

Оглавление

1. Особенности CASE-технологии. История развития CASE-средств.	3
2. Системная модель CASE-средств.	5
3. Критерии развития CASE-средств.	6
4. Понятие проекта. Масштаб проекта. Общие принципы управления проектом	9
5. Три составляющие программного проекта: система обозначений, процесс и инструмент. Их роль и значение для проекта	11
6. Типы и особенности современных программных проектов.	12
7. Задачи и категории современных методологий создания программных проектов	12
8. Взаимосвязь между методологией, размером задачи и командой разработчиков	13
9. Целесообразность использования различных методологий для различных типов программных проектов	14
10. Унифицированный процесс разработки программных средств	15
11. Основные и дополнительные элементы объектно-ориентированного подхода.	16
12. История появления, особенности и назначение унифицированного языка моделирования UML	17
13. Требования к программному обеспечению. Первичные и детальные требования. Функциональные и нефункциональные требования	18
14. Назначение, особенности и построение диаграммы Use Case.	20
15. Назначение, особенности и построение диаграммы Deployment.	21
16. Назначение, особенности и построение диаграммы Statechart	22
17. Назначение, особенности и построение диаграммы Activity	24
18. Назначение, особенности и построение диаграммы Sequence	26
19. Назначение, особенности и построение диаграммы Collaboration	28
20. Назначение, особенности и построение диаграммы Component	29
21. Назначение, особенности и построение диаграммы Class, виды и особенности связей между классами на диаграммах.	30
22. Понятие шаблонов проектирования и их классификация. Шаблоны в нотации языка UML.	33
23. Шаблон “Фасад” и его обозначение в нотации языка UML	34
24. Шаблон “Наблюдатель” и его обозначение в нотации языка UML	35
25. Декомпозиция программной системы на модули. Принцип модульности. Оценка сложности программной системы через принцип модульности. Затраты на работу с модулями	36
26. Определение модуля. Связность и сцепление модулей. Типы связности и сцепления модулей	37
27. Создание модели предметной области программной системы с помощью диаграммы классов.	38

28. Создание модели анализа с помощью диаграммы классов. Различные стереотипы для классов и их назначение	39
29. Декомпозиция системы. Правила выделения подсистем	40
30. Особенности создания шаблона приложения в среде Rational Rose с использованием библиотеки MFC. Структура и классы приложения.	41
31. Функциональные возможности Rational Rose: модуль Component Assignment Tool, компонент Model Assistant, обновление кода по модели и модели по коду.	42
32. Особенности генерации исходного кода в среде Rational XDE. Способы синхронизации модели.	42
33. Назначение, возможности, особенности использования модуля Web Modeler в Rational Rose.	43
34. Возможности и особенности построения Web-модели в среде Rational XDE	43
35. Понятие проектных рисков. Действия по управлению рисками	44
36. Статический и динамический аспекты Rational Unified Process (RUP).	45
37. Принципы и стадии разработки ПС в технологии Rational Unified Process	46
38. Содержание и результаты первой и второй стадий в технологии Rational Unified Process	47
39. Содержание и результаты третьей и четвертой стадий в технологии Rational Unified Process	48
40. Этапы создания программных средств в технологии Oracle.	49
41. Процессы создания программных средств в технологии Oracle.	50
42. Сравнительный анализ технологий создания ПС Rational Unified Process, Oracle, нахуй этот ваш Borland	51
43. Понятия CASE-средство, CASE-система, CASE-технология, CASE-индустрия и различия между ними	52
То, что она спрашивает на экзамене	53

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

Особенности CASE-технологии

CASE-средства – программные средства, обеспечивающие поддержку многочисленных технологий проектирования информационных систем, охватывая всевозможные средства автоматизации и весь ЖЦ ПО.

CASE-система – набор CASE-средств, имеющих определенное функциональное предназначение и выполненных в рамках единого программного продукта.

CASE-технология обычно определяется как методология проектирования информационных систем плюс инструментальные средства, позволяющие наглядно моделировать предметную область, анализировать ее модель на всех этапах разработки и сопровождения информационно системы и разрабатывать приложения для пользователей (когда теорию объединяют с практикой).

CASE-индустрия объединяет сотни фирм и компаний различной ориентации. Практически все серьезные зарубежные программные проекты осуществляются с использованием CASE-средств, а общее число распространяемых пакетов >500 наименований.

CASE-средства используются для анализа и формирования требований, проектирования баз данных и приложений, генерации кода, тестирования, обеспечения качества. Особенности курса – ОО подход, автоматизация.

