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

CASE-технология включает в себя методологию анализа, проектирования, разработки и сопровождения сложных систем ПО, поддержанную комплексом взаимосвязанных средств автоматизации. 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. Начиная с 60-ых годов ХХ столетия, через каждые десять лет указанный разрыв значительно увеличивался.

На первых порах проблему пытались решить путем перевода людей из одной области труда в другую, что привело к падению общих темпов роста производительности труда и экономики. Кроме этого, ситуацию осложняли следующие факторы:

1. Число различных классов технических систем удваивалось в среднем через каждые десять лет.

2. Сложность изделий, вызванная увеличением количества их комплектующих, удваивалась через десять лет.

3. Объем научно-технической информации удваивался через каждые восемь лет.

4. Период создания новых изделий уменьшался в два раза через десять лет, и при этом сокращалось время их морального старения.

В связи с этим были начаты работы по автоматизации процессов проектирования. В современных условиях необходима комплексная компьютеризация инженерной деятельности на всех этапах жизненного цикла (ЖЦ) программного продукта. 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.

**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; P – цели системы; Atr – общесистемные характеристики; Inp – входы системы; Out – выходы системы; Str – структура системы; Str={E, R}, где E–компоненты системы, R – связи между компонентами.

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.

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.

Уменьшение трудоемкости проектирования достигается за счет:

- автоматизации оформления документации,  
- информационной поддержки и автоматизации принятия решений,  
- параллельного проектирования.

Снижение себестоимости разработки происходит благодаря разумной экономии всех ресурсов.

Сокращение ЖЦ достигается за счет параллельного проектирования и создания виртуальных бюро.

Улучшение качества результатов проектирования обеспечивается путем:

- использования автоматизированного поискового и многовариантного проектирования,  
- применения математических методов оптимизации параметров и структур объектов и процессов,  
- привлечения стратегического проектирования.

Общесистемные характеристики участвуют в классификации CASE- средств по следующим признакам:

- прикладная область объектов проектирования,  
- сложность проектируемых объектов,  
- уровень автоматизации (менее 25% от общего количества автоматизированных процедур – низкоавтоматизированные CASE; от 25% до 50% автоматизированных процедур – среднеавтоматизированные CASE; более 50% автоматизированных процедур – высокоавтоматизированные CASE),  
- комплексность автоматизации проектирования (зависит от стадий проектирования, которые охватывает система, здесь выделяют: одноэтапные, многоэтапные и комплексные),  
- возможность работы в сетевом режиме и в Internet.

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

**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.

Независимо от критерия, которым измеряется, прежде всего производительность CASE-средства зависит от объема знаний и данных, заложенных в систему.

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

Критерий технологических возможностей отражает простоту разработки CASE-средства и подготовки ее к эксплуатации.

*Экономический* критерий CASE-средства служит для комплексного стоимостного учета положительного эффекта от автоматизации проектирования и основных затрат. В качестве этого показателя принято использовать КЭ – величину годового экономического эффекта от использования CASE-средства.

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.

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

Проект ­ работа/группа работ, которую необходимо выполнить за определённый промежуток времени, не выходя за рамки сметы. Хар­ки проекта:

1) Масштаб ­ цели и запланированные результаты  
2) Качество ­ уровень качества  
3) Время ­ этапы и сроки выполнения + время  
4) Стоимость ­ бюджет(смета)

<­ треугольник проекта

ЖЦ проекта (этапы развития проекта) (минимум 4):

1. постановка задачи
2. планирование проекта
3. реализация
4. завершение проекта

Управление проектом включает:  
1) Управление изменениями – определение, согласование, утверждение, и принятие к выполнению корректирующих воздействий и координация изменений по всему проекту  
2) Управление ресурсами – внесение изменений в состав исполнителей и назначение ресурсов на работы проекта   
3) Управление масштабом – корректировка целей проекта по результатам анализа  
4) Управление качеством – разработка мероприятий по устранению причин неудовлетворительного выполнения работ

Вспомогательные процессы:  
5) Управление рисками – реагирование на события и изменение рисков в процессе выполнения проекта  
6) Управление контрактами – координация работы подрядчиков, корректировка контрактов, разрешение конфликтов

Современное профессиональное Управление проектами базируется на следующих основных принципах:  
- четкое определение целей, результатов и работ проекта с учетом возможных приемлемых рисков;  
- определение центров ответственности за проект в целом и отдельные его части;  
- создание системы комплексного и прогнозирующего планирования работ и параметров проекта;  
- создание системы контроля и регулирования хода выполнения проекта;  
- создание команды проекта и управление ею с целью объединения и координации усилий всех исполнителей, вовлеченных в проект.

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

Одинаково важны все три компонента — система обозначений (нотация, язык), процесс и инструмент.

Можно изучить язык, но, не зная, как его применять (т.е. не владея процессом), легко попасть впросак. Если в вашем распоряжении имеется замечательный процесс, но вы не умеете с ним обращаться (т.е. не знаете системы обозначений), работа также обречена на неудачу. Наконец, не приходится ожидать ничего хорошего и в том случае, когда в отсутствие подходящего инструмента вы не в состоянии зафиксировать результаты своей работы.

Система обозначений

Система обозначений, или нотация (notation), важна в любой модели — это своего рода клей, который скрепляет все составляющие процесса. Нотация выполняет следующие функции:

- играет роль языка, используемого для описания не очевидных выводов, которые не проистекают непосредственно из кода как такового;

- сообщает семантику всех стратегических и тактических решений;

- обеспечивает форму представления, достаточно конкретную для восприятия человеком и возможности манипуляций ею с помощью инструментальных средств”.

Примером исчерпывающе полной и весьма надежной системы обозначений может служить унифицированный язык моделирования (Unified Modeling Language — UML), охватывающий и этап анализа, и стадию дизайна.

Значимость процесса

Успешный проект, способный удовлетворить или даже превзойти чаяния потребителя, должен разрабатываться с помощью эффективных средств в соответствии со строгим графиком и обеспечивать возможность внесения изменений и адаптации к конкретным условиям и требованиям.

Жизненный цикл проекта обязан содействовать проявлениям творческой инициативы и в то же время быть вполне управляемым, что позволит гарантировать его непременное завершение.

Инструмент

Ни одна из методологий разработки программ не обходится без некоторого инструмента. Сегодня на рынке имеется масса различных инструментальных средств — от простейших чертежных программ до самых изощренных систем моделирования объектов. Без наличия необходимого инструмента вы не сможете зафиксировать результат своей работы.

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

Типы программ:

1) автономные ­ 1. установл. на одиночный комп; - 2. не связ аппаратн и прогр обеспеч  
2) встроенные ­ часть уник прилож с привл аппар обеспеч  
3) реального времени  
4) сетевые

Хар­ки программ (объекта внедрения):

1) структурная сложность (многоуровневая иерархия структура организации)  
2) функциональная сложность (большое кол­во ф­ий, сложные связи между ними)  
3) сложная динамика поведения (высокая изменчивость внешней и внутренней среды)

Технические:

1) отсутствие(наличие) аналогов  
2) большое кол­во локальных объектов внедрения

Организ. хар­ки:

1) различные формы организации и управления проектом:  
- эксперимент  
- с участ разработч / сторон. Комп  
2) значит длительность ЖЦ =>  
­ нестабильность функционала  
­ поддержка проекта  
3) высокие требования со стороны заказчика к уровню технологич. зрелости организ разработчиков.

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

Согласно Алистеру Кобурну, методология нужна для того, чтобы:

1. Облегчить процедуру введения новых людей в курс процесса производства  
2. Обеспечить взаимозаменяемость людей  
3. Распределить ответственность  
4. Произвести впечатление на спонсора/заказчика  
5. Демонстрировать видимый прозрачный процесс  
6. Создать учебную базу для своих сотрудников

По определению Джима Хайсмита «Настоящее назначение методологий – это увеличение производительности, обеспечивающее решения для заказчиков»

Методологии можно условно разбить на 3 категории:

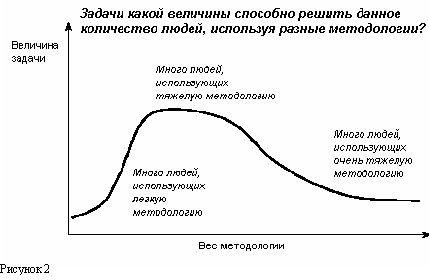
1. тяжелые 2. легкие 3. средние

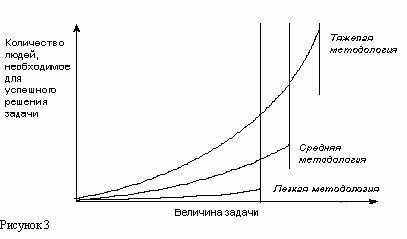
Тяжелые методологии отличаются тем, что охватывают все аспекты деятельности компании, производящей программное обеспечение – от управления требованиями и планирования процесса до регламентирования взаимоотношений с субподрядчиками и описания требований к вспомогательным процессам. Все методологии данной категории нетерпимы к изменениям и рассматривают людей как обычный ресурс.

Вторая категория(легкие) методологий появилась на свет в качестве некоторой совокупности методов и практик, применявшихся небольшими командами разработчиков в успешных проектах. Здесь огромное значение уделяется взаимоотношениями между людьми внутри команды, учитываются вопросы терпимости и другие психологические аспекты. Все процессы данной категории предусматривают итерационный жизненный цикл разработки ПО.

В третью(средние) категорию методологий попадают так называемые «универсальные» процессы. Самым ярким и известным представителем данной категории является RUP. Основной характеристикой подобных процессов является масштабируемость – т.е. процесс может быть настроен как на работу в малой команде над небольшим проектом, так и в большой команде над большим и серьезным проектом.

