция – это процесс отделения друг от друга элементов объекта, определяющих его устройство и поведение. Инкапсуляция служит для того, чтобы разделить интерфейс и внутреннюю реализацию объекта.

Модульность – это свойство системы, связанное с ее декомпозицией на ряд внутренне сильно связанных, но слабо связанных между собой модулей. Инкапсуляция и модульность создают барьеры между абстракциями.

Иерархия – это ранжированная или упорядоченная система абстракций. Иерархия по номенклатуре – это структура классов, а иерархия по составу – это структура объектов.

Типизация – это ограничение, накладываемое на класс объектов и препятствующее взаимозаменяемости различных классов.

Параллелизм – это свойство объектов находиться в активном или пассивном состоянии и различать между собой активные и пассивные объекты.

Устойчивость – это свойство объекта существовать во времени и в пространстве вне зависимости от процесса, породившего данный объект.

Еще два важных понятия объектно-ориентированного подхода – это полиморфизм и наследование. Полиморфизм можно интерпретировать, как способность класса принадлежать более чем одному типу. Наследование означает построение новых классов, на основе существующих с возможностью добавления 2 или переопределения данных и методов.

ТЕМА 3

**12. История появления, особенности и назначение унифицированного языка моделирования UML.**

Язык моделирования UML (Unified Modeling Language) – это нотация, которая используется методом для описания проектов. Нотация представляет собой совокупность графических объектов, которые используются в моделях. Она является синтаксисом языка моделирования. Процесс – это описание шагов, которые необходимо выполнить при разработке проекта.

Основная идея UML – возможность моделировать ПО и другие системы как наборы взаимодействующих объектов.

Семантика языка UML представляет собой некоторую метамодель, которая определяет абстрактный синтаксис и семантику понятий объектного моделирования на языке UML. Семантика определяется для двух видов объектных моделей: структурных моделей и моделей поведения. Структурные модели, известные также как статические модели, описывают структуру сущностей или компонентов некоторой системы, включая их классы, интерфейсы, атрибуты и отношения. Модели поведения, называемые иногда динамическими моделями, описывают поведение или функционирование объектов системы, включая их методы, взаимодействие и сотрудничество между ними, а также процесс изменения состояний отдельных компонентов и системы в целом.

Авторами UML являются основоположники объектно-ориентированного подхода: Буч, Рамбо, Якобсон.

Создание UML началось в 1994 году, а в 1995 появилась первая спецификация языка. В 2000 году появилась версия UML 1.4 как существенное расширение UML, достигнутое добавлением семантики действий. Это было серьезным достижением, поскольку сделало спецификацию UML полной в вычислительном отношении, что обеспечило возможность создавать UML-модели исполняемыми. В 2005 году была завершена спецификация UML 2.0.

ТЕМА 3

**13. Требования к программному обеспечению. Первичные и детальные требования. Функциональные и нефункциональные требования.**

Требования к программному обеспечению — совокупность запросов/утверждений относительно атрибутов, свойств или качеств [программной системы](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0), подлежащей реализации. Создаются в процессе проработки (анализа и синтеза) задания на разработку/модернизацию [программного обеспечения](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) (ПО).

Первичные требования документируют желания и потребности заказчика и пишутся на языке, понятном заказчику. Детальные требования документируют требования в специальной, структурированной форме, они детализированы по отношению к первичным требованиям.

Требования могут выражаться в виде текстовых утверждений и графических моделей.

По характеру требования делятся на функциональные и нефункциональные.

* Функциональный характер — требования к поведению системы
  + Бизнес-требования
  + Пользовательские требования
  + Функциональные требования
* Нефункциональный характер (ограничения архитектуры/реализации/эксплуатации) — требования к характеру поведения системы
  + Бизнес-правила — определяют ограничения, проистекающие из предметной области и свойств автоматизируемого объекта (предприятия)
  + Системные требования и ограничения — определения элементарных операций, которые должна иметь система, а также различных условий, которым она может удовлетворять. К системным требованиям и ограничениям относятся:
    - Ограничения на программные интерфейсы, в том числе к внешним системам
    - Требования к атрибутам качества
    - Требования к применяемому оборудованию и ПО
  + Требования к документированию
  + Требования к дизайну и юзабилити
  + Требования к безопасности и надежности
  + Требования к показателям назначения (производительность, устойчивость к сбоям и т. п.)
  + Требования к эксплуатации и персоналу
  + Прочие требования и ограничения (внешние воздействия, мобильность, автономность и т. п.).

НЕТ В PDF ИЗ СЭО!!!! ЭТА ИНФА ЗАГУГЛЕНА

**14. Назначение, особенности и построение диаграммы Use Case.**

Диаграмма Use Case позволяет создать список операций, которые выполняет система. Часто Use Case называют диаграммой функций, так как на основе набора таких диаграмм создается список требований к системе и определяется множество выполняемых ею функций. Данный тип диаграмм используется при описании бизнес-процессов автоматизируемой предметной области, определении требований к будущей программной системе.

Основные компоненты диаграммы Use Case - Use Case (варианты использования) и - Actor (действующие лица). Вместе они определяют сферу применения создаваемой системы. При этом первые описывают все то, что происходит внутри системы, а вторые – то, что происходит снаружи. Кроме этого, на диаграмме Use Case и Actor объединяются при помощи соответствующих связей.

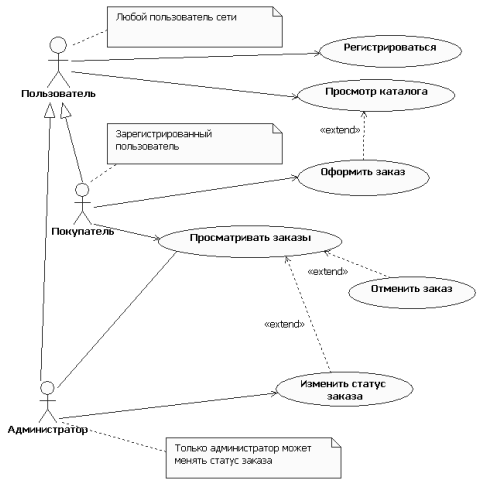
Для одной системы может создаваться несколько диаграмм Use Case.Рекомендуется придерживаться следующих правил, создавая диаграмму Use Case:

1. Не моделировать связи между Actors, так как по определению они находятся вне сферы действия системы. Следовательно, связи между ними также не относятся к ее компетенции. s

2. Не соединять непосредственно два Use Case, поскольку данная диаграмма только перечисляет варианты использования, доступные системе, а не указывает порядок их выполнения.

3. Каждый вариант использования инициируется действующим лицом, поэтому должна быть связь, начинающаяся на действующем лице и заканчивающаяся на варианте использования.

ТЕМА 4



Пример use case

**15. Назначение, особенности и построение диаграммы Deployment.**

Диаграмма Deployment предназначена для анализа аппаратной части системы. При помощи Deployment проектировщик может провести анализ необходимой аппаратной конфигурации, на которой будут работать процессы системы, и описать их взаимодействие между собой. Этот тип диаграмм также позволяет анализировать взаимодействие процессов, работающих на разных компьютерах сети.

С точки зрения используемых инструментов Deployment является самой простой диаграммой, так как в ней присутствуют только два вида основных значков.

Processor (процессор) – это устройство, способное выполнять программы, и

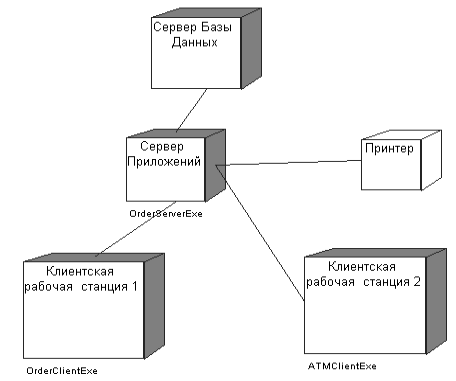
Device (устройство) – значок, отражающий на диаграмме все другие типы устройств. Процессором считается любая техническая система, способная обрабатывать данные. В эту категорию попадают серверы, рабочие станции и другие устройства, содержащие физические процессоры. Устройством на диаграмме обозначается аппаратура, не обладающая вычислительной мощностью, как, например, терминалы ввода/вывода, принтеры и сканеры. С помощью раздела спецификации вводится описание аппаратной части системы: стереотипы и характеристики. Стереотипы применяются для классификации процессоров и устройств, характеристики – это их физические особенности, например, скорость процессора и объем памяти.

Кроме процессоров и устройств на Deployment могут отображаться процессы, правила работы с которыми аналогичные. Процессом считается поток информации, например, исполняемый файл, выполняющийся на процессоре.