Особенности CASE-технологии: комплексный подход к разработке программных средств опирается на CASE- средства и в целом на CASE-технологию (в стремлении к достижению комплексного подхода).

Главная цель CASE-подхода – разделить и максимально автоматизировать любые этапы разработки программных средств.

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 на их основе. **Средства** – это инструментарий для поддержки методов на основе принятой нотации.

Основные преимущества CASE-подхода

- **улучшение качества ПО** за счет автоматического контроля всех выполняемых действий
- возможность **быстрого создания прототипа** будущей информационной системы, позволяет на ранних стадиях разработки увидеть результат
- **ускорение и усовершенствование** процессов проектирования и программирования
- **освобождение разработчиков** от выполнения рутинных операций
- возможность **повторного использования** ранее созданных компонентов

История развития CASE-средств

Основной причиной по которой возникла необходимость в появлении систем автоматизации создания информационных систем явился дисбаланс между

производительностями труда в сфере производства и в сфере инженерного труда (обработки информации) в пользу производства.

Сначала пытались решить проблему за счет людей, перебросом из производства в инженерный труд, что привело к падению общих темпов роста производительности труда и экономики.

Ситуация осложнялась следующими факторами:

- количество различных классов технических систем удваивалась в среднем каждые 10 лет
- сложность изделий повышалась
- объем научно-технической информации удваивался каждые 5 лет
- срок создания изделий уменьшался
- нельзя решить за счет количества людей (?)

Появились прародители систем автоматизации

- отличались от современных тем, что охватывали один или несколько этапов ЖЦ, те этапы, которые были лучше формализованы.

Современные CASE-средства делят на категории

- тяжелые
- средние
- легкие

Факторы определения классификации CASE-средств

- средства, вложенные в систему
- ресурсы, на которые опирается CASE-система (менее актуально) (система автоматизированного проектирования должна взаимодействовать только с активными ресурсами.):
 - активные (их составляет информация, доступная для автоматизированного хранения, поиска и обеспечивающая обработку данных)
 - пассивные
- охватываемые этапы ЖЦ (нынче - все) (?)
- хороший интерфейс, т.е. массовое восприятие или отторжение (?)

2. Системная модель CASE-средств.

Любая техническая система, включая и CASE-средства может быть представлена 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 (Ind) Каждая коммерческая система должна иметь зарегистрированный товарный знак, который в совокупности с обозначением паспортных данных (версии и модификации системы) представляет собой обозначение системы. Наименование включает в себя ее функциональное описание.

P - цель системы обычно достигается за счет ее технических функций, которые характеризуют способность системы 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 следующих действий:

- автоматизация оформления документов
- автоматизация поддержки и принятия проектных решений
- автоматизация программирования
- параллельное проектирование

Atr - используются в классификации CASE-средств по следующим признакам:

- прикладная область объектов проектирования
- сложность проектируемых объектов
- уровень автоматизации
- комплексность автоматизации
- возможность работы в сетевом режиме и в Internet.

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

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

3. Критерии развития CASE-средств

Критерии развития САПР

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 данной системы**.

Наборы критериев многих систем совпадают. В него входят:

- функциональные
 - скорость обработки информации (натуральный критерий производительности, информационный критерий производительности)
 - интенсивность обработки информации (информационный критерий эффективности)
 - степень автоматизации труда (критерий автоматизации)
 - непрерывность процесса проектирования (критерий непрерывности проектирования)
- технологические
 - трудоемкость
 - технологические возможности (критерий технологических возможностей)
- экономические
 - годовой экономический эффект от использования CASE-средства (экономический критерий)
- эргономические
 - КПД человека в системе (критерий эргономичности)

Функциональный критерий рассматривается как интегральный показатель, зависящий от ряда частных функциональных критериев, перечисленных выше.

Скорость обработки информации характеризуется двумя величинами: $K_{НП}$ – **натуральным критерием производительности CASE-средства** и $K_{ИП}$ – **информационным критерием производительности CASE-средства**.

$K_{НП} = \frac{(\sum_{i=1}^N K_{C_i} * N_{A4_i} / T_{ч_i})}{N}$, где K_{C_i} – коэффициент сложности работы i-ого проекта; N_{A4_i} – количество листов формата А4 i-ого проекта; $T_{ч_i}$ – время в часах, затраченное на автоматический выпуск указанного количества листов i-ого проекта; N – количество проектов. $K_{НП} > 0$, измеряется количеством листов, выпускаемых в часах, и возрастает с развитием CASE-средства. Если объем выполненной работы измеряется количеством информации в модели создаваемого объекта или процесса, то $K_{НП}$ можно заменить $K_{ИП}$

$K_{ИП} = \frac{(\sum_{i=1}^N K_{C_i} * l_i / T_{ч_i})}{N}$, где l_i – количество информации в принятых единицах измерения в i-ой модели. Независимо от критерия, которым измеряется, прежде всего производительность CASE-средства зависит от объема знаний и данных, заложенных в систему.