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





Алистер Кобурн является автором концепции «Разработка программного обеспечения – это совместная игра изобретения и взаимодействия». Он выделяет 2 цели у этой игры:

- разработать и внедрить работающее программное обеспечение.

- обеспечить все условия для успешного протекания следующей игры.

Таким образом, формулируется минимаксовая задача: произвести на свет ПО и обеспечить развитие следующих версий данного ПО. В тоже время, легкие процессы неприменимы для работы на больших проектах или над продуктами, степень критичности правильной работы которых очень высока.

На рисунке 1 показана динамика производительности команды, состоящей из нескольких человек и использующей разные методологии.

Рисунок 2 отражает динамику производительности большой команды, использующей методологии различного веса.

Рисунок 3 отражает потолок возможностей команды при использовании методологий разного веса.

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

1) из­за размера команд разработчиков (большая команда ­ большая методология)

2) у разных заказчиков разные желания (получить ПП быстро + мин дефекты; ­ наблюдение за каждой стадией и т.д.)

Принципы разработки методологии:

1. большая методология ­ большое число разработчиков. Чем больше проект, тем больше методология.

“Большая методология” ­ методология с большим числом элементов

1. большая плотность, когда скрытые ошибки влекут знач. ущерб  
   Принцип: затраты на защиту могут себя оправдать
2. увеличение размера/плотности методологии ведет к увеличению стоимости
3. эффективная форма коммуникации ­ взаимодействие лицо к лицу

Если проект растёт и нельзя обеспечить непосредственную коммуникацию между разработчиками ­> эффективность падает затраты растут

**10.Унифицированный процесс разработки программных проектов**

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

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 мини- проектов 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.

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.

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

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 или переопределения данных и методов. Система изначально строится с учетом ее эволюции, которую позволяют реализовать полиморфизм и наследование: потомки могут добавлять в родительские классы новые структуры данных и методы.

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

UML (англ. Unified Modeling Language — унифицированный язык моделирования) — язык графического описания для объектного моделирования в области разработки ПО, моделирования бизнес­процессов, системного проектирования и отображения организационных структур. UML позволяет разработчикам ПО достигнуть соглашения в графических обозначениях для представления общих понятий (класс, компонент, обобщение, агрегация, поведение), больше сконцентрироваться на проектировании и архитектуре.

В 1994 Гради Буч и Джеймс Рамбо, из компании Rational Software, объединились для создания нового языка объектно­ориентированного моделирования. За основу были взяты методы моделирования Object­Modeling Technique и Booch. OMT был ориентирован на анализ, а Booch — на проектирование ПС.

Осенью 1995 к компании Rational присоединился Ивар Якобсон, автор метода Object­Oriented Software Engineering — OOSE. OOSE обеспечивал возможности для спецификации бизнес­процессов и анализа требований при помощи сценариев использования. OOSE был также интегрирован в унифицированный метод.

В настоящее время UML является общепризнанным стандартом моделирования. В UML­модели есть два аспекта:

- статическая структура – описывает, какие типы объектов важны для моделирования системы и как они взаимосвязаны;

- динамическое поведение – описывает ЖЦ этих объектов и то, как они взаимодействуют для обеспечения требуемой функциональности системы.

Основные цели создания унифицированного языка моделирования:

1. Предоставить пользователям готовый к применению выразительный язык визуального моделирования, для разработки осмысленных моделей и обмена ими.

2. Предусмотреть механизмы для расширения базовых концепций.

3. Обеспечить независимость UML от конкретных ЯП и процесса разработки.

4. Создать формальную основу для понимания языка моделирования.

5. Стимулировать рост рынка объектно­ориентированных инструментальных средств.

6. Интегрировать лучший практический опыт.

**13.Сравнительный анализ CASE­средств Rational Rose и Rational XDE, Enterprise Architect.**

Rational Rose – это ОО­средство автоматизированного проектирования ПС. В его основе лежит CASE­технология, комплексный подход и использование единой унифицированной нотации на всех этапах ЖЦ создания ПС. В среде Rational Rose проектировщик и программист работают в тандеме: 1ый создает логическую модель системы, а 2ой дополняет ее моделями классов на конкретном ЯП.

Преимущества от применения Rational Rose:

1) сокращение цикла разработки приложения “заказчик­программист­заказчик”;  
2) увеличение продуктивности работы программиста;  
3) улучшение потребительских качеств создаваемых программ за счет ориентации на пользователей и бизнес;  
4) способность вести большие проекты и группы проектов;  
5) возможность повторного использования уже созданного программного обеспечения за счет упора на разбор их архитектуры и компонентов.

Главное отличие Rational XDE от своего предшественника, программы Rational Rose, это полная интеграция с платформой Microsoft Visual Studio.NET, позволяющая в одной оболочке работать как с моделями программной системы, так и с кодом. За счет этого заметно сократилось время синхронизации модели и программного кода, в результате чего повысилась производительность рабочей станции.

Возможности Rational XDE:

1) представляет собой единый процесс проектирования и программирования. Таким образом, на этапах бизнес­моделирования и кодирования может использоваться одна и та же модель. Причем, аналитики могут работать в традиционной Rose, а разработчики ­ в XDE, не мешая друг другу;  
2) позволяет проводить прямое и обратное проектирование, а также синхронизацию кода и модели, что позволяет структурировать проект любого типа сложности;   
3) имеет возможность по созданию шаблонов проектирования, так называемых паттернов, используя которые разработчики могут быстро формировать основные элементы проекта, основываясь на уже устоявшихся подходах;  
4) поддерживает моделирование в свободной форме, с различными настройками под предметную область, с учетом традиций компании­разработчика;  
5) имеет высокоэффективные механизмы поиска определенных моделей или их свойств, что позволяет легко ориентироваться в сложных системах;  
6) поддерживает одновременную совместную работу разных моделей с помощью общих представлений и различных связей;  
7) поддерживает все возможности по управлению версиями.

Развитие возможностей вышеперечисленных объектно-ориентированных CASE-средств можно увидеть на примере инструмента Enterprise Architect. К его основным характеристикам относятся следующие:

1. Продукт представляет единый процесс проектирования и моделирования. Это позволяет на всех этапах, начиная от бизнес- моделирования и заканчивая кодированием, включая тестирование и создание документации, использовать одну и ту же модель. При этом все разработчики могут находиться в одной среде, не мешая друг другу.

2. Средство имеет возможность проводить прямое и обратное проектирование, а также синхронизацию кода и модели, что позволяет структурировать проект любого типа сложности.

3. В среде можно создавать шаблоны проектирования, используя которые разработчики могут быстро формировать основные элементы проекта, основываясь на уже устоявшихся подходах.

4. Инструмент имеет эффективные механизмы поиска определенных моделей или их свойств, что позволяет легко ориентироваться в сложных системах.

5. Продукт поддерживает моделирование в свободной форме с различными настройками под предметную область с учетом стандартов и традиций компании-разработчика.

6. Enterprise Architect предоставляет средства для управления проектом. Менеджеры проекта могут его использовать, чтобы связывать ресурсы с элементами, определять риски, объем работ и оценивать размер проекта.

7. Средство позволяет компилировать, отлаживать, тестировать, запускать и выполнять скрипты, доступные в данной среде разработки.

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

Этот вид диаграмм 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 бизнес­процессов автоматизируемой предметной области, определении требований к будущей программной системе.

После активизации диаграммы Use Case соответствующая ей строка инструментов по умолчанию состоит из десяти значков, главными из которых являются ­ Use Case (варианты использования) и ­ Actor (действующие лица). Use Case и Actor объединяются при помощи соответствующих связей.

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.

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.

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

Deployment diagram (диаграмма развёртывания)

Этот вид диаграмм предназначен для анализа аппаратной части системы, то есть «железа», а не программ. В прямом переводе с английского Deployment означает «развертывание», но также может переводиться как «топология».

Для каждой модели создается только одна такая диаграмма, отображающая процессоры (Processor), устройства (Device) и их соединения.

Обычно этот тип диаграмм используется в самом начале проектирования системы для анализа аппаратных средств, на которых она будет эксплуатироваться.

При создании диаграммы Deployment строка инструментов насчитывает 7 кнопок, главными из которых являются Device, Processor и Connection, которые объединяют компоненты диаграммы.

Разработка диаграммы развертывания начинается с идентификации всех аппаратных, механических и других типов устройств, которые необходимы для выполнения системой всех своих функций. В первую очередь специфицируются вычислительные узлы системы, обладающие памятью и/или процессором. При этом используются имеющиеся в языке UML стереотипы, а в случае отсутствия последних, разработчики могут определить новые стереотипы. Отдельные требования к составу аппаратных средств могут быть заданы в форме ограничений, свойств и помеченных значений.

Дальнейшее построение диаграммы развертывания связано с размещением всех исполняемых компонентов диаграммы по узлам системы. Если отдельные исполняемые компоненты оказались не размещенными, то подобная ситуация должна быть исключена введением в модель дополнительных узлов, содержащих процессор и память.

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

Состояние (state) ­ ситуация в жизненном цикле объекта, во время которой он удовлетворяет некоторому условию, выполняет определенную деятельность или ожидает какого­то события. Состояние объекта определяется значениями некоторых его атрибутов и присутствием или отсутствием связей с другими объектами.

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.

Диаграмма состояний полезна при моделировании жизненного цикла объекта. От других диаграмм диаграмма состояний отличается тем, что описывает процесс изменения состояний только одного экземпляра определенного класса ­ одного объекта, причем объекта реактивного, то есть объекта, поведение которого характеризуется его реакцией на внешние события.

Имеется ряд обозначений. Скругленные прямоугольники представляют состояния, через которые проходит объект в течение своего жизненного цикла. Стрелками показываются переходы между состояниями, которые вызваны выполнением методов описываемого диаграммой объекта. Существует также два вида псевдосостояний: начальное, в котором находится объект сразу после его создания (обозначается сплошным кружком), и конечное, которое объект не может покинуть, если перешел в него (обозначается кружком, обведенным окружностью).