На диаграмме Deployment может быть связана любая пара элементов. Чаще всего связи отражают физическую сеть соединений между узлами. Кроме того, это может быть ссылка Интернета, связывающая два узла.

Для распределенных систем, какими часто бывают Web-приложения, этот тип диаграмм очень важен, поскольку именно здесь определяется, на каком сервере сети будет работать конкретный компонент или Web-служба, и с какими другими сетевыми устройствами будет осуществляться взаимодействие.

ТЕМА 5

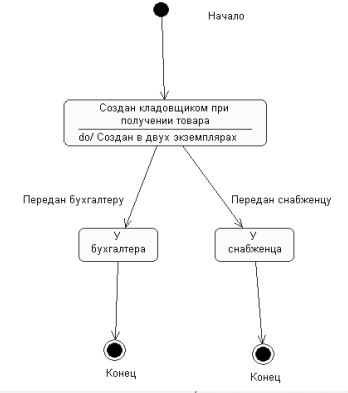


Пример deployment

**16. Назначение, особенности и построение диаграммы Statechart.**

Диаграмма состояний Statechart предназначена для изучения состояний объектов и условий переходов между ними. Модель состояний позволяет представить поведение объекта при получении им сообщений и взаимодействии с другими объектами.

Рассмотрим диаграмму Statechart, приведенную на рисунке. Первым шагом на пути построения является создание точки начала работы с помощью инструмента Start State. Обычно следующим состоянием системы после начала ее работы является ожидание наступления событий. После чего в нашем случае создается состояние (State), которое соединяется стрелкой State Transition с начальной точкой.



Для повышения информативности состояниям и событиям, переводящим объекты из одних состояний в другие, присваиваются имена. С состоянием объекта могут быть связаны события и действия. Разница между ними заключается в том, что действие осуществляется самим классом, для которого строится диаграмма Statechart, то есть вызывается метод данного класса, а посылка сообщения направлена на объект другого класса, чей метод вызывается при помощи сообщения. Для того чтобы получить доступ к действиям и событиям, нужно через контекстное меню объекта перейти в окно его спецификации, где выбрать вкладку Actions, а в ней – кнопку Insert. На рисунке состояние «Создан кладовщиком при получении товара» связано с действием «Создан в двух экземплярах». Построение диаграммы завершается добавлением на нее значка – End State, который отражает окончание работы. Направление перехода может быть установлено только в End State. Однако нет ограничений на количество самих элементов и переходов в них. ТЕМА 6

**17. Назначение, особенности и построение диаграммы Activity.**

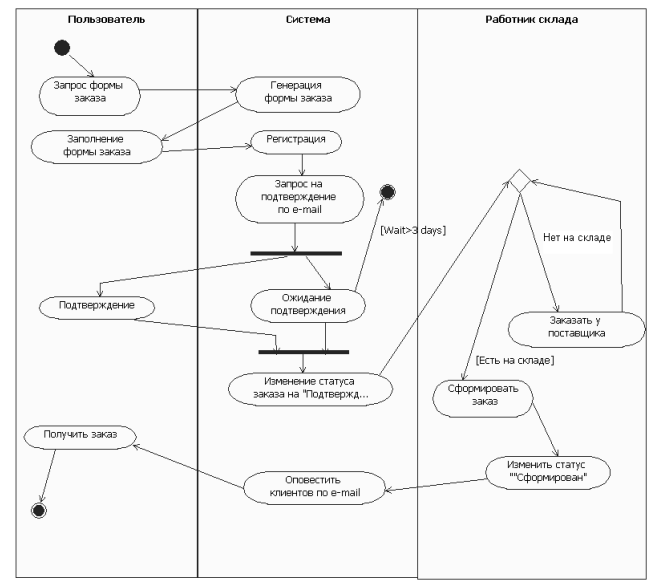
Диаграмм Activity является разновидностью диаграмм состояний. Главное различие между Activity и Statechart заключается в том, что первая характеризует действия, а вторая – статичные состояния. При этом Activity больше подходит для моделирования последовательности действия, а Statechart – для моделирования дискретных состояний объекта.

Для построения диаграммы используются следующие компоненты:

Значок обозначает выполнение определенных действий в течение жизни объекта. В отличие от , обычно обозначающего ожидание какого-либо события, показывает непосредственное действие. Деятельности соединяются на диаграмме значком – State Transition (переход состояния). Кроме этого, показывает получение и обработку сообщения объектом. Переход состояния может происходить между Action-Action, State-State, State-Action, Action-State. Возможна установка нескольких переходов между двумя состояниями или действиями. Каждый такой переход уникален и показывает реакцию объекта на определенное сообщение. Поэтому нельзя создать несколько переходов между двумя состояниями с указанием одного и того же сообщения. Уникальным инструментом диаграммы Activity является значок – Swimlanes (плавательные дорожки), который позволяет моделировать последовательность действий различных объектов и связи между ними. С его помощью можно строить бизнес-процессы организации, отражая на диаграмме различные подразделения и объекты. Swimlanes помогают показать роли каждого участника бизнес-процесса. Для этого необходимо переместить соответствующие значки активности или состояний в определенную часть диаграммы, отделенную от остальных Swimlanes.

Для построения диаграмм Activity имеется еще несколько важных инструментов. Значок - Decision (решение), позволяет показать зависимость дальнейшей работы от внешних условий и решений. Этот инструмент аналогичен командам языка программирования if или case и может иметь больше двух выходов, но обычно используют выбор из двух переходов, определенных булевыми выражением. Значки и – Synchronizations (синхронизация) позволяют определить независимо выполняемые действия. При этом действия разделяются на несколько выполняемых независимо, и только по завершении всех действий объект продолжает работу.

ТЕМА 6



Пример activity

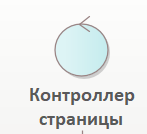
**18. Назначение, особенности и построение диаграммы Sequence.**

После того как в системе изучено поведение каждого объекта, необходимо точно представить взаимодействие этих объектов между собой, определить клиентов, серверы и порядок обмена сообщениями между ними. Диаграмма Sequence позволяет получить отражение процесса обмена сообщениями во времени.

В течение работы системы объекты, являющиеся клиентами, посылают друг другу различные сообщения, а объекты-серверы обрабатывают их. В простейшем случае можно рассматривать сообщение как вызов метода какого-либо класса, в более сложных случаях сервер имеет обработчик очереди сообщений, и сообщения им обрабатываются асинхронно, т.е. сервер накапливает несколько сообщений в очереди, если не может обработать их сразу.

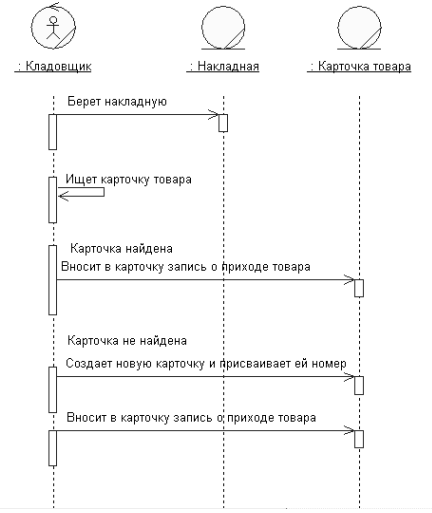
Основные компоненты диаграммы sequence:  
Значок – Object (объект) позволяет включить объект в диаграмму. Каждый из них является реализацией какого-нибудь класса, поэтому в объекте можно указать соответствующий ему класс. Для повышения наглядности в примере использованы стереотипы классов. Кладовщик представлен как business worker, Накладная – как business entity, Карточка товара – как business entity. Значок – Message (сообщение) предназначен для передачи сообщения от одного объекта к другому. Классы должны позволять отправку или прием сообщений. Инструмент – Message to self (сообщение самому себе) показывает, что отправитель сообщения является одновременно и его получателем. В этом случае объект выполняет функции и сервера, и клиента.

Также есть actor, entity и control, actor обозначает действующее лицо, entity - некоторую сущность, а control обозначает некоторый контроллер для доступа к сущности, например:



actor entity control

ТЕМА 6



Пример sequence

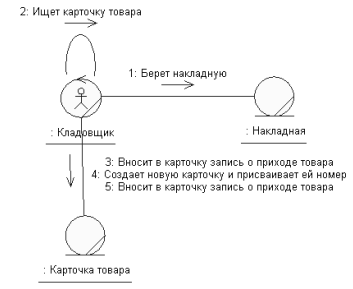
**19. Назначение, особенности и построение диаграммы Collaboration.**

Второй тип диаграмм взаимодействия – это Collaboration. Данная диаграмма отличается от Sequence тем, что здесь не акцентируется внимание на последовательности передачи сообщений, а отражается наличие взаимосвязей между клиентами и серверами вообще. Поскольку на Collaboration для демонстрации сообщений не применяется временная шкала, диаграмма получается более компактной и оптимально подходит для представления взаимодействий сразу всех объектов. Однако такое представление является мгновенным снимком системы в некотором состоянии, так как объекты создаются и уничтожаются на всем протяжении работы программы. В связи с этим появляются такие понятия, как время жизни и область видимости объектов.