С **интенсивностью обработки информации** в CASE-средства связан **информационный критерий эффективности** $K_{ИЭ}$. Он представляет собой усредненное отношение объема выходной информации к суммарному объему входной и выходной информации.

$K_{из}$ принимает значения в диапазоне от 0 до 1 и является безразмерной величиной. Система считается тем совершеннее, чем меньше данных в нее нужно вводить. Предельным минимумом для этого является техническое задание.

Критерий, характеризующий степень автоматизации CASE-средства $K_A = \sum_{i=1}^{N_A} K_{T_i}$ – это **критерий автоматизации**, где N_A – количество автоматизированных операций, а K_{T_i} – удельная трудоемкость i -ой проектной операции. Сумма удельных трудоемкостей всех операций равна 1, т.е. $\sum_{i=1}^N K_{T_i} = 1$, где N – количество всех проектных операций. Следовательно, K_A принимает значения в диапазоне от 0 до 1.

Критерий непрерывности процесса проектирования $K_H = \frac{\sum_{i=1}^N \frac{T_i}{T_{цикл i}}}{N}$, где T_i – чистая трудоемкость i -ого проекта в человеко-часах, а $T_{цикл i}$ – длительность календарного цикла i -ого проекта. $T_{цикл i}$ становится неоправданно большим из-за перехода с одной операции на другую, нехватки информации и т.д. С развитием CASE-средств стремятся к увеличению данного критерия.

Критерий технологических возможностей отражает простоту разработки CASE-средства и подготовки ее к эксплуатации. Он является безразмерной величиной и определяется по формуле:

$K_{ТВ} = \frac{K_C A_C + K_Y A_Y + K_A A_A + K_H A_H}{A_C + A_Y + A_A + A_H}$, где A_C – количество стандартных программных модулей (ПМ), A_Y – количество унифицированных ПМ, A_A – количество адаптированных ПМ, A_H – количество нестандартных ПМ, требующих разработки. K – весовые коэффициенты, причем $K_C = 1$ и $K_C > K_Y > K_A > K_H$, $K_Y = 0.8$, $K_A = 0.5$, $K_H = 0.05$. Критерий технологических возможностей принимает значения в диапазоне от 0 до 1, и чем он больше, тем совершеннее CASE-средство.

В информатике принято различать старую и новую технологии. Для первой из них характерно доминирующее влияние нестандартных ПМ, когда $A_H \gg A_C, A_Y, A_A$, что не позволяет добиться увеличения $K_{ТВ}$. Новая технология характеризуется тем, что $A_H \rightarrow 0$. Доминирующее влияние при создании CASE оказывают унифицированные и стандартные ПМ, а при подготовке к эксплуатации – адаптированные.

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

$K_э = D_C + Э_K - (D_K + K_E) * E_H$, где

D_C – общее изменение себестоимости проектирования в расчетном году, связанное с производительностью CASE-средства;

$Э_K$ – годовая экономия от повышения качества проектных решений, в основе которых лежит новая информационная база;

D_K – дополнительные капиталовложения (в вычислительную технику), связанные с созданием и внедрением CASE-средства;

K_E – предпроизводственные затраты на CASE-средство, связанные с трудоемкостью ее разработки;

E_H – нормативный коэффициент.

Рассматриваемый критерий может принимать как положительные, так и отрицательные значения. Однако он имеет тенденцию к возрастанию.

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

При создании ПО приходится решать ряд задач, объединенных под названием - проект.

Проект - уникальный комплекс взаимосвязанных мероприятий, направленных на достижение конкретных целей при определенных требованиях к срокам, бюджету и характеристикам ожидаемых результатов. В этом определении следует обратить внимание на следующее:

- каждый проект характеризуется конкретной целью, ради которой он затевается
- в каждом проекте должна быть уникальность
- любой проект ограничен по времени жизни
- каждый проект характеризуется конкретными ресурсами, выделенными на его выполнение

С каждым проектом связано понятие **масштаб** - совокупность целей проекта и планируемых для ее достижений затрат времени и средств.