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

Данный тип диаграмм 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. Диаграммы деятельности могут создаваться на всех стадиях разработки ПО. Перед началом работ с их помощью моделируются важные рабочие процессы предметной области, с целью определения структуры и динамики бизнеса. На этапе обсуждения требований диаграммы деятельности используются для описания процессов и событий выявленных прецедентов. В течение фазы анализа и проектирования Activity помогает моделировать процесс операций объектов. Activity обозначает выполнение определенных действий в течение жизни объекта. State обычно обозначает ожидание какого­ либо события. Деятельности соединяются на диаграмме стрелками – State Transition (переход состояния), кроме этого, он показывает получение и обработку сообщения объектом.

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

Диаграмма Sequence позволяет получить отражение процесса обмена сообщениями во времени. В течение работы системы объекты, являющиеся клиентами, посылают друг другу различные сообщения, а объекты­серверы обрабатывают их. В простейшем случае можно рассматривать сообщение как вызов метода какого­либо класса, в более сложных случаях сервер имеет обработчик очереди сообщений, и сообщения им обрабатываются асинхронно.

Рассмотрим построение диаграммы Sequence.

– Object (объект) позволяет включить объект в диаграмму. Каждый из них является реализацией какого­нибудь класса, поэтому в объекте можно указать соответствующий ему класс.   
– Message (сообщение) предназначен для передачи сообщения от одного объекта к другому. Классы должны позволять отправку или прием сообщений.

– 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.

Таким образом, можно настроить следующие параметры жизни объекта:

● Persistent – область видимости объекта превышает время жизни.

● Static – элемент существует на всем протяжении работы программы.

● Transient – время жизни объекта и области видимости совпадают.

Горизонтальные линии,проведенные от объекта к объекту,означают передачу сообщений между ними. Свойства задаются в окне спецификации с помощью двух групп кнопок.

1. Synchronization определяет порядок обмена сообщениями например:

● Simple – простая посылка сообщения;

● Synchronous – операция происходит только в том случае, когда клиент посылает сообщение, а сервер может принять сообщение клиента;

● Asynchronous – клиент выдает сообщение, и, не ожидая ответа сервера, продолжает выполнение своего программного кода.

и др.

2. Frequency определяет частоту обмена сообщениями:

● Periodic – сообщения поступают от клиента с заданной периодичностью;

● Aperiodic – сообщения поступают от клиента нерегулярно.

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

Collaboration ­ диаграмма кооперации. Она акцентирует внимание на структуре обмена сообщениями между отдельными объектами. 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.

Главная особенность диаграммы кооперации заключается в возможности графически представить не только последовательность взаимодействия, но и все структурные отношения между объектами, участвующими в этом взаимодействии.

Прежде всего, на диаграмме кооперации в виде прямоугольников изображаются участвующие во взаимодействии объекты, содержащие имя объекта, его класс и, возможно, значения атрибутов. Далее, как и на диаграмме классов, указываются ассоциации между объектами в виде различных соединительных линий.

В отличие от диаграммы последовательности, на диаграмме кооперации изображаются только отношения между объектами, играющими определенные роли во взаимодействии. На этой диаграмме не указывается время в виде отдельного измерения. Поэтому последовательность взаимодействий и параллельных потоков может быть определена с помощью порядковых номеров. Следовательно, если необходимо явно специфицировать взаимосвязи между объектами в реальном времени, лучше это делать на диаграмме последовательности.

**20.Принцип модульности. Декомпозиция системы на подсистемы и модули.**

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.

Декомпозиция системы – возможность представления проекта в виде удобных для обработки частей-модулей.

Мо́дульное программи́рование — это организация программы как совокупности небольших независимых блоков, называемых модулями, структура и поведение которых подчиняются определенным правилам. Использование модульного программирования позволяет упростить тестирование программы и обнаружение ошибок.

Мо́дуль — функционально законченный фрагмент программы.

Принцип модульности является средством упрощения задачи проектирования программного обеспечения (ПО) и распределения процесса разработки между группами разработчиков. При разбиении ПО на модули для каждого модуля указывается реализуемая им функциональность, а также связи с другими модулями. Удобство использования модульной архитектуры заключается в возможности обновления (замены) модуля, без необходимости изменения остальной системы.

Роль модулей могут играть структуры данных, библиотеки функций, классы, сервисы и др. программные единицы, реализующие некоторую функциональность и предоставляющие интерфейс к ней.

Одним из методов написания модульных программ является объектно-ориентированное программирование. ООП обеспечивает высокую степень модульности благодаря таким свойствам, как инкапсуляция, полиморфизм и позднее связывание.

**21.Связность модуля. Типы связностей.**

Связность, или прочность (англ. cohesion, module strength), — мера силы взаимосвязанности элементов внутри модуля; способ и степень, в которой задачи, выполняемые некоторым программным модулем, связаны друг с другом.

**Типы связности:**  
*Случайная* связность. Тип связности, при котором задачи, выполняемые программным модулем, не имеют функциональной взаимосвязи друг с другом.

*Коммуникационная* связность. Тип связности, при котором задачи, выполняемые программным модулем, используют одни и те же входные данные или участвуют в формировании одних и тех же выходных данных.

*Функциональная* связность. Тип связности, при котором все задачи, выполняемые программным модулем, вносят вклад в выполнение одной и той же функции.

*Логическая* связность. Тип связности, при котором задачи, программным модулем, реализуют логически сходные функции (например, одинаково обрабатывают разные типы входных данных).

*Процедурная* связность. Тип связности, при котором все задачи, выполняемые программным модулем, участвуют в некоторой программной процедуре.

*Последовательностная* связность. Тип связности, при котором выходные данные одной задачи, выполняемой программным модулем, служат входным данными для другой задачи, выполняемой этим же модулем.

*Временна́я* связность. Тип связности, при котором все задачи, выполняемые программным модулем, требуются для некоторой фазы выполнения программы (например, модуль содержит все задачи для инициализации).

**22.Сцепление модулей. Типы сцеплений.**

Зацепление, сцепление (англ. coupling) — способ и степень взаимозависимости между программными модулями; сила взаимосвязей между модулями; мера того, насколько взаимозависимы разные подпрограммы или модули.

Сильное зацепление рассматривается как серьезный недостаток, поскольку затрудняет понимание логики модулей, их модификацию, автономное тестирование, а также переиспользование по отдельности. Слабое зацепление, напротив, является признаком хорошо структурированной и хорошо спроектированной системы, и, когда оно комбинируется с сильной связностью, соответствует общим показателям хорошей читаемости и сопровождаемости.

**Типы зацепления:**

*Зацепление по общей области*. Тип зацепления, при котором два программных модуля совместно используют общую область данных.

*Зацепление по содержимому*. Тип зацепления, при котором некоторые или все программные модули включены в некоторый модуль как составные части.

*Зацепление по управлению*. Тип зацепления, при котором один программный модуль обменивается данными с другим модулем с явной целью повлиять на его последующее выполнение.

*Зацепление по данным*. Тип зацепления, при котором выходные данные одного программного модуля служат входными данными другого модуля.

*Смешанное зацепление*. Тип зацепления, при котором различные подмножества значений некоторого элемента данных используются в нескольких программных модулях для разных и несвязанных целей.

*Патологическое зацепление*. Тип зацепления, при котором один программный модуль зависит от деталей внутренней реализации другого модуля или влияет на них.

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

**Диаграмма компонентов** описывает особенности физического представления системы. Она позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых может выступать исходный и исполняемый код.

Основными графическими элементами диаграммы компонентов являются:

* + компоненты
  + интерфейсы
  + зависимости между ними.

Диаграмма компонентов разрабатывается для следующих **целей**:

* + визуализации общей структуры исходного кода программной системы;
  + спецификации исполняемого варианта программной системы;
  + обеспечения многократного использования отдельных фрагментов программного кода;
  + представления концептуальной и физической схем баз данных.

Для представления физических сущностей в языке UML применяется специальный термин ­ **компонент**. Компонент реализует некоторый набор интерфейсов и служит для общего обозначения элементов физического представления модели.

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

Диаграмма классов – основная для создания кода приложения. С ее помощью строится внутренняя структура системы. Обычно данная диаграмма строится для всех классов, становясь логической моделью системы.

Спецификация класса (вкладки):

- General: задаются главные свойства класса: имя, тип, стереотип и доступ к нему, когда класс находится в пакете, а также документация к классу.

- Detail позволяет указать дополнительные характеристики класса:

­ Components: отражает компоненты, с которыми ассоциирован класс.

- Attributes: позволяет добавлять, удалять, редактировать атрибуты класса. Можно задать тип хранения атрибута в классе: By Value – по значению; By Reference – по ссылке; Unspecified – не указано.

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

- Operations: предназначена для добавления, удаления, редактирования операции класса.

- Relations: позволяет добавлять, удалять, редактировать связи класса.

Назначение и виды связей в диаграмме Class   
однонаправленная ассоциация (один класс включается в другой как атрибут по ссылке или по значению);  
Aggregate – один класс содержит другой.

зависимость (один класс использует объекты другого. Это может осуществляться при передаче параметров или вызове операций класса);

ассоциированный класс (отображение свойства ассоциации; предназначена для задания дополнительных атрибутов у связи; обозначает, что некоторый класс со своими атрибутами включается как элемент в два других);

наследование (один класс является родительским по отношению к другому);

реализация (один класс является реализацией, т.е. создан на основе шаблона другого; используется понятие параметризованный класс).

**25.Понятие шаблонов проектирования и их классификация. Шаблоны в нотации языка UML.**

Встречаются ситуации, когда решение проблем в различных проектах имеют сходные структурные черты. Классификация паттернов по категориям их применения:

*Архитектурные* паттерны­ множество предварительно определенных подсистем со спецификацией их ответственности, правил и базовых принципов установления отношений между ними.