Удобной возможностью работы в Rational Rose является то, что на основе Sequence-диаграммы можно создавать Collaboration и наоборот.

Для построения приведенной диаграммы Collaboration можно использовать ее основные инструменты. Значок Object позволяет создавать объекты, которые имеют состояния и поведение. Значок – Object Link (связь объекта) отражает взаимодействие объектов посредством показа их связей. При этом один из объектов может посылать сообщение другому. Значок – Link To Self показывает, что объект имеет обратную связь с самим собой. Значок – Link Message (передача сообщения) позволяет отразить связь, которая подразумевает обязательную передачу сообщения. У каждой связи Link Message есть соответствующие свойства, которые позволяют настроить область видимости для связанных объектов. Доступные в rational rose значения:

1. Unspecified – не определено, это значение присваивается по умолчанию.
2. Field – объект включен в другой объект.
3. Parameter – объект передается параметром в другой объект.
4. Local – объект локально определен в границах другого объекта.
5. Global – объект глобален по отношению к другому объекту.



Пример Collaboration. ТЕМА 7

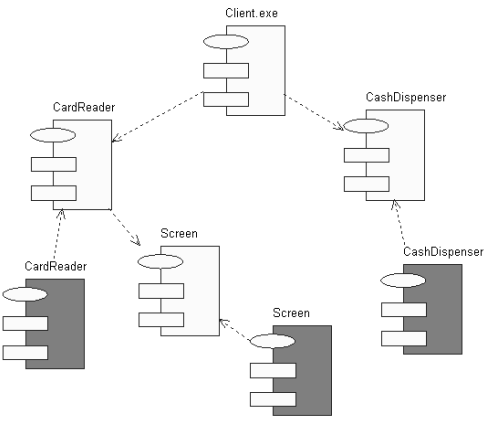
**20. Назначение, особенности и построение диаграммы Component.**

Диаграмма компонентов позволяет создать физическое отражение текущей модели. Component показывает организацию и взаимосвязи программных компонентов, представленных в файлах различных типов, а ее связи отражают зависимости одного компонента от другого. В текущей модели может быть создано несколько диаграмм компонентов для отражения пакетов, компонентов верхнего уровня или описания содержимого каждого пакета компонентов.

Для систем, состоящих из большого количества классов, целесообразно строить диаграмму компонентов, когда определены все связи классов и структура наследования. Но поскольку на всем протяжении проектирования системы, вплоть до выхода готового программного продукта в диаграммы будут вноситься изменения, оправдано создание Component для нескольких классов, чтобы получить практику работы с данным типом диаграмм.

Для построения приведенной диаграммы компонентов были использованы некоторые из ее инструментов. Значок Package specification позволяет отобразить определение пакета, а значок Package body выполняет описание тела пакета. Обычно эти инструменты связаны между собой. Здесь Package specification –заголовочный файл с расширением. h, а Package body – файл с расширением .cpp. С помощью значка dependency устанавливаются связи между компонентами. Этот тип связи показывает, что классы, содержащиеся в компоненте-клиенте, наследуются, содержат элементы, используют или каким-либо другим образом зависят от классов, которые экспортируются из компонента-сервера. Строка инструментов диаграммы компонентов содержит еще несколько элементов, позволяющих отражать программную реализацию системы. Значок - Component (компонент) на диаграмме представляет собой модуль программного обеспечения, такой как исходный код, двоичный файл, выполняемый файл, динамически подключаемые библиотеки. Компоненты могут использоваться для показа взаимосвязи модулей на этапе компиляции или выполнения программы, а также показывать, какие классы используются для создания определенных компонентов. Значок - Package (пакет) позволяет отобразить пакет, который объединяет группу компонентов в модели. Значок - Main program (главная программа) позволяет добавить в модель компонент, обозначающий главную программу. Значок - Subprogram body (тело подпрограммы) позволяет добавить в модель компонент, обозначающий тело подпрограммы. Значки и - Task specification / body (определение / тело задачи) позволяют отобразить независимые потоки в многопотоковой системе.

ТЕМА 8



Пример component

**21. Назначение, особенности и построение диаграммы Class, виды и особенности связей между классами на диаграммах.**

Диаграмма классов является основной для создания кода приложения. С ее помощью строится внутренняя структура системы. Обычно данная диаграмма строится для всех классов, становясь логической моделью системы. Кроме того, Rational Rose позволяет на основе Class diagram создавать исходный код приложения на любом языке программирования, который поддерживается генератором кода Rational Rose. С ее помощью возможно изменение свойств любого класса или его связей, при этом диаграммы или спецификации, связанные с изменяемым классом, будут автоматически обновлены.

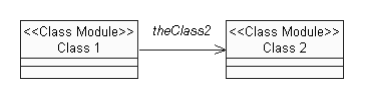
Диаграмма class состоит из компонента сlass diagram . В каждом таком компоненте задается имя класса, язык программирования, перечислены атрибуты и операции соответствующего класса. В rational rose для атрибутов есть возможность задать следующие типы хранения:

1. By Value – по значению.
2. By Reference – по ссылке.
3. Unspecified – не указано.

Кроме того, можно указать, что атрибут является Static (статическим) или Derived (производным).

В большинстве случаев классы взаимодействуют друг с другом, что отображается при помощи различного вида связей, влияющих на получаемый при генерации код. В диаграмме классов различают следующие виды связей:

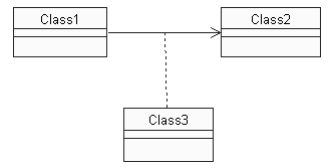
1. Unidirectional association (однонаправленная ассоциация) - один класс включается в другой как атрибут по ссылке или по значению



1. Dependency (зависимость) - один класс использует объекты другого



1. Association class (ассоциированный класс) - обозначает, что некоторый класс со своими атрибутами включается как элемент в два других, хотя при генерации кода это не отображается.



1. Generalization (наследование) - тип связи Generalization позволяет указать, что один класс является родительским по отношению к другому



1. Realization (реализация) - позволяет показать, что один класс является реализацией, т.е. создан на основе шаблона другого



**22. Понятие шаблонов проектирования и их классификация. Шаблоны в нотации языка UML. ТЕМА 10**

Очень часто при проектировании программных систем встречаются ситуации, когда решение определенных проблем даже на разных проектах имеют схожие черты. Это привело к появлению шаблонов(паттернов) проектирования.

Используется следующая общая классификация паттернов по категориям их применения: Архитектурные паттерны; Паттерны проектирования; Паттерны анализа; Паттерны тестирования; Паттерны реализации.

Архитектурные - множество заранее заранее определенных систем, у которых есть зоны ответственности, правила и принципе отношений между друг другом.

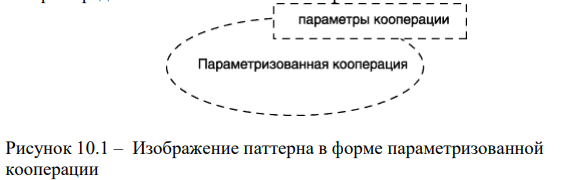
Проектирования - специальные схемы для уточнения структуры подсистем или компонентов программной системы и отношений между ними. Паттерны проектирования описывают общую структуру взаимодействия элементов программной системы. Известными являются паттерны, описанные в книге “банды четырех”.

Анализа - специальные схемы для представления общей организации процесса моделирования. Паттерны анализа относятся к одной или нескольким предметным областям. Наиболее известными паттернами этой группы являются паттерны бизнес-моделирования ARIS (Architecture of Integrated Information Systems), которые характеризуют абстрактный уровень представления бизнес-процессов.

Паттерны тестирования – специальные схемы для представления общей организации процесса тестирования программных систем. К этой категории паттернов относятся такие паттерны, как тестирование черного ящика, белого ящика, отдельных классов, системы.

Паттерны реализации – совокупность компонентов и других элементов реализации, используемых в структуре модели при написании программного кода. Эта категория паттернов делится на следующие подкатегории: паттерны организации программного кода, паттерны оптимизации программного кода, паттерны устойчивости кода, паттерны разработки графического интерфейса пользователя и др.