Управление проектом - это процесс планирования организации и контроля состояния задач и ресурсов проекта, направленный на своевременное достижение целей проекта. В ходе управления любым проектом должно быть обеспечено решение следующих задач:

- соблюдение директивных сроков завершения проекта
- рациональное распределение всех ресурсов и исполнителей между задачами проекта, учитывая время
- при необходимости в ходе ведения проекта вполне допустима коррекция исходного плана в соответствии с реальностью

В ведении проекта выделяют 3 основных этапа:

- формирование плана
- контроль реализации плана и управление проектом
- завершение проекта

Автоматизация проектирования должна помогать справляться со всеми сложностями на любых этапах разработки ПО, поэтому независимо от прикладной области задачи и принятого уровня абстракции необходимо, чтобы используемое CASE-средство поддерживало выполнение всех основных этапов ЖЦ ПС. Программные средства являются исполнительными элементами многих компьютерных систем различного назначения, например, CASE, гибких проектируемых систем, автоматизированных систем управления, компьютерных игр и др.

В связи с этим ПС становятся продуктом научно-технического назначения, создаются в строгом соответствии с действующими стандартами и утвержденной технологией, сопровождаются научно-технической документацией и обеспечиваются гарантиями поставщика. Это объясняется тем, что технические возможности, адаптируемость и эффективность КС определяются качеством используемого ПО, а без современной индустриальной технологии его создания требуемое качество недостижимо.

В области программирования так же, как и в промышленном производстве, значительный эффект может дать многократное использование хорошо отработанных компонентов в качестве комплектующих изделий. Такие компоненты выполняют

типовые функции или функции, характерные для определенных предметных областей применения компьютерных систем, и называются программными модулями.

Для обеспечения повторного использования ПС необходима стандартизация их создания на всех этапах ЖЦ. Это позволяет значительно сократить дублирующие разработки, внедрить сборочное программирование и ввести на предприятиях накопление высококачественных программных продуктов для их многократного использования в качестве типовых комплектующих изделий.

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

Целесообразно использовать CASE-технологии при создании ПС только в тех случаях, когда прикладные программные продукты будут многократно использоваться, состоять из хорошо отработанных элементов (программных модулей), являться составными частями программных комплексов/систем разного назначения. Для достижения этого используют CASE-технологии, в основе которых ЖЦ ПС. Комплекс мероприятий по созданию ПС принято называть **проект**. Успех любого программного проекта зависит от трех составляющих:

- **Система обозначений** (нотация, язык). Нужна в любой модели для ведения любого программного проекта. Это связующее звено между всеми составляющими процесса разработки. В качестве нотации выберем UML (Unified Modeling Language) – унифицированный язык моделирования. С его помощью можно описывать модели на любом этапе разработки ПС: от анализа требований до проектирования и реализации
- **Процесс**. В качестве процесса, подходящего для современных IT-проектов, выберем унифицированный процесс (RUP). Его методология основана на UML и поддерживается современными инструментами, позволяет команде разработки преобразовывать требования заказчика в работоспособный продукт
- **Инструмент**. В качестве инструмента выберем объектно-ориентированное CASE-средство (ПС, обеспечивающие поддержку многочисленных технологий проектирования информационных систем, охватывая всевозможные средства автоматизации и весь ЖЦ ПО), использующее нотацию UML

6. Типы и особенности современных программных проектов.

Основные типы проектов:

1. **Проект для постоянного заказчика** - когда команда разработчиков в течение длительного времени работает с одним заказчиком (*легкая или средняя методологии*)
2. **Продукт под заказ** - ситуация, когда команда разработчиков находит разового заказчика, с которым заключается договор (*тяжелая или средняя*)
3. **Тиражируемый продукт** - ситуация, когда команда разработчиков вообще не имеет конкретных заказчиков, либо заказчиков много, и можно тиражировать свой продукт (*легкая или средняя*)
4. **Аутсорсинг** - фирма часть своих заказов передает посредникам (небольшим фирмам) (*тяжелая или средняя*)

7. Задачи и категории современных методологий создания программных проектов

Методология — набор методов, практик, метрик и правил, используемых в процессе производства ПО.

Главные задачи современной методологии и основанного на ней процесса следующие:

- облегчить процедуру введения новых людей в курс процесса производства;
- обеспечить взаимозаменяемость людей;
- распределить ответственности;
- продемонстрировать видимый прозрачный процесс;
- создать учебную базу для сотрудников

Тяжелые методологии

- появились раньше других и связаны с моделью качества ПО
- охватывают все этапы компании, производящей ПО
- большие команды разработчиков
- достаточно сложные задачи
- длительный период разработки
- этапы ЖЦ не пропускаются, все проходятся, сопровождаются документированием и тестированием
- ISO9001

Легкая методология

- небольшие команды разработчиков
- небольшие проекты
- короткий период разработки
- максимальная скорость разработки без потери качества
- SCRUM

Средняя методология

- универсальные процессы разработки
- Rational Unified Process - унифицированный процесс разработки
- **масштабируемость**
- наиболее предпочтительная

8. Взаимосвязь между методологией, размером задачи и командой разработчиков

Рассмотрим 3 категории методологии создания программных средств.