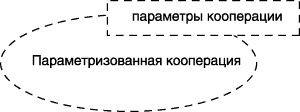
Паттерны *проектирования* ­ специальные схемы для уточнения структуры подсистем или компонентов программной системы и отношений между ними. Описывают общую структуру взаимодействия элементов программной системы

Паттерны *анализа* ­ специальные схемы для представления общей организации процесса моделирования. Относятся к одной или нескольким предметным областям и описываются в терминах предметной области. В дальнейшем паттерны анализа конкретизируются в типовых моделях с целью выполнения аналитических оценок или имитационного моделирования бизнес­процессов.

Паттерны *тестирования* ­ специальные схемы для представления общей организации процесса тестирования программных систем. К нему относятся: тестирование черного ящика, белого ящика, отдельных классов, системы.  
Паттерны *реализации* ­ совокупность компонентов и других элементов реализации, используемых в структуре модели при написании программного кода.

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

При изображении паттерна используется обозначение параметризованной кооперации языка UML, которая обозначается пунктирным эллипсом. В правый верхний угол эллипса встроен пунктирный прямоугольник, в котором перечислены параметры кооперации, которая представляет тот или иной паттерн.



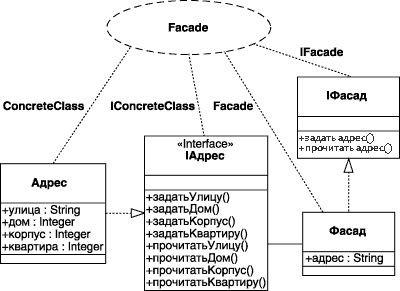
**26.Шаблон “Фасад” и его обозначение в нотации языка UML**

*Паттерн* Фасад предназначен для замены нескольких разнотипных интерфейсов доступа к определенной подсистеме некоторым унифицированным интерфейсом.

Пунктирная линия со стрелкой в форме треугольника служит для обозначения отношения реализации. Пример для выполнения операций по заданию и считыванию адресов из *базы данных* сотрудников.

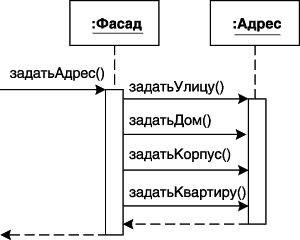
Фрагмент соответствующей *диаграммы классов* содержит 2

класса: *Адрес* и *интерфейс* к операциям этого класса IАдрес. Очевидно, отслеживать при каждом обращении правильность выполнения этих последовательностей операций неудобно. С этой целью к данному фрагменту следует добавить еще один *интерфейс*, реализацию паттерна Фасад для рассматриваемой ситуации. Соответствующий фрагмент модифицированной *диаграммы классов* будет содержать 4 класса, изображенные таким образом, чтобы иллюстрировать реализацию параметрической кооперации.



При задании адреса нового сотрудника в этом случае достаточно обратиться к интерфейсу IФасад и выполнить единственную операцию: задатьАдрес(), используя в качестве аргумента идентификационный номер нового сотрудника.

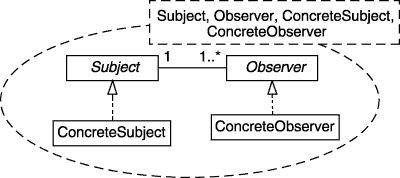
Для получения информации об адресе сотрудника также достаточно обратиться к этому интерфейсу и выполнить единственную операцию: прочитатьАдрес(), используя в качестве аргумента идентификационный номер интересующего сотрудника. Реализацию данных операций следует предусмотреть в классе Фасад. Взаимодействие объектов этих классов может быть представлено с помощью диаграммы последовательности



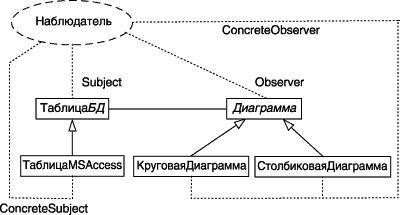
Фасад обеспечивает для клиента не только простоту доступа к информации об адресах, но и независимость представления объектов класса *Адрес* от запросов клиентов. Это актуально при изменении формата представления информации или смене соответствующей *базы данных*. В этом случае потребуется внести изменения только в реализацию операций класса Фасад.

**27. Шаблон “Наблюдатель” и его обозначение в нотации языка UML**

Паттерн Наблюдатель предназначен для контроля изменений состояния объекта и передачи информации об изменении этого состояния множеству клиентов. В общем случае паттерн Наблюдатель также может быть изображен в виде параметризованной кооперации



Пример для отслеживания изменений в таблице БД и отражении этих изменений на диаграммах. Для определенности можно использовать таблицу БД MS Access и две диаграммы ­ круговую и столбиковую. Фрагмент соответствующей диаграммы классов содержит 5 классов



В этом случае за субъектом Таблицей MS Access может "следить" произвольное число наблюдателей, причем их добавление или удаление не влияет на представление информации в БД. Класс Таблица MS Access реализует операции по отслеживанию изменений в соответствующей таблице, и при их наличии сразу информирует абстрактного наблюдателя. Тот в свою очередь вызывает операциипо перерисовке соответствующих диаграмм у конкретных наблюдателей, в качестве которых выступают классы Круговая Диаграммаи Столбиковая Диаграмма.

Использование паттерна Наблюдатель упрощает взаимодействие между объектами соответствующих классов и позволяет вносить изменения в реализацию операций классов субъекта и наблюдателей независимо друг от друга.

**28. Создание модели предметной области с помощью диаграммы классов.**

Диаграмма классов — диаграмма, демонстрирующая классы системы, их атрибуты, методы и взаимосвязи между ними.

Входит в UML. *Классом* называется именованное описание совокупности объектов с общими атрибутами, операциями, связями и семантикой. *Атрибутом* класса называется именованное свойство класса, описывающее множество значений, которые могут принимать экземпляры этого свойства. *Методом* класса называется именованная услуга, которую можно запросить у любого объекта этого класса.

Для описания модели предметной области с помощью диаграммы классов используется Концептуальная точка зрения. В этой модели присутствуют только классы прикладных объектов.

Концептуальная точка зрения используется на начальных этапах моделирования и разработки. В концептуальной точке зрения диаграммы классов служат для представления понятий изучаемой предметной области. Эти понятия будут соответствовать реализующим их классам, но прямое соответствие может отсутствовать. Концептуальная модель может иметь слабое отношение или вообще не иметь никакого отношения к реализующему ее программному обеспечению, поэтому ее можно рассматривать без привязки к какому­то языку программирования. Диаграмма Классов будет отображать классы без атрибутов и операций, а также отношений между классами. Эти диаграммы полезны, когда проект находится в стадии анализа.

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

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.

Диаграмма Class выполняет целый ряд функций: используется для создания иерархии классов; на ее основе строятся модели данных, и проектируется структура Web приложений; на этапе анализа и проектирования используется для создания диаграмм реализации прецедентов; с ее помощью создается модель предметной области, которая используется на этапе анализа.

Для создания модели используются три стереотипа классов, которые определяют их назначение: граничный класс (boundary), сущность (entity), управление (control).

Стереотип *граничный класс* показывает, что класс предназначен для взаимодействия с внешними актерами и стоит на границе системы, поэтому и называется граничным. Такой класс, получая сообщение от внешнего актера, транслирует их внутрь системы, генерируя и передавая соответствующие сообщения другим классам.

Классы *сущности* используются для моделирования классов, которые отвечают за хранение определенной информации. Эти классы реализуют возможности по получению, изменению и сохранению информации в базе данных.

Классы *управления* используются для координации работы других классов приложения. Поведение этих классов обычно реализует один или несколько прецедентов, показанных на диаграммах моделирования.

**30.Декомпозиция системы. Правила выделения подсистем.**

Метод проектирования удовлетворяет критерию Декомпозиции, если он помогает разложить задачу на несколько менее сложных подзадач, объединяемых простой структурой, и настолько независимых, что в дальнейшем можно отдельно продолжить работу над каждой из них.

Такой процесс часто будет циклическим, поскольку каждая подзадача может оказаться достаточно сложной и потребует дальнейшего разложения.

Следствием требования декомпозиции является разделение труда (division of labor): как только система будет разложена на подсистемы, работу над ними следует распределить между разными разработчиками или группами разработчиков. Это трудная задача, так как необходимо ограничить возможные взаимозависимости между подсистемами:

[x]. Необходимо свести такие взаимозависимости к минимуму; в противном случае разработка каждой из подсистем будет ограничиваться темпами работы над другими подсистемами.

[x]. Эти взаимозависимости должны быть известны: если не удастся составить перечень всех связей между подсистемами, то после завершения разработки проекта будет получен набор элементов программы, которые, возможно, будут работать каждая в отдельности, но не смогут быть собраны вместе в завершенную систему, удовлетворяющую общим требованиям к исходной задаче.

Наиболее очевидным примером, удовлетворяющим критерию декомпозиции, является метод *нисходящего* (сверху вниз) проектирования (top-down design). В соответствии с этим методом разработчик должен начать с наиболее абстрактного описания функции, выполняемой системой. Затем последовательными шагами детализировать это представление, разбивая на каждом шаге каждую подсистему на небольшое число более простых подсистем до тех пор, пока не будут получены элементы с настолько низким уровнем абстракции, что становится возможной их непосредственная реализация. Этот процесс можно представить в виде дерева.

**31.Особенности создания шаблона приложения в среде Rational Rose с использованием библиотеки MFC. Структура и классы приложения.**

Мастер создания приложений VC++ (AppWizard) может строить несколько типов приложений:

Single document – приложение работает с одним документом;  
Multiple document – приложение работает с несколькими документами;

Dialog based – приложение основано на окне диалога.