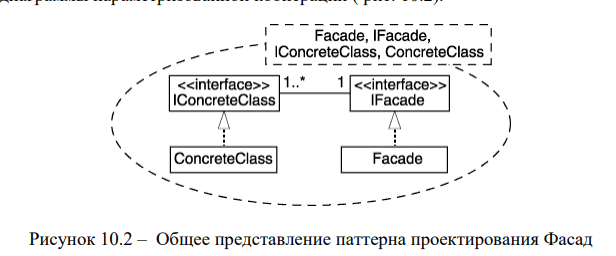
Паттерн проектирования в контексте языка UML представляет собой параметризованную кооперацию вместе с описанием базовых принципов ее использования. Изображение паттерна:



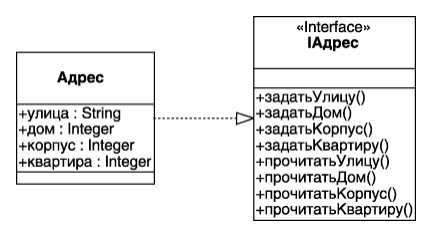
В последующем параметры паттерна могут быть заменены различными классами, чтобы получить реализацию паттерна в рамках конкретной кооперации. Эти параметры специфицируют используемые классы в форме ролей классов в рассматриваемой подсистеме. При связывании или реализации паттерна любая линия помечается именем параметра паттерна, которое является именем роли соответствующей ассоциации. В дополнение к диаграммам кооперации особенности реализации отдельных паттернов представляются с помощью диаграмм последовательности.

**23. Шаблон “Фасад” и его обозначение в нотации языка UML. ТЕМА 10**

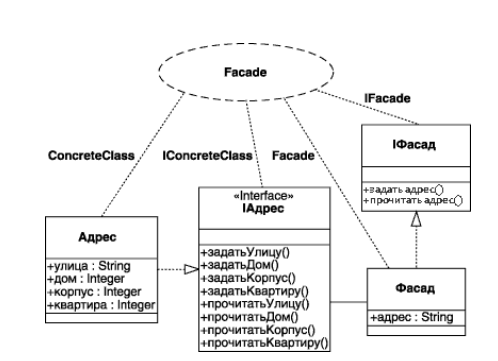
Паттерн Фасад предназначен для замены нескольких разнотипных интерфейсов доступа к определенной подсистеме некоторым унифицированным интерфейсом, что существенно упрощает использование рассматриваемой подсистемы.



Например, нам нужно устанавливать адрес и читать его. У нас есть интерфейс со всеми функциями: задать улицу, задать дом, задать корпус и тд. То есть для установки или чтения адреса, нам нужно в правильной последовательности вызвать все методы:



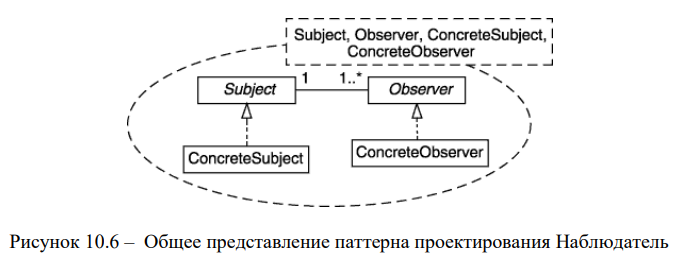
С паттерном фасад:



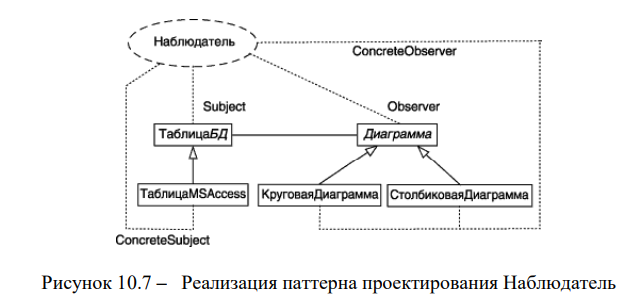
(Здесь параметр +адрес в классе Фасад должен быть типа IАдрес). Здесь мы вместо вызова целой кучи методов класса Адрес будем вызывать методы класса Фасад. А эти методы внутри себя будут отвечать за правильный вызов методов класса Адрес.

**24. Шаблон “Наблюдатель” и его обозначение в нотации языка UML. ТЕМА 10**

Паттерн Наблюдатель предназначен для контроля изменений состояния объекта и передачи информации об изменении этого состояния другим системам(объектам). Кооперация содержит 4 параметра. Пунктирная линия со стрелкой в форме треугольника служит для обозначения отношения обобщения классов:



Пример: отслеживание изменений в таблице БД и отображение изменений на диаграммах



В данном случае следят за субъектом ТаблицаMSAcess. Причем следить за ней может любое количество наблюдателей. И для этого ТаблицаБД отслеживает любые свои изменения и при их наличии сразу информирует абстрактного наблюдателя Диаграмма. Тот в свою очередь вызывает перерисовку диаграмм у определенных наблюдателей.

**25. Декомпозиция программной системы на модули. Принцип модульности. Оценка сложности программной системы через принцип модульности.**

Модуль - фрагмент программного текста, являющийся строительным блоком для физической структуры системы.

Модульность - свойство системы, которая может подвергаться декомпозиции на ряд внутренне связанных и слабо зависящих друг от друга модулей.

Оптимальный модуль должен удовлетворять двум критериям:

* снаружи он проще, чем внутри
* его проще использовать, чем построить

Модульная организация должна удовлетворять принципу информационной закрытости.

Информационная закрытость означает следующее:

* все модули независимы, обмениваются только информацией, необходимой для работы
* доступ к операциям и структурам данных модуля ограничен

Достоинства информационной закрытости:

* обеспечивает возможность разработки модулей различными, независимыми коллективами
* обеспечивается легкая модификация системы

Сложность программной системы:

V(i) - размерная сложность модуля(в виде LOC- или FP-оценки)

LOC - lines of code(число строк кода)

FP - Для продуктов с высокой алгоритмической сложностью используются метрики указателей свойств (Features Points)

func1 - число входов модуля

func2 - число структур данных, которое использует модуль

func3 - количество вызовов других модулей из данного

func4 - количество структур данных, обновляемых данным модулем

ИНФА ИЗ ГУГЛА!!!

**26. Создание модели предметной области программной системы с помощью диаграммы классов.**

Модель предметной области — это тип концептуальной модели, используемый для описания структурных элементов и их концептуальных ограничений в интересующей области (иногда называемой проблемной областью). Модель предметной области включает в себя различные сущности, их атрибуты и отношения, а также ограничения, определяющие концептуальную целостность элементов структурной модели, составляющих эту проблемную область. Подобно моделям сущностных отношений, модели предметной области могут использоваться для моделирования концепций или для моделирования объектов и событий реального мира.

Диаграмма классов позволяет описать объекты и связи в пределах предметной области. Для отображения сущности МПО используется класс. Сама сущность МПО представляет из себя абстракцию, которой может не существовать в реализации в описанном виде.

*Атрибуты* сущности МПО содержат перечень основных полей, при этом они могут содержать не все используемые в коде поля, так как их может быть достаточно много и в этом нет смысла.

Название атрибута задается в наиболее удобной для понимания форме. В случае с атрибутами чаще всего было проще использовать то название, которое указано в коде.

*Методы* бизнес-модели содержат все возможные действия, над данной моделью. Это могут быть CRUD и какие либо еще возможные методы.

Название метода в бизнес-модели указывается в свободной форме, таким образом, чтобы по названию можно было понять его суть.

ИНФА ИЗ ГУГЛА!!!

**27. Создание модели анализа с помощью диаграммы классов. Различные стереотипы для классов и их назначение.**

Модель анализа - модель, предназначенная для уточнения функциональных возможностей(требований) системы с учетом внутренней организации(архитектуры) проектируемой системы.

Построение этой модели необходимо:

- для выявления внутренней архитектуры (определения подсистем и основных классов);

- для поиска альтернативных вариантов реализации системы (подсистем) и выбора основного;

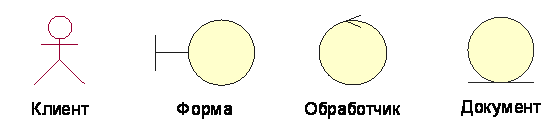
- для уточнения всех требований (функциональных и нефункциональных).

Стереотипы являются одним из трех типов механизмов расширяемости в унифицированном языке моделирования (UML). Они позволяют проектировщикам расширять словарь UML для создания новых элементов моделирования, получаемых из существующих, но имеющих определенные свойства, которые подходят для конкретной проблемы предметной области или для другого специализированного использования.

Стереотипы применимы для диаграммы классов, так как она является UML-диаграммой.