Пред вопрос?

Тяжелые методологии

- появились раньше других и связаны с моделью качества ПО
- охватывают все этапы компании, производящей ПО
- *большие команды разработчиков*
- достаточно сложные задачи
- длительный период разработки
- этапы ЖЦ не пропускаются, все проходятся, сопровождаются документированием и тестированием
- ISO9001

Легкая методология

- *небольшими команды разработчиков*
- небольшие проекты
- не длительный период разработки
- максимальная скорость разработки без потери качества
- SCRUM

Средняя методология

- *универсальные процессы разработки*
- Rational Unified Process - унифицированный процесс разработки
- масштабируемость
- наиболее предпочтительная

9. Целесообразность использования различных методологий для различных типов программных проектов

==Вопрос 6

Основные типы проектов:

1. проект для постоянного заказчика - когда команда разработчиков в течение длительного времени работает с одним заказчиком (*легкая или средняя методологии*)

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

2. продукт под заказ - ситуация, когда команда разработчиков находит разового заказчика, с которым заключается договор (*тяжелая или средняя*)

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

3. тиражируемый продукт - ситуация, когда команда разработчиков вообще не имеет конкретных заказчиков, либо заказчиков много, и можно тиражировать свой продукт (*легкая или средняя*)

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

4. аутсорсинг - фирма часть своих заказов передает посредникам (небольшим фирмам) (*тяжелая или средняя*)

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

10. Унифицированный процесс разработки программных средств

Унифицированный процесс можно рассматривать как основу средней методологии, которую мы признали наиболее гибкой. Часто **унифицированный процесс** рассматривают как специализированный набор методов и средств, подходящий для широкого круга задач программных систем.

Унифицированный процесс имеет 3 основные характеристики:

1. итеративный и инкрементный подход к созданию ПО
2. управление вариантами использования
3. процесс архитектурно-ориентированный

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 итеративным и инкрементным.

Вариант использования – это часть функциональности системы, необходимая для получения пользователем значимого для него, осязаемого и измеримого, результата. Варианты использования обеспечивают функциональные требования. Все варианты использования в совокупности составляют **модель вариантов использования**, которая описывает полную функциональность системы.

Варианты использования в Unified Process – это не только средство описания требований к системе. Они направляют далее весь процесс ее разработки. Основываясь на модели вариантов использования, разработчики создают все последующие модели.

Поскольку варианты использования управляют процессом, они не выделяются изолированно, а разрабатываются совместно с созданием архитектуры системы. Следовательно, варианты использования управляют архитектурой системы, которая, в свою очередь, оказывает влияние на их выбор. И архитектура системы, и варианты использования развиваются по мере хода жизненного цикла.

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

1. Архитектура (1 ветвь)
 - a. Составление требований -> спецификация требований (документ)
 - i. Первичные (заказчика)
 - ii. Детальные (разработчика)
 - b. Архитектурное проектирование (результат: архитектура программы и данных)
 - c. Детальное проектирование
2. Разработка интерфейса (взаимодействие человека с системой) (2 отдельная ветвь)

Результат: структуры данных и нотации (?).

11. Основные и дополнительные элементы объектно-ориентированного подхода.

Главное отличие объектного подхода от структурного заключается в объектной декомпозиции системы

Концептуальной основой объектного подхода является объектная модель. Рассмотрим ее основные принципы:

1. **абстрагирование** - выделение существенных характеристик некоторого объекта, которые отличают его от всех других видов объектов и тем самым позволяют определить его границы для дальнейшего анализа/проектирования/всего остального. Выбор правильного набора абстракций - это главная задача ОО подхода.
2. **инкапсуляция** - процесс отделения друг от друга элементов объекта, определяющих его устройство и поведение. Инкапсуляция служит для того, чтобы разделить интерфейс и внутреннюю реализацию объекта.
3. **модульность** - свойство системы, связанное с ее декомпозицией на ряд внутренне сильно связанных и слабо связанных между собой модулей. Инкапсуляция и модульность создают барьеры между абстракциями.
4. **иерархичность** - ранжированная и упорядоченная система абстракций. Иерархия по номенклатуре - это структура классов. Иерархия по составу - это структура объектов.
5. **типизация** - ограничение, накладываемое на класс объектов и препятствующее взаимозаменяемости различных классов.
6. **параллелизм** - свойство объектов находиться в активном или пассивном состоянии и различать эти состояние между собой.
7. **устойчивость** - свойство объекта существовать во времени и пространстве вне зависимости от процесса, породившего данный объект.
8. **полиморфизм** - способность класса принадлежать более чем одному типу.
9. **наследование** - построение новых классов на основе существующих с возможностью добавления или переопределения данных и методов.