Для приложения, работающего с одним документом, мастер строит код следующих классов:

главный класс приложения C\*\*\*App;  
класс документа C\*\*\*Doc, в котором должна проходить обработка данных ;

класс просмотра C\*\*\*View отображающий данные на экране компьютера ;

класс для окна «О программе» CAboutDlg;  
класс основного окна программы CMainFrame.

Все приложения VC++ MFC являются объектами. Поэтому приложение – это главный класс, который включает в себя все необходимые для работы классы. Таким образом, на основе стандартных классов документа, предоставляемых MFC, строится приложение, в котором необходимо будет только добавить функциональность

Ассоциация проекта Rational Rose с проектом VC++ выполняется аналогично тому, как это делалось для одного класса. С помощью Component Assigment Tool в строку VC++ перетаскиваются все необходимые классы. Для простоты работы все классы, для которых необходимо создание исходного кода, помещаются в один компонент, заключенный в проект \*\*\*. Библиотека классов MFC импортируется в модель путем следующих действий: Menu:Tools=>Visual C++=>Quick Import MFC 6.0. Затем в модель Rational Rose можно загрузить перечисленные ранее классы, созданные в VC++. Для того чтобы они появились в проекте, необходимо обновить проект по готовому коду, выполнив действия: Menu:Tools => Visual C++=> Update Model From Code. После обновления в модели Rational Rose и в проекте VC++ содержатся одинаковые наборы классов. Теперь можно запустить Visual Studio, перейти в нужный проект, откомпилировать его и получить результат работы программы.

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

*Component Assignment Tool*

Это окно предоставляет возможность создания новых компонентов в модели, ассоциации компонентов с проектами на конкретных языках программирования и назначения классов в компоненты. При этом созданные компоненты будут содержать всю необходимую информацию для генерации кода на выбранном языке программирования. Данное средство позволяет просмотреть классы, которые еще не назначены в компоненты, что уменьшает вероятность ошибки. Component Assignment Tool может быть открыт как посредством меню Tools, так и из контекстного меню компонента в окне Code Update Tool.

*Model Assistant* позволяет обновлять и конкретизировать классы в модели, используя дополнительные ключевые слова C++ для необходимой генерации кода. Model Assistant представляет собой окно, позволяющее создавать атрибуты и операции и изменять их свойства, причем значительно проще и нагляднее, чем это происходит для C++.

На языке Visual С++ программа Model Assistant Tool может создавать и определять операции для классов, такие как конструкторы, деструкторы и операции доступа для атрибутов и отношений.

В поле Preview (Предварительный просмотр) отображается код, который будет получен для выбранного элемента. Это позволяет увидеть, как повлияют на код настройки параметров генерации.

Возвратное проектирование

После внесения изменений и дополнений в код на Visual С++ необходимо обновить модель, чтобы отразить в ней сделанные изменения. Это выполняется с помощью мастера Model Update Tool. Его можно использовать и для создания новой модели по исходному коду.

Если в проект добавлены классы, например, при помощи Class Wizard, и они еще не отражены в модели Rational Rose. В этом случает необходимо провести обновление кода при помощи функции Update Code (обновить код).

По завершении возвратного проектирования мастер Model Update Tool отображает диалоговое окно с итоговым отчетом. Вкладка Summary содержит сведения об обновленной модели, а все ошибки, возникшие при обновлении, можно увидеть на вкладке Log.

**33.Особенности генерации исходного кода в среде Rational XDE. Способы синхронизации модели.**

Генерация по готовым диаграммам сокращает количество ошибок на стадии преобразования готовой модели в программный код. В любой программе, написанной при помощи среды Microsoft .NET, можно провести обратное проектирование и разобраться в архитектуре, представленной графически в виде иерархии классов.

Rational XDE предоставляет большое количество настроек для автоматической и ручной генерации кода приложения. Для работы с исходным кодом в 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 одним­двумя 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.

Для настройки параметров автоматической синхронизации модели необходимо выбрать из главного меню пункт:

Tools => Options => Rational XDE => Round Trip Engineering

=> Synchronization Setting.

После выбора эл­тов в 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 – синх. происх. в момент сохр. кода после его изменения;

– When Code gets Focus – синх. происходит в момент активизации окна кода.

**34.Сравнительный анализ процедур генерации исходного кода в Rational Rose и Rational XDE и Enterprise Architect.**

Дело в том, что традиционные решения на базе Rationаl Rose позволяют проводить прямое и обратное проектирование кодов для многих языков программирования, т. е. возможна прямая генерация кода, обратная (так называемый реверс­инжиниринг) и постоянная модификация кода и модели с поддержанием их в актуальном состоянии (round­trip engineering). Однако применение этих инструментов ограничивалось тем, что они не допускали генерацию в реальном масштабе времени: код можно было получать, но при этом нужно было выполнить определенные дополнительные действия по встраиванию его в систему разработки. Кроме того, проблемы возникали и при постановке сгенерированных (перегенерированных) модулей под версионное управление.

Новый инструмент XDE Developer решает эти задачи, реализуя функцию поддержания кода и модели в актуальном состоянии при работе в реальном масштабе времени. Таким образом, нарисовав диаграмму классов с иерархической структурой, можно тут же сгенерировать код со всеми связями, атрибутами и методами. XDE работает со всеми типами диаграмм из UML 1.3. В то же время XDE Developer пока не покрывает полностью функциональность Rose, которая, в частности, предоставляет возможность работы с более широким кругом языков программирования, обладает более развернутыми функциями проектирования данных.

Щелкнув правой кнопкой мыши на конкретной модели и выбрав в контекстном меню пункт разработка кода, я могу импортировать весь исходный каталог проекта и генерировать диаграммы классов для всех объектов проекта. Я также могу просто создать диаграмму классов для одного исходного файла.

**35.Назначение, возможности, особенности использования модуля Web Modeler в Rational Rose.**

Web­Modeler обеспечивает возможность строить диаграммы классов при проектировании WEB­приложений с учетом специфики WEB­программирования (классами являются страницы и скрипты, предусмотрены отношения передачи данных из формы, генерации страницы скриптом и т,д,).

Дает возможность для предусмотренных платформ ASP, JSP:

- построить модель классов по коду;  
- создать код по модели классов.

Для того чтобы начать работу с Web Modeler, необходимо установить его при помощи Add­Ins=>Add­Ins Manager=>Rose Web Modeler=ON.

После этого в меню Tools появится новый пункт Web Modeler. В подменю Web Modeler есть 2 подпункта:

1. «Установки пользователя» служит для настройки генератора кодов при построении кода по модели. Установки позволяют настроить действия генератора для тегов HTML и скриптов.

2. «Reverse Engineer a New Web Application(Построение модели по коду)» позволяет создать модель по коду. При его активизации запускается мастер Reverse Engineer. Мастер позволяет задать платформу ASP (Active Server Pages) или JSP(Java Server Pages) и каталог, в котором находится исходный код.

Для моделирования Web приложения используется диаграмма классов. Классы имеют специальные стереотипы, которые влияют не только на изображение класса на диаграмме, но и дополняют контекстное меню класса еще одной строкой спецификаций и пунктом Web Modeler. Создавать классы необходимо в Web нотации.

Для того, чтобы создавать классы в Web нотоции: Menu:Tools=> Oplions=>Notation=>Default Language=Web Notation.

**36.Возможности и особенности построения Web­модели в среде Rational XDE.**

В Rational XDE для создания структуры системы, ориентированной на работу в Web, используется диаграмма классов, в которой при создании архитектуры приложения учитываются ограничения реализации Web­приложения.

Рассмотрим возможности Rational XDE по созданию Web­модели.

Client Page позволяет создать на диаграмме отображение простых страниц HTML, не имеющих собственного поведения. Обычно такие страницы предоставляют пользователям определенную, заранее заданную информацию.

Link Relation позволяет отобразить связи между страницами в том случае, когда на одной странице есть ссылка на другую. Rational XDE не знает, куда добавлять созданную ссылку, и вставляет ее в конец файла.

HTML Form позволяет отобразить формы ввода, которые присутствуют на страницах HTML. Форма не может существовать сама по себе, она включается на страницу при помощи агрегирования. Поэтому для ее разработки начинают с создания страницы, на которой будет находиться форма.

При помощи Server Page with Code­Behind создается набор элементов, связанных между собой и содержащих необходимые элементы для создания ASP.NET приложения.

Для отражения передачи управления другой странице используется NETTransfer Relation. При генерации кода создается директива Transfer, которая позволяет передавать управление другой странице с сохранением доступа к внутренним объектам исходной страницы.

Для отражения простой переадресации с одной страницы на другую используется связь при помощи значка Redirect Relation.

**37.Понятие проектных рисков. Действия по управлению рисками.**

Проектный риск – это осознанная и зафиксированная возможность возникновения ущерба и потерь.

Управление рисками — процесс принятия и выполнения управленческих решений, направленных на снижение вероятности возникновения неблагоприятного результата и минимизацию возможных потерь проекта, вызванных его реализацией.

Управление риском включает шесть действий:  
1. *Идентификация риска* — выявление элементов риска в проекте.

2. *Анализ риска* — оценка вероятности и величины потери по каждому элементу риска.

3. *Ранжирование риска* — упорядочение элементов риска по степени их влияния.

4. *Планирование управления риском* — подготовка к работе с каждым элементом риска.

5. *Разрешение риска* — устранение или разрешение элементов риска.

6. *Наблюдение риска* — отслеживание динамики элементов риска, выполнение корректирующих действий.

Первые три действия относят к этапу оценивания риска, последние три действия — к этапу контроля риска.

Выделяют три категории источников риска: проектный риск, технический риск, коммерческий риск.

Источниками проектного риска являются: выбор бюджета, плана, человеческих ресурсов программного проекта; формирование требований к программному продукту; сложность, размер и структура программного проекта; методика взаимодействия с заказчиком.