Графически стереотип отображается как имя, заключенное в кавычки («», или, если такие кавычки недопустимы, <<>>) и расположенное над именем другого элемента. В дополнение или в качестве альтернативы он может быть обозначен соответствующей иконкой. Значок может даже заменить весь символ UML. Например, стереотипы диаграммы классов могут быть использованы для описания методов поведения, таких как «конструктор» и «геттер». Несмотря на свое внешнее представление, «интерфейс» - не стереотип, а классификатор.

Примеры стереотипов для классов:

****

ИНФА ИЗ ГУГЛА!!!

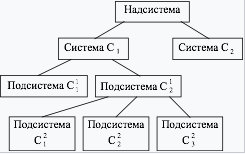
**28. Декомпозиция системы. Правила выделения подсистем.**

Правила декомпозиции:

1. Каждое расчленение образует свой уровень

Исходная система располагается на нулевом уровне. После её расчленения получаются *подсистемы* первого уровня. Расчленение этих подсистем или некоторых из них приводит к появлению подсистем второго уровня и т. д.

Упрощенное графическое представление декомпозированной системы называется ее *иерархической структурой*.

****

1. Система расчленяется только по одному, постоянному для всех уровней, признаку

В качестве признака декомпозиции может быть:

1. функциональное назначение частей,
2. конструктивное устройство (вид материалов, формы поверхностей и др.),
3. структурные признаки (вид схемы, способы и др.),
4. виды этапов и процессов (жизненный цикл, физическое состояние и др.),
5. предметные характеристики (экономические, информационные, технологические и др.),
6. и другие.
7. Вычленяемые подсистемы в сумме должны полностью характеризовать систему

Вычленяемые подсистемы должны взаимно исключать друг друга. Для обозримости рекомендуют выделять на каждом уровне не более 7 подсистем. Недопустимо, чтобы одной из подсистем являлась сама система.

1. Глубина декомпозиции

Степень подробности описания и количество уровней определяются требованиями обозримости и удобства восприятия получаемой иерархической структуры, её соответствия уровням знания работающему с ней специалисту.

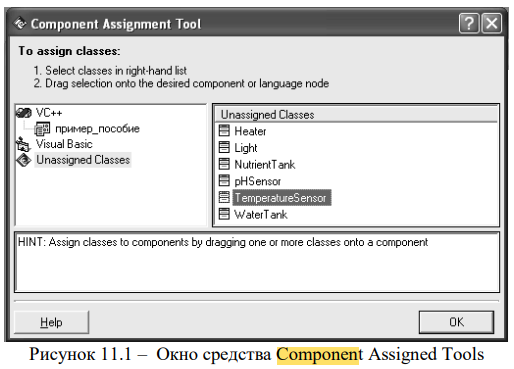
Обычно в качестве нижнего (элементарного) уровня подсистем берут такой, на котором располагаются подсистемы, понимание устройства которых или их описание доступно исполнителю (руководителю группы людей или отдельному человеку). Таким образом, иерархическая структура всегда субъективно ориентирована: для более квалифицированного специалиста она будет менее подробна.

Число уровней иерархии влияет на обозримость структуры: много уровней — задача труднообозримая, мало уровней — возрастает число находящихся на одном уровне подсистем и сложно установить между ними связи. Обычно, в зависимости от сложности системы и требуемой глубины проработки, выделяют 3…6 уровней.

ИНФА ИЗ ГУГЛА!!!

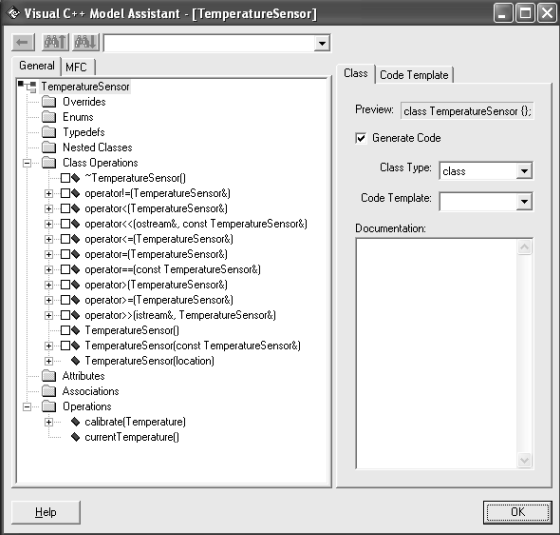
**29. Функциональные возможности Rational Rose: модуль Component Assignment Tool, компонент Model Assistant, обновление кода по модели и модели по коду. ТЕМА 11**

Модуль Component Assignment Tool - это инструмент для генерации кода на выбранном языке программирования. Процесс построения кода класса на заданном языке включает в себя использование встроенного в пакет модуля Model Assistant. Для того чтобы генератор Rational Rose мог создавать на основе описанной модели программный код, для каждого класса необходимо указать язык и определить компонент, в котором класс будет храниться. В случае языка VC++ открывается доступ ко всей иерархии классов библиотеки Microsoft Foundation Class. Ассоциация класса с VC++ происходит в результате выполнения следующих действий: Menu=> Tools=>Visual C++=>Component Assigned Tools, после чего открывается окно:



В его правой части выбирается класс и перетаскивается на значок VC++, а затем подтверждаются создание VC++ компонента и его связь с классом. Следующим открывается окно выбора проекта, где можно создать проект или выбрать уже существующий для помещения в него новый класс.

Model Assistant представляет собой окно, в котором создаются атрибуты, операции, а также задаются их свойства:



В окне имеется ряд полей: Preview показывает описание класса в текущий момент. Generate Code – ключ, определяющий необходимость создания для класса исходного текста на VC++; если ключ снят, то генерация кода не происходит и класс не показывается в списке классов для обновления кода. Class Type – установка типа класса: “class”, “struct”, “union”. Documentation – произвольные комментарии для класса.

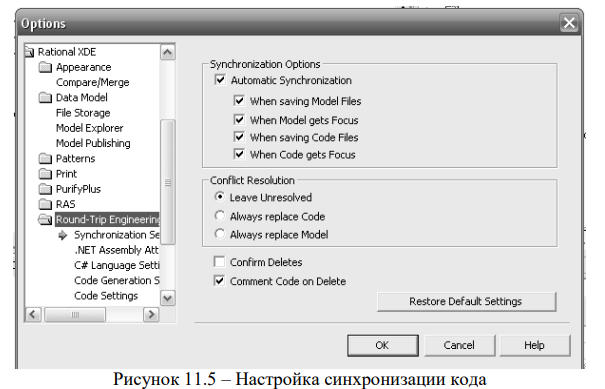
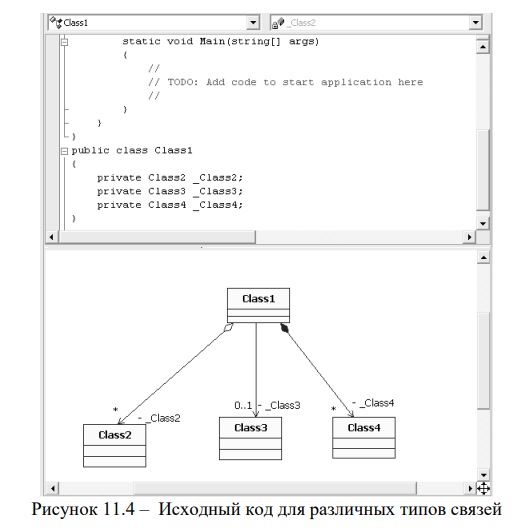
Одна из самых важных возможностей Rational Rose – это Update Code/Update Model. Данное средство позволяет создать проект Visual C++ по разработанной модели и обновить модель по уже готовому проекту, созданному при помощи MFC. Предположим, уже создан проект, и известно, что после того как последний раз изменялась модель, исходный код определенного класса не был исправлен, и его необходимо обновить. Для этого выбирается пункт Update Model, после чего появляется окно, в котором можно обновить либо все классы путем установки на них отметок, либо только выбранные, сняв с остальных отметки. Затем Rational Rose получает информацию из проекта Visual C++, для чего загружается Microsoft Visual Studio, и активизируется нужный проект. Если в код были внесены изменения или удалены компоненты, программа предложит выполнить такие же изменения и в модели. В случае, когда в проект Visual C++ были добавлены классы, и они еще не отражены в модели Rational Rose, необходимо провести обновление кода при помощи функции Update Code. Процесс обновления кода по изменениям в модели происходит аналогично обновлению модели.

**30.Особенности генерации исходного кода в среде Rational XDE. Способы синхронизации модели. ТЕМА 11**

Генерация по готовым диаграммам позволяет исключить рутинный труд по ручному кодированию и созданию шаблонов, что сокращает количество ошибок на стадии преобразования готовой модели в программный код. Кроме того, по любой программе, написанной при помощи среды Microsoft .NET, можно провести обратное проектирование и разобраться в архитектуре, представленной графически в виде иерархии классов.