Благодаря применению абстрагирования, модульности и полиморфизма на всех стадиях разработки программной системы, существует согласованность между моделями всех этапов ЖЦ, когда модели ранних стадий могут быть сравнимы с моделями реализации и наоборот.

Шаблоны ОО анализа и проектирования

Попытки выявить схожие схемы или структуры в рамках ООП и анализа привели к появлению понятия "шаблон/паттерн". Это понятие сегодня широко применяется в современных case-средствах.

Различаются степенью детализации и уровнем абстракции.

- **архитектурные паттерны** - множество предварительно определенных подсистем со спецификацией их ответственности, правил и базовых принципов установления отношений между ними GRASP (General Responsibility Assignment Software Pattern)
- **паттерны проектирования** - специальные схемы для уточнения структуры подсистем или компонентов программной системы и отношений между ними GoF (Gang of Four)

- **паттерны анализа** - специальные схемы для представления общей организации процесса моделирования ARIS (Architecture of Integrated Information Systems)
- **паттерны тестирования** - специальные схемы для представления общей организации процесса тестирования программных системы (IBM Test Studio)
- **паттерны реализации** - совокупность компонентов и других элементов реализации, используемых в структуре модели при написании программного кода

Паттерны проектирования представляются в наглядной форме с использованием обозначений языка UML. При изображении шаблона используются обозначения параметризованной кооперации языка UML.

На современном этапе широко используются на всех этапах ЖЦ шаблоны (паттерны), которые позволяют выбрать для создания программной системы наиболее удачные решения.

Рассмотрим несколько примеров архитектур, построенных с применением архитектурных шаблонов:

MVC (Контроллер - представление - модель)

Паттерн хранилища данных: к хранилищу данных подключаются утилиты для эффективной обработки больших объемов данных

Клиент-серверная архитектура сетевой библиотеки

Трехъярусная архитектура

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

UML — нотация, которая используется методом для описания проектов. Нотация (синтаксис языка моделирования) представляет собой совокупность графических примитивов, которые используются в моделях. **Процесс** в данном случае - это описание шагов, которые необходимо выполнить при разработке проекта. Авторы UML: Бутч, Рамбо, Якобсон. UML не привязан к какой-либо конкретной методологии или ЖЦ и может использоваться во всех существующих методологиях (больше используется в ОО подходе). Основная идея UML - это возможность моделировать ПО (ПС) как наборы взаимодействующих объектов. В UML модели есть 2 аспекта:

- статическая структура: описывает, какие типы объектов важны для моделирования системы и как они взаимодействуют между собой
- динамическое поведение: описывает ЖЦ объектов и их взаимодействие между собой для обеспечения нужной функциональности системы

Основные цели создания UML:

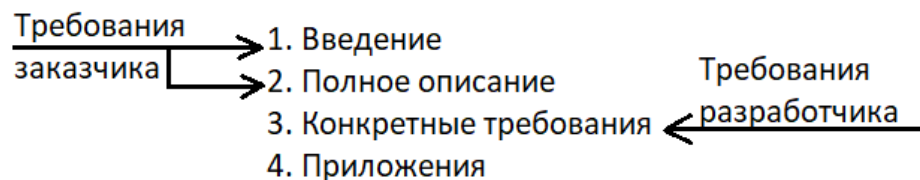
1. предоставить пользователям готовый к применению выразительный язык визуального моделирования, позволяющий разрабатывать модели и обмениваться ими
2. предусмотреть механизмы для расширения базовых концепций языка
3. создать язык, независимый от конкретных языков программирования и процессов разработки
4. стимулирование роста рынка ОО инструментальных средств
5. интегрировать лучший практический опыт

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

Придерживаясь ЖЦ создания ПС в качестве первого этапа в создании программного средства является разработка требований. Требования содержат в себе функциональные возможности, ограничения и другие специфические особенности разработки, которые нужно учесть. На первом этапе два вида требований:

- первичный - выдвигает заказчик
 - документирует все желания и требования заказчика и составляются на языке, понятном заказчику.
 - требования заказчика помещаются в документ под названием **системная спецификация**
- детальный - выдвигает разработчик
 - документируют требования в специальной структурированной форме
 - детализированы по отношению к первичным требованиям
 - оформляются в виде **спецификации анализа**

Результат работы заказчика и разработчика организуется в виде **спецификации требования**:



Кроме первичных и детальных требований, требования делятся на:

- функциональные требования
 - описывают поведение системы и функции, которые она должна выполнять
 - исходят из всестороннего анализа предметной области
- нефункциональные требования
 - относятся к характеристикам системы и ее внешнего окружения (вместе: как внешняя среда будет влиять на функциональность системы)
 - могут перечисляться ограничения и условия разработки

Группы нефункциональных требований:

1. **Требования к программной системе.** Описывают свойства и характеристики системы. Сюда относятся требования к скорости работы, производительности, емкости необходимой памяти, надёжности, переносимости системы на разные компьютерные платформы и удобство эксплуатации
2. **Организационные требования.** Отражают вопросы работы и организации взаимодействия заказчика и разработчика (стандарты разработки, набор документации, сроки разработки, требования к методам разработки)
3. **Внешние требования.** Учитывают факторы внешней среды. Определяют требования по взаимодействию системы с внешним окружением, юридические обстоятельства

Алгоритм формирования требований:

1. Определение представителей заказчика
2. Проведение опроса представителей заказчика (определяется порядок и длительность опроса)
3. Документирование результатов опроса
4. Проверка требований

Анализ требований

1. Организация первичных требований
 - a. По режиму
 - b. По категориям пользователей
 - c. По объектам
 - d. По свойствам
2. Преобразование первичных требований в детальные
(одно требование заказчика -> одно или несколько детальных требований)
Рекомендации по работе с детальным требованием
 1. Обеспечить прослеживаемость требования
 2. Обеспечить тестируемость требования
 3. Анализ однозначности толкования
 4. Назначение приоритета требования
 5. Проверка полноты требования
 6. Проверка согласованности требования с другими требованиями
 7. Требование заносится в спецификацию анализа
3. Аттестация детальных требований (должна подтвердить, что требования действительно определяют ту систему, которая нужна заказчику)
 1. Проверка правильности
 2. Проверка на непротиворечивость
 3. Проверка на полноту
 4. Проверка на выполнимость

14. Назначение, особенности и построение диаграммы Use Case

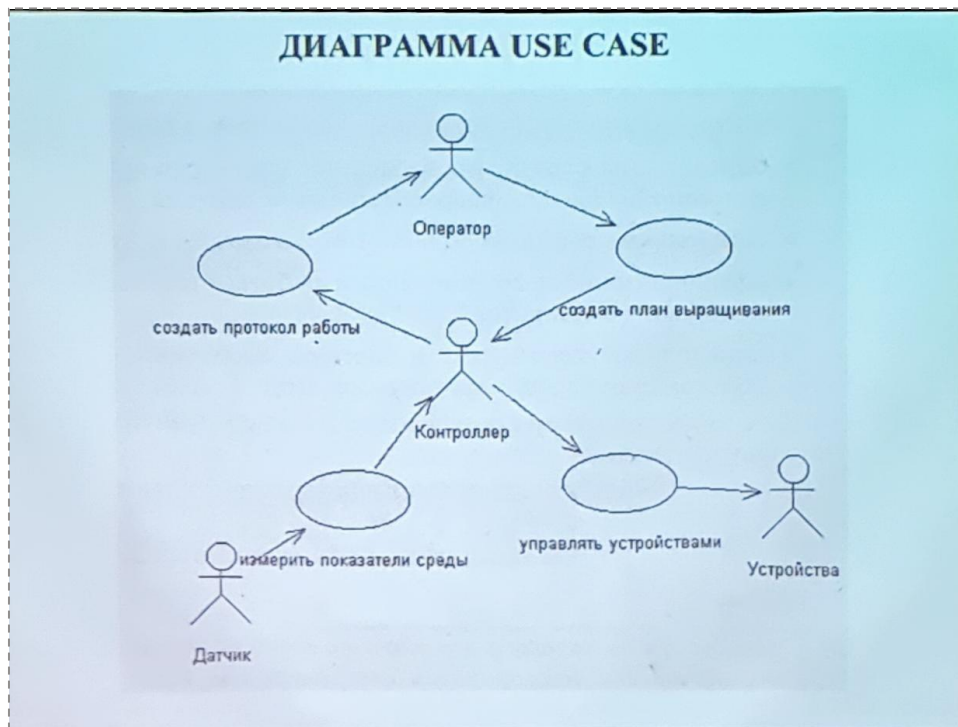


Диаграмма Use Case позволяет создать список операций, которые выполняет система. Часто Use Case называют диаграммой функций, так как на основе набора таких диаграмм создается список требований к системе и определяется множество выполняемых ею функций. Данный тип диаграмм используется при описании бизнес-процессов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

Согласно постановке задачи, построим диаграмму Use Case

Начинаем с оператора, который запускает процесс (создать план выращивания).