После идентификации элементов риска следует количественно оценить их влияние на программный проект, решить вопросы о возможных потерях. Эти вопросы решаются на шаге анализа риска.

В ходе *анализа* оценивается вероятность возникновения Рi и величина потери Li для каждого выявленного i­го элемента риска. В результате вычисляется влияние REi i­го элемента риска на проект.

Вероятности определяются с помощью экспертных оценок или на основе статистики, накопленной за предыдущие разработки. Итоги анализа сводятся в таблицу, которая содержит след. поля: элемент риска, вероятность, потери, влияние риска)

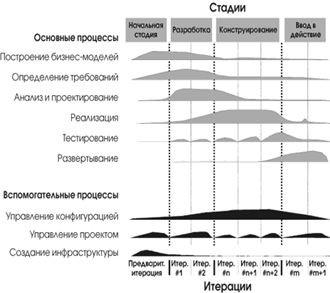
*Ранжирование* заключается в назначении каждому элементу риска приоритета, который пропорционален влиянию элемента на проект. Это позволяет выделить категории элементов риска и определить наиболее важные из них.

*Планирование управления* рисков производится исходя из информации, полученной на этапе их анализа, и имеет своей целью выработку стратегий, планов и конкретных шагов.

*Мониторинг* рисков производится для наблюдения за конкретными рисками и прогрессом в осуществлении составленных планов. Мониторингу должны быть подвергнуты сделанные оценки вероятности риска, его угрозы, ожидаемая величина риска и прочие факторы, способные повлиять на приоритет рисков.

**38.Статический и динамический аспекты Rational Unified Process (RUP).**

На рисунке показано общее представление RUP в двух измерениях. Горизонтальное измерение представляет время, отражает динамические аспекты процессов и оперирует такими понятиями, как стадии, итерации и контрольные точки. Вертикальное измерение отражает статические аспекты процессов и оперирует такими понятиями, как виды деятельности (технологические операции), рабочие продукты, исполнители и дисциплины (технологические процессы).



Для крупных проектов начальная стадия может вылиться во всестороннее изучение всех возможностей реализации проекта, которое займет месяцы. Во время начальной стадии вырабатывается бизнес-план проекта ­ определяется, сколько приблизительно он будет стоить и какой доход принесет. (результаты стадии: общие описание системы, начальный проектный глоссарий, начальный бизнес-план, 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. (результаты стадии: работающий прототип, уточненный бизнес­план, план разраб. проекта)

Результатом стадии конструирования является продукт, готовый к передаче конечным пользователям.

Назначением стадии ввода в действие является передача готового продукта распоряжение пользователей. (включает:бета­тестирование, конвертирование БД, оптимизация производительности, обучение пользователей)  
Cтатический аспект RUP представлен четырьмя основными элементами:

● роли;

● виды деятельности;

● рабочие продукты;

дисциплины.

**39.Принципы и стадии разработки ПС в технологии RUP**

Rational Unified Process (RUP) — методология разработки программного обеспечения, созданная компанией Rational Software. В основе RUP лежат следующие принципы:

● Ранняя идентификация и непрерывное (до конца проекта) устранение основных рисков.

● Концентрация на выполнении требований заказчиков к исполняемой программе (анализ и построение модели прецедентов (вариантов использования)).

● Ожидание изменений в требованиях, проектных решениях и реализации в процессе разработки.

● Компонентная архитектура, реализуемая и тестируемая на ранних стадиях проекта.

● Постоянное обеспечение качества на всех этапах разработки проекта (продукта).

● Работа над проектом в сплочённой команде, ключевая роль в которой принадлежит архитекторам.

RUP использует итеративную модель разработки. В конце каждой итерации (~ 2­6 недель) проектная команда должна достичь запланированных на данную итерацию целей, создать или доработать проектные артефакты и получить промежуточную, но функциональную версию конечного продукта. Итеративная разработка позволяет быстро реагировать на меняющиеся требования, обнаруживать и устранять риски на ранних стадиях проекта, а также эффективно контролировать качество создаваемого продукта. Полный жизненный цикл:

1. Начальная стадия (Inception)

● Формируются видение и границы проекта.

● Создается экономическое обоснование (business case).

● Определяются требования, ограничения и ключевая функциональность продукта.

● Создается базовая версия модели прецедентов.

● Оцениваются риски.

При завершении начальной фазы оценивается достижение этапа жизненного цикла цели (англ. Lifecycle Objective Milestone), которое предполагает соглашение заинтересованных сторон о продолжении проекта.

2. Уточнение (Elaboration) Производится анализ предметной области и построение исполняемой архитектуры. Это включает в себя:

● Документирование требований (с детальным описание большинства прецедентов).

● Спроектированную, реализованную и оттестированную исполняемую архитектуру.

● Новое экономическое обоснование и более точные оценки сроков и стоимости.

● Сниженные основные риски.

Успешное выполнение фазы разработки означает достижение этапа жизненного цикла архитектуры (англ. Lifecycle Architecture Milestone).

3. Построение (Construction). Происходит реализация большей части функциональности продукта. Фаза Построение завершается первым внешним релизом системы и вехой начальной функциональной готовности (Initial Operational Capability).

4. Внедрение (Transition). Создается финальная версия продукта и передается от разработчика к заказчику. Это включает в себя программу бета­тестирования, обучение пользователей, а также определение качества продукта. В случае, если качество не соответствует ожиданиям пользователей или критериям, установленным в фазе Начало, фаза Внедрение повторяется снова. Выполнение всех целей означает достижение вехи готового продукта (Product Release) и завершение полного цикла разработки.

**40. Содержание и результаты первой и второй стадий в технологии Rational Unified Process.**

*Первая стадия(начальная стадия)*

Начальная стадия может принимать множество разных форм. Для крупных проектов она связана с всесторонним изучением всех возможностей реализации проекта. В это же время вырабатывается бизнес­план проекта, 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 бизнес­план;  
- план проекта, отражающий стадии и итерации;  
- один или несколько прототипов.

*Вторая стадия(стадия разработки)*

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

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

Сущность планирования заключается в определении последовательности итераций конструирования и вариантов использования, реализуемых на каждой итерации. Итерации на стадии конструирования являются одновременно инкрементными и повторяющимися:

● Каждая итерация добавляет очередные конструкции к вариантам использования, реализованным во время предыдущих итераций;

● На каждой итерации некоторая часть существующего кода переписывается с целью сделать его более гибким.

Результатом стадии конструирования является:

● ПО, интегрированное на требуемых платформах;

● руководства пользователя;

● описание текущей реализации.

Назначением стадии ввода в действие является передача готового продукта в распоряжение пользователей. Данная стадия включает:

● бета­тестирование, позволяющее убедиться, что новая система соответствует ожиданиям пользователей;

● параллельное функционирование с существующей (legacy) системой, которая подлежит постепенной замене;

● конвертирование баз данных;

● оптимизацию производительности;

● обучение пользователей и специалистов службы сопровождения.

**42. Этапы создания ПС в технологии Oracle.**

В технологии Oracle существует понятие Custom Development Method(CDM) – разработка прикладного ПО. Метод CDM оформлен в виде консалтингового продукта CDM Advantage – библиотеки стандартов и руководств. По существу, CDM является методическим руководством по разработке прикладного ПО с использованием инструментального комплекса Oracle Developer Suite, а сам процесс проектирования и разработки тесно связан с Oracle Designer и Oracle Forms.

В соответствии с CDM ЖЦ ПО формируется из определенных этапов (фаз) проекта и процессов:

• стратегия (определение требований);

• анализ (формулирование детальных требований к системе);

• проектирование (преобразование требований в детальные спецификации системы);

• реализация (написание и тестирование приложений);

• внедрение (установка новой прикладной системы, подготовка к началу эксплуатации);

• эксплуатация.

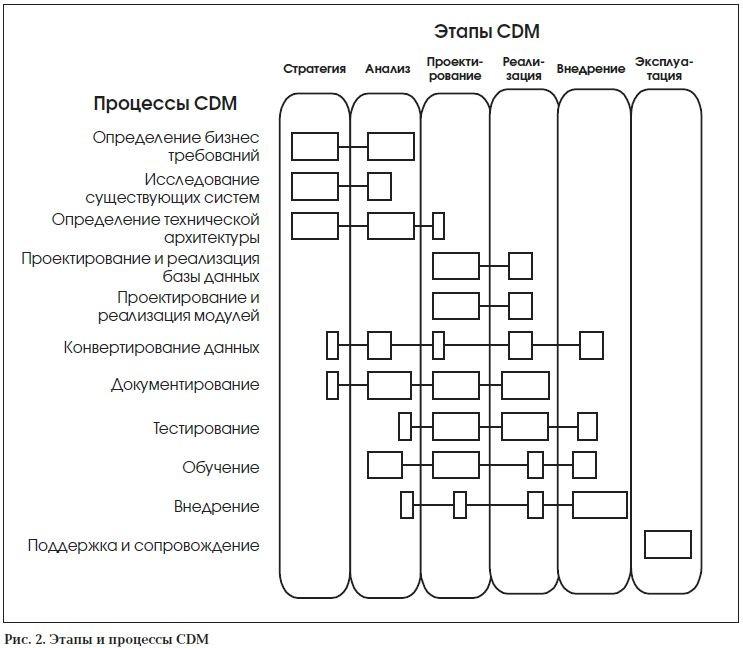
На этапе *стратегии* определяются цели создания системы, приоритеты и ограничения, разрабатывается системная архитектура и составляется план разработки.

На этапе *анализа* строятся модель информационных потребностей (диаграмма «сущность\_связь»), диаграмма функциональной иерархии (на основе функциональной декомпозиции системы), матрица перекрестных ссылок и диаграмма потоков данных.

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.

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.