Для работы с исходным кодом в контекстном меню класса доступна следующая группа пунктов меню: – Generate Code – генерация кода по созданной модели. – Synchronize – синхронизация кода и модели. При этом все изменения, внесенные в код, отражаются в модели, а изменения модели переносятся в исходный код. – Browse Code – позволяет переключаться в режим просмотра созданного кода. Для работы с одним-двумя классами этого вполне достаточно, но при работе с большими моделями удобнее использовать синхронизацию в автоматическом режиме.

При создании исходного кода ассоциативные связи создают одинаковый код независимо от того, направленная это ассоциация или нет, отражает ли связь агрегацию или композицию классов. При генерации кода будет создана переменная класса, заданная в параметрах связи. При этом направление связи не играет роли, а важно задание имени переменной. При создании связи Directed Association Rational XDE автоматически создает переменную класса того типа, который находится в окончании стрелки связи. Эта переменная включается в класс, из которого стрелка исходит. Для настройки параметров автоматической синхронизации модели необходимо выбрать из главного меню пункт Tools => Options => Rational XDE => Round Trip Engineering => Synchronization Setting. После отметки в приведенном окне пункта Automatic Synchronization становятся доступными следующие варианты синхронизации: – When Saving Model Files – синхронизация в момент сохранения модели; – When Model gets Focus – синхронизация в момент активизации модели; – When saving Code Files – синхронизация происходит в момент сохранения кода после его изменения; – When Code gets Focus – синхронизация происходит в момент активизации окна кода. Каждый вариант синхронизации удобен для своего случая. На этапе анализа и проектирования, когда основную работу по моделированию выполняет аналитик, синхронизации в момент сохранения модели позволит программистам, работающим в проекте, пользоваться самой последней версией структуры приложения. На этапе реализации, когда основная работа ложится на программистов, установка синхронизации после сохранения внесения изменений в файлы кода позволит поддерживать модель системы в актуальном состоянии. Когда же разработка закончена или создается новая версия уже работающей системы, чтобы не нарушить работу программы, синхронизация может быть вообще отключена.



**31.Назначение, возможности, особенности использования модуля Web Modeler в Rational Rose. ТЕМА 12**

Подключаемые модули позволяют использовать Rational Rose еще и для создания модели баз данных и разработки структуры Web сайтов. Это логично, поскольку приложение, работающее на Internet сервере, в принципе ничем не отличается от обычной программы. Структура Web сайта также разрабатывается, затем кодируется и тестируется и, наконец, выставляется для всеобщего использования. Internet приложение моделируется на основе диаграммы классов. В Rational Rose для создания Web приложений включен модуль Web Modeler, который появится в меню инструментов после включения. Web Modeler позволяет моделировать Web приложения, создавать модель по существующему программному коду, а затем снова выполнять генерацию jsp, asp, html файлов согласно изменениям в модели. Меню Web Modeler состоит из двух пунктов: User Preferences (установки пользователя) и Reverse Engineer a New Web Application.

Пункт User Preferences предназначен для изменения установок модели, принятых по умолчанию для конкретного пользователя, что позволяет нескольким пользователя работать с одной Web моделью, но со своими настройками. Эти установки позволяют контролировать построение модели по коду, создание кода по модели, генерировать структуру каталога для ссылок, связей и HTML форм. Установки позволяют настроить действия генератора для тегов HTML и скриптов.

Пункт Reverse Engineer a New Web Application позволяет создавать модель по готовому коду приложения. При его активизации запускается мастер Reverse Engineer, который помогает осуществить использование готового кода для создания модели. Мастер позволяет задать платформу ASP или JSP (Java Server Pages) и каталог, в котором находится исходный код. После того как мастер закончит работу в текущей модели, образуется структура выбранных классов, в которой отражаются связи, атрибуты и стереотипы. Для построения Web приложения используется диаграмма классов, и их следует создавать в Web нотации. Для этого необходимо выбрать следующие пункты Menu:Tools=>Options=>Notation=>Default Language=Web Notation. При этом классы имеют специальные стереотипы, которые влияют на изображения класса на диаграмме. Для моделирования Web приложения Rational Rose предоставляет ряд стереотипов классов:

Стереотип Server page отображает скрипты, выполняемые на стороне сервера. Это может быть ввод данных и обработка запросов. Стереотип Client page позволяет показать использование скриптов на стороне клиента, кроме того, он отражает простые HTML страницы, текст, графику, мультимедиа объекты. HTML формы позволяют пользователю взаимодействовать с Web приложением. Для связей страниц и форм Web Modeler предоставляет дополнительные стереотипы, которые позволяют отразить характер связей. Рассмотрим имеющиеся стереотипы:

- Link показывает гиперссылку одной страницы на другую и может быть двунаправленной.

- Submit определяет, что форма взаимодействует со страницей на сервере и передает ей данные. - Build показывает, что страница создается сервером.

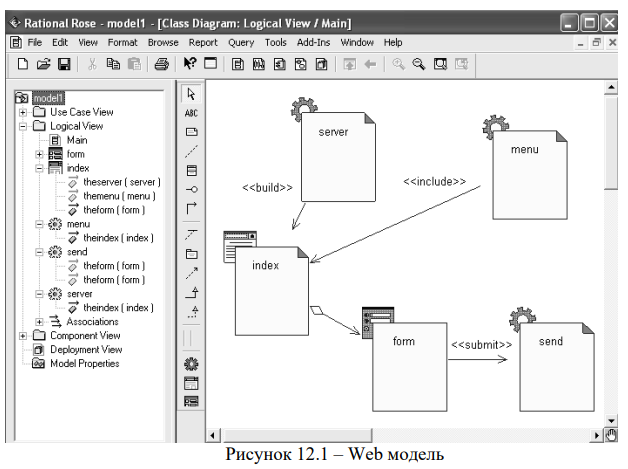
- Redirect определяет передачу управления одной серверной странице от другой и используется для страниц ASP.

- Includes означает, что одна страница включается в другую.

- Forward похожа на связь Redirect, только используется для страниц JSP.

**32. Возможности и особенности построения Web-модели в среде Rational XDE. ТЕМА 12**

После того как рассмотрены стереотипы, с их помощью можно создавать простые Web-приложения. Предположим, необходимо построить приложение, состоящее из главной страницы, которая создается сервером и включает меню. На этой же странице находится форма ввода, которая позволяет отправлять сообщения. Форма ввода будет взаимодействовать со страницей сервера, которая и будет непосредственно отправлять запросы:



Для разработки приложения необходимо пройти несколько шагов. 1. Установка языка моделирования, как было описано ранее. 2. Из контекстного меню выбрать Web Modeler =>New=>Virtual Directory Name и задать каталог, в котором будет находиться исходный текст, сгенерированный на основе модели приложения. 3. Добавить необходимые элементы в модель посредством строки инструментов диаграммы классов. 4. Установить свойства и добавить атрибуты. 5. Создать исходный текст.

Особенности:

1. В Rational XDE для создания структуры системы, ориентированной на работу в Web, используется диаграмма классов, в которой при создании архитектуры приложения учитываются ограничения реализации Web приложения. В спецификации UML не предусмотрен отдельный тип диаграмм для выполнения этой задачи. Вполне достаточно диаграммы классов с дополнительными стереотипами для работы с Web-элементами.
2. Для построения Web-модели можно использовать новое или существующее приложение, создать в нем новую модель, а затем выполнить синхронизацию проекта с моделью. Для этого в контекстном меню проекта необходимо выбрать пункт Synchronize. В случае, если модель еще не создана, Rational XDE создает новую модель с таким же названием, как у проекта, и устанавливает все необходимые ссылки. После синхронизации Rational XDE открывает новую диаграмму, готовую к работе.

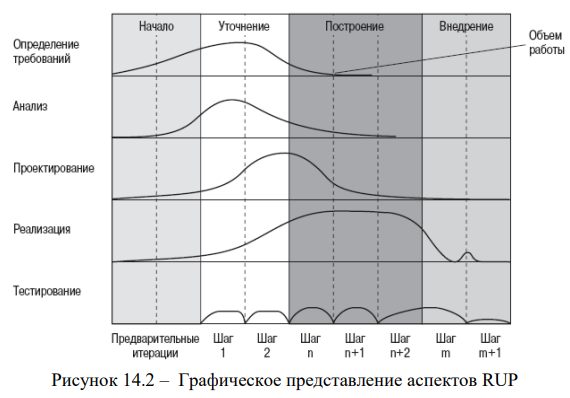
**33. Понятие проектных рисков. Действия по управлению рисками. ТЕМА 14**

Риск — это событие с долей неопределённости, которая может сказаться на проекте отрицательно или положительно, привести к возникновению угроз для проекта или появлению возможностей. Риск проекта - это степень опасности для успешного осуществления проекта. Риски бывают негативными и позитивными. Пример позитивного риска: для разработки визуального интерфейса приложения были наняты сотрудники из другой компании. Когда заключали договор оказалось, что компания сделает скидку на разработку, в результате чего стоимость разработки GUI снизится на 10 процентов. Негативный риск — это не проблема, а только вероятность. Если разработчики его учитывают, то можно скорректировать действия, когда риск приведёт к проблеме.

Одним из способов управлять рисками, а именно их уменьшить, является использование индустриальных методов разработки, например, разработка по Rational Unified Process. Поскольку каждый программный проект по-своему уникален, то хотя в основу RUP и положены рекомендации по всем стадиям и фазам разработки программ, не для каждого проекта они подходит на сто процентов. Но в любом случае, применение унифицированного процесса разработки позволит уменьшить затраты проекта, уложиться в сроки и повысить качество создаваемого программного продукта. RUP обеспечивает строгий подход к распределению задач и ответственности внутри организации-разработчика. Его цель заключается в том, чтобы гарантировать создание качественного ПО точно в срок и в рамках установленного бюджета, отвечающего требованиям конечных пользователей. Частью RUP является создание моделей при помощи унифицированного языка моделирования UML. Унифицированный процесс можно представить как сумму различных видов деятельности компании-разработчика, необходимых для перевода требований заказчика в программную систему, которая выполняла бы именно то, что от нее ожидают пользователи, и выдавала требуемый результат. Диаграммы позволяет легче общаться членам проекта между собой, и, что особенно ценно, вовлекают в процесс конечных пользователей системы. Моделирование позволяет уменьшить риски проекта.

**34. Статический и динамический аспекты Rational Unified Process (RUP). ТЕМА 14**

RUP организует работу над проектом в терминах статических и динамических аспектов процесса. К статическим аспектам относятся последовательности действий, продуктов деятельности, исполнителей, а к динамическим – циклы, фазы, итерации и временные отметки завершения определенных этапов в создании ПО:



Каждая из фаз процесса разработки состоит из нескольких итераций, целью которых является последовательное осмысление стоящих проблем, наращивание эффективных решений и снижение риска потенциальных ошибок в проекте. В то же время, каждая из последовательностей действий по созданию ПО выполняется в течение нескольких фаз, проходя пики и спады активности. Каждый цикл итерации проекта начинается с планирования того, что должно быть выполнено. Результатом выполнения должен быть качественный продукт. Заканчивается же цикл оценкой того, что было сделано, и были ли достигнуты цели. В основу Rational Unified Process положена база знаний, размещаемая на Web-сервере и состоящая из руководств, шаблонов, наставлений по использованию инструментальных средств.

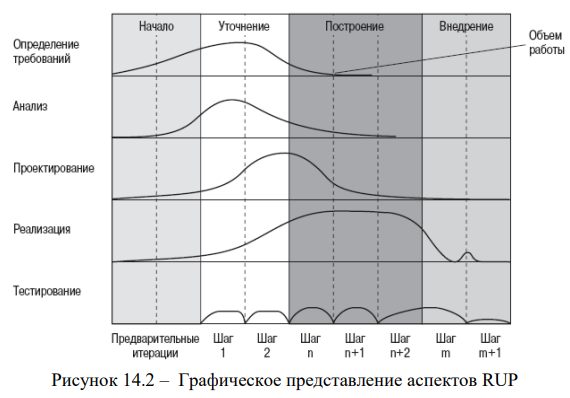
**35. Принципы и стадии разработки ПС в технологии Rational Unified Process. ТЕМА 14**

В основе RUP лежит ЖЦ создания ПС.

1. Определение требований. Фактически, это взгляд пользователей на программную систему снаружи. Таким образом, одним из важнейших этапов разработки, согласно RUP, является этап определения требований, который заключается в сборе всех возможных пожеланий к работе системы. Позднее эти данные 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. Работа усложняется тем, что пользователи часто сами не представляют, что они должны получить в конечном итоге. Для облегчения этого процесса аналитики используют диаграммы прецедентов (Use Case), а для детализации конкретного прецедента применяются диаграммы активности (Activity Diagram). Простота и понятность диаграммы прецедентов позволяет аналитикам легко общаться с заказчиками в процессе определения требований. Определение требований к системе напрямую связано с понимание контекста (части предметной области), в котором будет работать система. Для этого создаются модель предметной области и бизнес-модель. Различия этих моделей в том, что первая описывает важные понятия, с которыми будет работать система и их связи между собой. А вторая описывает бизнес-процессы, 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 диаграмма классов (Class Diagram), однако для создания бизнес-модели ее уже недостаточно. В этом случае применяется диаграмма прецедентов с использованием дополнительных значков, отражающих сущность бизнес процессов – это бизнес-актант, бизнес-прецедент, бизнес-сущность и бизнес управление.
2. Анализ требований. После определения требований и контекста, в котором будет работать система, наступает этап 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. Модель анализа является в большей степени концептуальной моделью и только приближает разработчиков к классам реализации. Для отображения модели анализа при помощи UML используется диаграмма классов с образцами поведения (стереотипами) «граничный класс», «сущность», «управление», а для детализации используется диаграмма сотрудничества (Collaboration Diagram). Если акцентировать внимание на порядке взаимодействий, то удобнее это реализовать с помощью диаграммы последовательности (Sequence Diagram). 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 обе они позволяют отразить взаимодействие между объектами.
3. Проектирование. Следующим этапом в процессе создания системы будет проектирование, в ходе которого на основании моделей, созданных ранее, строится модель проектирования. Она отражает физическую реализацию 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. Эта модель должна поддерживаться в актуальном состоянии на протяжении всего жизненного цикла разработки системы. Для построения модели проектирования используется целый набор UML диаграмм: классов, кооперации, взаимодействия, активности. Дополнительно в этом рабочем процессе может создаваться модель развертывания, которая реализуется на основе диаграммы развертывания (Deployment Diagram).
4. Реализация. Основная задача процесса реализации – создание системы в виде набора компонентов. Это могут быть исходные тексты программ, сценарии, двоичные файлы, исполняемые модули и т.д. Модель реализации строится в виде диаграммы компонентов (Component Diagram)
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 в виде UML диаграмм.

**36. Содержание и результаты первой и второй стадий в технологии Rational Unified Process. ТЕМА 15**

Согласно RUP ЖЦ ПО разбивается на отдельные циклы, в каждом из которых создается новое поколение продукта. Каждый цикл, в свою очередь, разбивается на четыре последовательные стадии: - начальная стадия(начало); - стадия разработки(уточнение); - стадия конструирования(построение); - стадия ввода в действие(внедрение):



Каждая стадия завершается в четко определенной контрольной точке. В этот момент времени должны достигаться важные результаты и приниматься критически важные решения о дальнейшей разработке.

Начальная стадия может принимать множество разных форм. В это же время вырабатывается бизнес-план проекта: определяется, сколько приблизительно он будет стоить и какой доход принесет. Кроме того, выполняется начальный анализ для оценки размеров проекта. Результатами начальной стадии являются:

- общее описание системы: основные требования к проекту, его характеристики и ограничения;

- начальная модель вариантов использования;

- начальный проектный глоссарий;

- начальный бизнес-план;

- план проекта, отражающий стадии и итерации;

- один или несколько прототипов.

На стадии разработки выявляются более детальные требования к системе, выполняется высокоуровневый анализ предметной области и проектирование для построения базовой архитектуры системы, создается план конструирования и устраняются наиболее рискованные элементы проекта. Результатами стадии разработки являются:

- завершенная модель вариантов использования, определяющая функциональные требования к системе;

- перечень дополнительных требований, включая требования нефункционального характера и требования, не связанные с конкретными вариантами использования;

- описание базовой архитектуры будущей системы;

- работающий прототип;

- уточненный бизнес-план;

- план разработки всего проекта, отражающий итерации и критерии оценки для каждой итерации.

Самым важным результатом стадии разработки является описание базовой архитектуры будущей системы. Эта архитектура включает в себя:

- модель предметной области, которая отражает понимание бизнеса и служит отправным пунктом для формирования основных классов предметной области;

- технологическую платформу, определяющую основные элементы технологии реализации системы и их взаимодействие.

Созданная архитектура является основой всей дальнейшей разработки. В будущем неизбежны незначительные изменения в деталях архитектуры, однако, серьезные изменения маловероятны. Стадия разработки занимает около пятой части общей продолжительности проекта.