На этапах *внедрения* и *эксплуатации* анализируются производительность и целостность системы, выполняется поддержка и, при необходимости, модификация системы.



**43. Процессы создания ПС в технологии Oracle.**

Методическую основу ТС ПО корпорации Oracle составляет метод Oracle (Oracle Method) ­ комплекс методов, охватывающий большинство процессов ЖЦ ПО.

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

● стратегия (определение требований);

● анализ (формулирование детальных требований к системе);

● проектирование (преобразование требований в детальные спецификации системы);

● реализация (написание и тестирование приложений);

● внедрение (установка новой прикладной системы, подготовка к началу эксплуатации);

● эксплуатация.   
Процессы CDM:

● определение бизнес­требований, или постановка задачи;исследование существующих систем, определение технической архитектуры;

● проектирование и реализация базы данных, проектирование и реализация модулей.

● конвертирование данных (Data Conversion).

● документирование (Documentation);

● тестирование (Testing);

● обучение (Training);

● внедрение, или переход к новой системе, поддержка и сопровождение.

PJM ­ это определенная дисциплина ведения проекта, позволяющая гарантировать, что цели проекта, четко определенные в его начале, остаются в центре внимания на протяжении всего проекта.

● Управление проектом и предоставление отчетности (Control and Reporting).

● Управление работой (Work Management).

● Управление ресурсами (Resource Management).

● Управление качеством (Quality Management).

● Управление конфигурацией (Configuration Management).

**44. Классический и быстрый подходы к разработке ПС в технологии Oracle. Факторы, определяющие выбор подхода.**

Разработка прикладного ПО(Custom Development method – CDM) предоставляет возможность выбрать требуемый подход к разработке.

При определении подхода к разработке оценивается масштаб, степень сложности и критичность будущей системы. При этом учитываются стабильность требований, сложность и количество бизнес­правил, количество автоматически выполняемых функций, разнообразие и количество пользователей, степень взаимодействия с другими системами, критичность приложения для основного бизнес­процесса компании и целый ряд других.

В соответствии с этими факторами в CDM выделяются два основных подхода к разработке:

*Классический подход* (Classic). Этапы данного подхода ­ стратегия, анализ, проектирование, реализация, внедрение, эксплуатация. Классический подход применяется для наиболее сложных и масштабных проектов, он предусматривает последовательный и детерминированный порядок выполнения задач. Применение классического подхода также рекомендуется при нехватке опыта у разработчиков, неподготовленности пользователей, нечетко определенной задаче. Много времени отводится на стадию анализа и проектирования. Работы ведутся линейно, итеративность не приветствуется. Пользователь привлекается к работам в основном на этапах анализа и внедрения. На протяжении всего проекта генерируется мощный поток документов о ходе работ, призванных убедить заказчика, что работы идут нормально. Продолжительность таких проектов от 8 до 36 месяцев. Но данный подход неэффективен, когда за время выполнения проекта изменяются требования заказчика к функционированию системы и когда невозможно получить постановку задачи от пользователя, предварительно не показав ему, как программа может работать.

Подход *быстрой разработки* (Fast Track). Данный подход, в отличие от каскадного классического, является итерационным и основан на методе разработки динамических систем). Его цель ­ сдать готовый проект вовремя и уложиться в бюджет, но, в то же время, регулируя изменения требований к проекту во время его разработки. В этом подходе четыре этапа – стратегия, моделирование требований, проектирование и генерация системы, и внедрение в эксплуатацию. Непосредственное интенсивное взаимодействие разработчиков с пользователями на протяжении всего проекта. Уменьшение времени на бумажную постановку. Подход используется для реализации небольших и средних проектов с несложной архитектурой системы, гибкими сроками и четкой постановкой задач. Продолжительность проекта от 4 до 16 месяцев. В полном виде неэффективен у нас, т.к. наши ГОСТы также не приветствуют итеративность и к тому же требуют в Техническом проекте описания всех форм и отчетов системы.

**45.Этапы разработки ПС в технологии Borland.**

Компания Borland в результате развития собственных разработок и приобретения целого ряда компаний представила интегрированный комплекс инструментальных средств, реализующих управление полным жизненным циклом приложений (Application Life Cycle Management, ALM). Процесс создания ПО включает в себя пять основных этапов:  
1) определение требований;  
2) анализ и проектирование;  
3) разработка;  
4) тестирование и профилирование;  
5) развертывание.

*Определение требований* реализуется с помощью системы управления требованиями CaliberRM. CaliberRM сохраняет требования в базе данных, документы с их описанием создаются с помощью встроенного механизма генерации документов MS Word на базе заданных шаблонов. Система обеспечивает экспорт данных в таблицы MS Access и импорт из MS Word. CaliberRM поддерживает различные методы визуализации зависимостей между требованиями, с помощью которых пользователь может ограничить область анализа, необходимого в случае изменения того или иного требования. Имеется модуль, который использует данные требования для оценки трудозатрат, рисков и расходов, связанных с реализацией требований.

Средство *анализа и проектирования* Together ControlCenter. В основе его применения лежит один из вариантов подхода "Быстрой разработки ПО" под названием Feature Driven Development (FDD)

Together ControlCenter ­ интегрированная среда проектирования и разработки, поддерживающая визуальное моделирование на UML с последующим написанием приложений для платформ J2EE (Java) и .Net (С#, C++ и Visual Basic).

В системе реализована технология LiveSource, которая обеспечивает синхронизацию между проектом приложения и изменениями. Контроль версий осуществляется благодаря функциональной интеграции Together и системы StarTeam.

В технологии Borland выделяется 3 уровня интеграции. *Функциональная* (touch­point) интеграция позволяет обратиться из одной системы к функциям другой, выбрав соответствующий пункт меню. Такая интеграция дает возможность разделять информацию между системами, но не обеспечивает единого рабочего пространства, вынуждает пользователя переключать окна и приводит к дублированию процессов управления структурой проекта. *Встроенная* (embedded) интеграция обеспечивает работу с одной системой непосредственно в среде другой. Например, не выходя из среды разработки Jbuilder, можно просматривать графики производительности, которые создает система Optimizeit. Самый высокий уровень интеграции ­ *синергетический* (synergistic), позволяющий сочетать функции двух различных продуктов незаметно для разработчиков.

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

RUP использует итеративную модель разработки. В конце каждой итерации (~ 2­6 недель) проектная команда должна достичь запланированных на данную итерацию целей, создать или доработать проектные артефакты и получить промежуточную, но функциональную версию конечного продукта. Итеративная разработка позволяет быстро реагировать на меняющиеся требования, обнаруживать и устранять риски на ранних стадиях проекта, а также эффективно контролировать качество создаваемого продукта.

Методическую основу ТС ПО корпорации Oracle составляет метод Oracle (Oracle Method) ­ комплекс методов, охватывающий большинство процессов ЖЦ ПО. В состав комплекса входят:

● CDM (Custom Development Method) ­ разработка прикладного ПО;

● PJM (Project Management Method) ­ управление проектом;

● AIM (Application Implementation Method) ­ внедрение прикладного ПО;

● BPR (Business Process Reengineering) ­ реинжиниринг бизнес­процессов;

● OCM (Organizational Change Management) ­ управление изменениями, и др.

Компания Borland представила интегрированный комплекс инструментальных средств, реализующих управление полным жизненным циклом приложений (Application Life Cycle Management, ALM). Процесс создания ПО включает в себя пять основных этапов:  
1) определение требований;  
2) анализ и проектирование;  
3) разработка;  
4) тестирование и профилирование;  
5) развертывание.

**47.Понятие CASE-средство, CASE-система, CASE-технология, CASE-индустрия и различия между ними.**

CASE-средство = это инструмент, который позволяет автоматизировать один или несколько этапов процесса разработки информационной системы и программного обеспечения.

CASE-система = совокупность различных, взаимодополняющих друг друга CASE-средств из одной области для решения объёмных задач, которые невозможно решить только одним CASE-средством. Примеры: Rational Rose, EA Architect.

CASE-технология = CASE-система + технические знания по её применению.

CASE-технология включает в себя методологию анализа, проектирования, разработки и сопровождения сложных систем ПО, поддержанную комплексом взаимосвязанных средств автоматизации. CASE-технология - именно то, что мы использовали при написании лабораторных.

CASE-индустрия = применение CASE-технологии для создания коммерческих продуктов и получения прибыли.

**48.Проектирование информационных систем с применением UML. Разработка моделей бизнес­прецедентов и бизнес­объектов**

1. Программы на UML представляются в виде диаграмм, состоящих из объектов и связей между ними или из этапов процесса проектирования. UML обеспечивает поддержку всех этапов жизненного цикла ИС и предоставляет для этих целей ряд графических средств — диаграмм. На этапе создания концептуальной модели для описания бизнес­деятельности используются модели бизнес­прецедентов и диаграммы видов деятельности, для описания бизнес­объектов – модели бизнес­объектов и диаграммы последовательностей.

2. Разработка модели бизнес­прецедентов

Модель бизнес­прецедентов описывает бизнес­процессы с точки зрения внешнего пользователя, т.е. отражает взгляд на деятельность организации извне. Проектирование системы начинается с изучения и моделирования бизнес­деятельности организации. На этом этапе вводится и отображается в модели ряд понятий, свойственных объектно­ориентированному подходу. Исполнитель (Действующее лицо, Actor) ­ личность, организация или система, взаимодействующая с ИС; различают внешнего исполнителя (который использует или используется системой, т.е. порождает прецеденты деятельности) и внутреннего исполнителя (который обеспечивает реализацию прецедентов деятельности внутри системы). На диаграмме исполнитель представляется стилизованной фигуркой человека. Прецедент ­ законченная последовательность действий, инициированная внешним объектом (личностью или системой), которая взаимодействует с ИС и получает в результате некоторое сообщение от ИС. На диаграмме представляется овалом с надписью, отражающей содержание действия.