**37. Содержание и результаты третей и четвертой стадий в технологии Rational Unified Process.**

Стадия конструирования напрямую связана с повторяющейся проработкой итераций. Повторяемость относится к разрабатываемому коду: на каждой итерации некоторая часть существующего кода переписывается с целью сделать его более гибким. Результатом стадии конструирования является продукт, готовый к передаче конечным пользователям и содержащий следующее:

- ПО, интегрированное на требуемых платформах;

- руководства пользователя;

- описание текущей реализации.

Стадия ввода в действие связана с передачей готового продукта в распоряжение пользователей. Она включает в себя:

- бета-тестирование, позволяющее убедиться, что новая система соответствует ожиданиям пользователей;

- параллельное функционирование с существующей системой, которая подлежит постепенной замене;

- конвертирование баз данных;

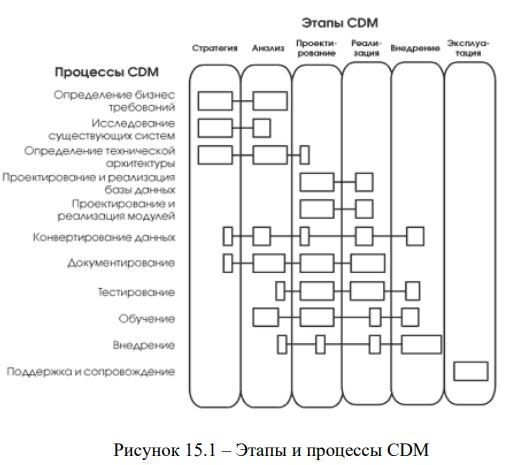
- оптимизацию производительности;

- обучение пользователей и специалистов службы сопровождения.

Статический аспект RUP представлен четырьмя основными элементами: роли; виды деятельности; рабочие продукты; дисциплины(последовательность действий человека данной роли).

**38. Этапы создания программных средств в технологии Oracle.**

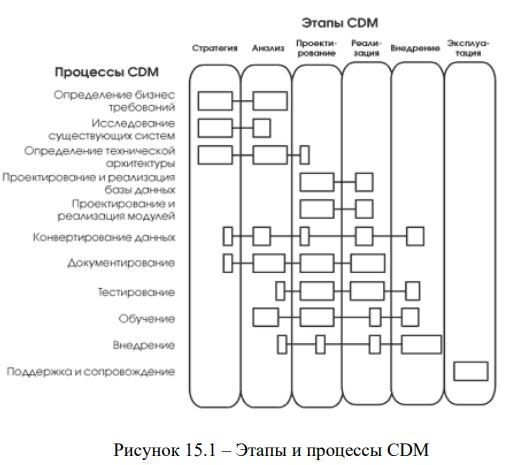
В соответствии с CDM ЖЦ ПО формируется из определенных этапов (фаз) проекта и процессов, каждый из которых выполняется в течение нескольких этапов.



1. На этапе стратегии определяются цели создания системы, приоритеты и ограничения, разрабатывается системная архитектура и составляется план разработки.
2. На этапе анализа строятся модель информационных потребностей (диаграмма "сущность-связь"), диаграмма функциональной иерархии, матрица перекрестных ссылок и диаграмма потоков данных.
3. На этапе проектирования разрабатывается подробная архитектура системы, проектируются схема реляционной БД и программные модули, устанавливаются перекрестные ссылки между компонентами системы для анализа их взаимного влияния и контроля за изменениями.
4. На этапе реализации создается БД, строятся прикладные системы, производится их тестирование, проверка качества и соответствия требованиям пользователей. Создается системная документация, материалы для обучения и руководства пользователей.
5. На этапах внедрения и эксплуатации анализируются производительность и целостность системы, выполняется поддержка и, при необходимости, модификация системы.

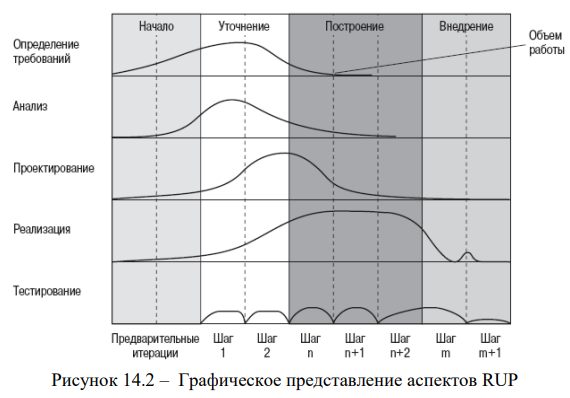
**39. Процессы создания программных средств в технологии Oracle.**

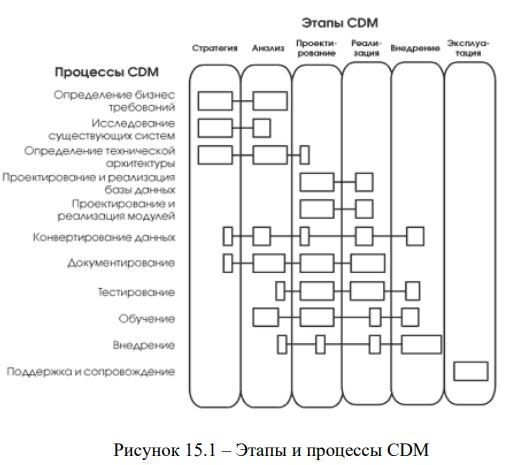
В соответствии с CDM ЖЦ ПО формируется из определенных этапов (фаз) проекта и процессов, каждый из которых выполняется в течение нескольких этапов.



1. Определение бизнес-требований подразумевает постановку задачи проектирования.
2. Исследование существующих систем должно обеспечить понимание состояния, созданного технического и программного обеспечения для планирования необходимых изменений.
3. Определение технической архитектуры связано с выбором конкретной архитектуры разрабатываемой системы.
4. Проектирование и реализация базы данных это создание реляционной базы данных, включая индексы и другие объекты БД.
5. Процесс проектирования и реализации модулей является основным в проекте. Он включает в себя непосредственное проектирование приложения и создание кода прикладной программы.
6. Конвертирование данных связано с преобразованием, переносом и проверкой согласованности и непротиворечивости данных, оставшихся от "старой" системы и необходимых для работы в новой системе.
7. Документирование, тестирование и обучение – это процессы подготовки системы к внедрению.
8. Процесс внедрения связан с решением задач установки, ввода новой системы в эксплуатацию, прекращением эксплуатации старых систем.
9. Поддержка и сопровождение обеспечивают эксплуатацию созданной системы.

**40. Сравнительный анализ технологий создания ПС Rational Unified Process, Oracle.**





Читать выше!

**41. Понятия CASE-средство, CASE-система, CASE-технология, CASEиндустрия и различия между ними.**

CASE-средства (Computer Aided Software Engineering) обеспечивают поддержку многочисленных технологий проектирования информационных систем, охватывая всевозможные средства автоматизации и весь жизненный цикл программного обеспечения (ПО).

CASE-технология включает в себя методологию анализа, проектирования, разработки и сопровождения сложных систем ПО, поддержанную комплексом взаимосвязанных средств автоматизации. Проще говоря, технология реализует средства.

CASE- системы возникли как средство повышения эффективности в разработке сложного программного обеспечения, соответствующие CASE-системы часто называют инструментальными средами разработки ПО. Они поддерживают:

1. Единый графический язык.
2. Единая БД проекта.
3. Поддержка коллективной разработки и управления проектом.
4. Макетирование.
5. Генерация документации.
6. Верификация проекта.
7. Автоматическая генерация объектного кода.
8. Сопровождение и реинжинирин**г**

Основная цель CASE-подхода – разделить и максимально автоматизировать все этапы разработки ПС. Большинство CASE-средств основано на парадигме методология / метод / нотация / средство. Методология определяет шаги работы и их последовательность, а также правила распределения и назначения методов. Метод – это систематическая процедура генерации описаний компонентов ПО. Нотация предназначена для описания структур данных, порождающих систем и метасистем. Средства – это инструментарий для поддержки методов на основе принятой нотации.

**42. Определение информационной системы. Бизнес-модель и информационная модель информационной системы.**

**43. Примеры построения общей диаграммы деятельности и модели бизнеспрецедентов информационной системы.**

**44. Определение и назначение пользовательского интерфейса. Семь свойств пользовательского интерфейса.**

**45.Показатели, оценивающие качество интерфейса. Правила создания эффективного пользовательского интерфейса.**

**46. Технологии создания пользовательских интерфейсов. Формы ввода с привязкой к СУБД. Обработчики шаблонов.**

**47. Технологии создания пользовательских интерфейсов. Объектноориентированные, событийные и декларативные инструменты.**

**48. Технологии создания пользовательских интерфейсов. Инструменты на основе моделей. Обобщенные пользовательские интерфейсы.**