3. Разработка модели бизнес­объектов

Следующим этапом проектирования ИС является разработка модели бизнес­объектов, которая показывает выполнение бизнес­процессов организации ее внутренними исполнителями. Основными компонентами моделей бизнес­объектов являются внешние и внутренние исполнители, а также бизнес­сущности, отображающие все, что используют внутренние исполнители для реализации бизнес­процессов. Для детального описания выполнения бизнес­процессов обычно используются диаграммы последовательностей. Основными элементами диаграммы последовательностей являются обозначения объектов (прямоугольники), вертикальные линии, отображающие течение времени при деятельности объекта, и стрелки, показывающие выполнение действий объектами.

Результатом этого этапа являются согласованные с заказчиком и достаточно подробные описания действий специалистов организации, внедряющей ИС, необходимые для обеспечения исполнения ее функций.

**49.Проектирование информационных систем с применением UML. Разработка концептуальной модели данных и требований к системе.**

*Разработка концептуальной модели данных*

На основе информации, выявленной на этапах бизнес-моделирования, выполняется разработка концептуальной модели данных, которые будут использоваться в разрабатываемой системе.

Этот этап завершает процедуры бизнес-моделирования и позволяет представить команде

проектировщиков в едином формате ту информацию, которая будет необходима для

создания системы. Разработанные диаграммы являются отправной точкой в процессах

проектирования баз данных и приложений системы, обеспечивают согласованность действий бизнес-аналитиков и разработчиков в процессе дальнейшей работы над системой. Эти диаграммы, конечно же, будут претерпевать изменения в процессе последующего проектирования, однако эти изменения будут фиксироваться в формате, уже привычном для всей команды разработчиков, и будут автоматически отражаться в последующих моделях.

*Разработка требований к системе*

Основные задачи этапа:

1) определить проект системы, который будет отвечать всем бизнес­требованиям;

2) разработать общий предварительный проект для всех команд разработчиков

Основным инструментом на данном этапе являются диаграммы классов системы, которые строятся на основе разработанной модели системных прецедентов. Одновременно на этом этапе уточняются диаграммы последовательностей выполнения отдельных прецедентов, что приводит к изменениям в составе объектов и диаграммах классов. Это естественное отражение средствами UML итеративного процесса разработки системы. Диаграммы классов системы заполняются объектами из модели системных прецедентов. Они содержат описание этих объектов в виде классов и описание взаимодействия между классами.

В результате этого этапа проектирования появляется достаточно подробное описание состава и функций проектируемой системы, а также информации, которую необходимо использовать в базе данных и в приложениях.

Поскольку диаграммы классов строятся на основе разработанных ранее бизнес­моделей, появляется уверенность в том, что разрабатываемая система будет действительно удовлетворять исходным требованиям заказчика.

**50. Проектирование физической реализации информационной системы с применением UML**

На этом этапе проектирования модели баз данных и приложений дополняются обозначениями их размещения на технических средствах разрабатываемой системы.

Основными понятиями UML, которые используются на данном этапе, являются следующие:

● компонент – самостоятельный физический модуль системы;

● зависимость – связь между двумя элементами, при которой изменения в одном элементе вызывают изменения другого элемента;

● устройство – узел, не обрабатывающий данные;

● процессор – узел, выполняющий обработку данных;

● соединение – связь между устройствами и процессорами.

Диаграммы развертывания позволяют отобразить на единой схеме различные компоненты системы (программные и информационные) и их распределение по комплексу технических средств

Разбиения ИС на подсистемы: структурное (или функциональное) разбиение и объектная (компонентная) декомпозиция.

С позиций проектирования ИС суть функционального разбиения может быть выражена известной формулой: " Программа = Данные + Алгоритмы ". При функциональной декомпозиции программной системы ее структура описывается блок­схемами, узлы которых представляют собой "обрабатывающие центры" (функции), а связи между узлами описывают движение данных.

При объектном разбиении в системе выделяются "активные сущности" – объекты (или компоненты), которые взаимодействуют друг с другом, обмениваясь сообщениями и выполняя соответствующие функции (методы) объекта.

Если при проектировании ИС разбивается на объекты, то для ее визуального моделирования следует использовать UML.

Они хорошо приспособлены для решения таких часто возникающих при создании систем задач как: физическое перераспределение вычислений и данных, обеспечение параллелизма вычислений, репликация БД, обеспечение безопасности доступа к ИС, оптимизация балансировки нагрузки ИС, устойчивость к сбоям и т.п. Визуализированные средствами UML модели ИС позволяют наладить плодотворное взаимодействие между заказчиками, пользователями и командой разработчиков. Они обеспечивают ясность представления выбранных архитектурных решений и позволяют понять разрабатываемую систему во всей ее полноте.

**51.Требования к программному обеспечению. Виды требований.**

Требования к программному обеспечению — совокупность утверждений относительно атрибутов, свойств или качеств программной системы, подлежащей реализации. Создаются в процессе разработки требований к программному обеспечению (ПО), в результате анализа требований.

Требования могут выражаться в виде текстовых утверждений и графических моделей.

В классическом техническом подходе совокупность требований используется на стадии проектирования ПО. Требования также используются в процессе проверки ПО, так как тесты основываются на определённых требованиях.

Виды требований по уровням:

- Бизнес-требования — определяют назначение ПО, описываются в документе о видении (vision) и границах проекта (scope).

- Пользовательские требования — определяют набор пользовательских задач, которые должна решать программа, а также способы (сценарии) их решения в системе. Пользовательские требования могут выражаться в виде фраз утверждений, в виде сценариев использования (англ. use case), пользовательских историй (англ. user stories), сценариев взаимодействия (scenario).

- Функциональный уровень (функции).

**52.Формирование требований. Анализ требований.**

На этом этапе команда разработчиков ПО работает с заказчиком и конечными пользователями системы для выяснения области применения, описания системных сервисов, определения режимов работы системы и ее характеристик выполнения, аппаратных ограничений и т.д.

Процесс формирования и анализа требований проходит через ряд этапов.

1. *Анализ предметной области*. Аналитики должны изучить предметную область, где будет эксплуатироваться система.

2. *Сбор требований*. Это процесс взаимодействия с лицами, формирующими требования. Во время этого процесса продолжается анализ предметной области.

3. *Классификация требований*. На этом этапе бесформенный набор требований преобразуется в логически связанные группы требований.

4. *Разрешение противоречий*. Без сомнения, требования многочисленных лиц, занятых в процессе формирования требований, будут противоречивыми. На этом этапе определяются и разрешаются противоречия такого рода.

5. *Назначение приоритетов*. В любом наборе требований одни из них будут более важны, чем другие. На этом этапе совместно с лицами, формирующими требования, определяются наиболее важные требования.

6. *Проверка требований*. На этом этапе определяется их полнота, последовательность и непротиворечивость.

**53.Основы технологии сетевого планирования и управления проектами.**

Под сетевым планированием и управлением (СПУ) принято понимать графическое изображение комплекса взаимосвязанных проектных работ, отражающее их логическую последовательность, взаимозависимость и планируемую продолжительность с целью его использования в оперативном управлении ходом работ при реализации проекта.

Сетевое планирование и управление основывается на двух методах: методе критического пути МКП (СРМ— Critical Path Method) и методе оценки и пересмотра планов ПЕРТ  
(PERT — Program Evaluation and Review Technique).

Планирование и управление в системах СПУ осуществляется с помощью сетевого графика (плана, модели).

Сетевой график (план, модель, сеть) — графическое изображение комплекса взаимосвязанных проектных работ (технологических операций), выполняемых в определенной последовательности.

Основными элементами сетевого графика являются работы (связи) и события, условно изображаемые соответственно стрелками и кружками.

Событие в сетевом графике отображает только факт получения (достижения) результата предшествующей работы (работ) и условие начала следующей за ним работы (работ).

**54. Размерно-ориентированные и функциональные метрики оценивания программных продуктов.**

Размерно-ориентированные метрики прямо измеряют программный продукт и процесс его разработки. Основываются размерно-ориентированные метрики на LOC – оценках (Lines Of Code) – это количество строк в программном продукте.

Исходные данные для расчета этих метрик сводятся в таблицу. На основе таблицы вычисляются размерно-ориентированные метрики производительности и качества проекта.  
Достоинства размерно-ориентированных метрик:

- широко распространены;

- просты и легко вычисляются.

Недостатки размерно-ориентированных метрик:  
- зависимы от языка программирования;  
- требуют исходных данных, которые трудно получить на начальной стадии проекта;

Функционально-ориентированные метрики косвенно измеряют программный продукт и процесс его разработки. Рассматривается не размер, а функциональность или полезность продукта. Используется 5 информационных характеристик.

1. Количество внешних вводов. Подсчитываются все вводы пользователя, по которым поступают разные прикладные данные.

2. Количество внешних выводов. Подсчитываются все выводы, по которым к пользователю поступают результаты, вычисленные программным приложением. В этом контексте выводы означают отчеты, экраны, распечатки, сообщения об ошибках.

3. Количество внешних запросов. Под запросом понимается диалоговый ввод, который приводит к немедленному программному ответу в форме диалогового вывода. При этом диалоговый ввод в приложении не сохраняется, а диалоговый вывод не требует выполнения вычислений.

4. Количество внутренних логических файлов. Подсчитываются все логические файлы (логические группы данных, которые могут быть частью базы данных или отдельным файлом).

5. Количество внешних интерфейсных файлов. Подсчитываются все логические файлы из других приложений, на которые ссылается данное приложение.

Достоинства функционально-ориентированных метрик:

1. Не зависят от языка программирования.

2. Легко вычисляются на любой стадии проекта.

Недостаток функционально-ориентированных метрик: результаты основаны на субъективных данных, используются не прямые, а косвенные измерения.