Все функции контроля, управления выполняет устройство - контроллер, который связан с протоколом работы, куда заносится работа.

От протокола работы переходим на оператора.

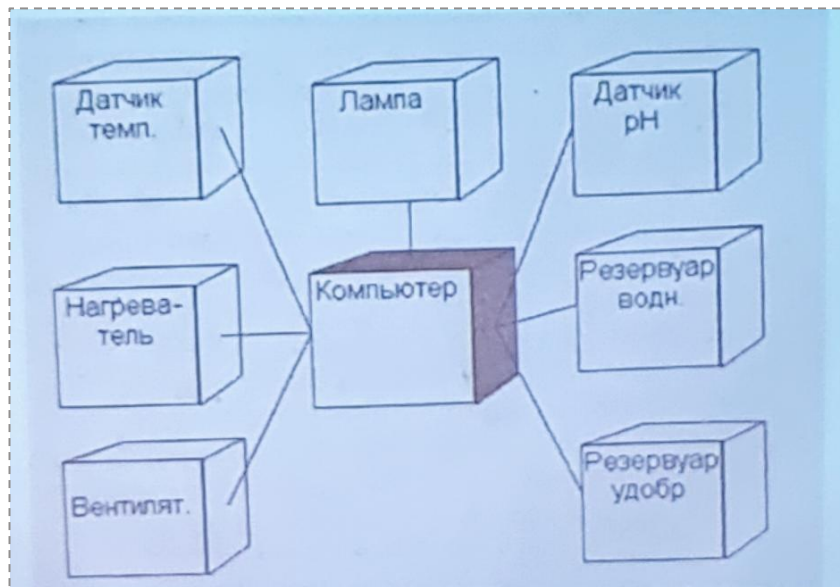
Для поддержки нормальных условий в системе контроллер связан с двумя типами устройств: с датчиками и исполнительное устройство.

Датчик измеряет показатели среды и сведения возвращает контроллеру.

Контроллер анализирует, если нужный момент - записывается в протокол работы, иначе - посылается сигнал исполнительным устройствам.

Принципы ООП на диаграмме: диаграмма UseCase освобождена от всех принципов ООП.

15. Назначение, особенности и построение диаграммы Deployment



Единственная диаграмма, моделирующая аппаратную часть системы.

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

После того, как построена диаграмма Use Case, известны все актеры в системе, имеет смысл построить диаграмму Deployment, предназначенную для анализа аппаратной части системы. На этой диаграмме используется 2 основных элемента: Processor и Device.

Процессор - любое устройство, выполняющее вычислительный процесс.

Девайс - любое другое устройство.

Связь - connection - умышленно без управления (отображает физ. связь)

Рассмотрим Deployment диаграмму для теплицы:

Только одно устройство процессорного типа - контроллер/компьютер и ряд датчиков и исполнительных устройств, которые потребуются в системе исходя из постановки задачи.

Построенная диаграмма показывает, что работа системы планируется на одном компьютере, соединенном с датчиками и исполнительными устройствами.

Датчиков в системе планируется 2: датчик температуры, датчик кислотности (pH).

Исполнительные устройства представлены: лампа, нагреватель, вентилятор, два резервуара: с водой и с удобрениями.

Сравнение Deployment и Component

- Физическая структура ПС – Component
- Физическая структура аппаратной части - Deployment

16. Назначение, особенности и построение диаграммы Statechart

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

Фактически процесс проектирования начинается с этих диаграмм. Они будут строиться на основе принципов ОО подхода. Все диаграммы до диаграммы Component будут строиться для объектов создаваемой системы или ПС, поэтому прежде чем переходить к работе с объектами необходимо добавить в модель классы, которым будут принадлежать все рассматриваемые объекты.

В основу диаграмм состояний положен принцип работы конечных автоматов, с деятельностью которых связывают начальное и конечное состояние, что мы видим в наборе инструментов диаграммы state chart.

Start state - начальное состояние, всегда одно

End state - конечное состояние, может быть несколько

Инструменты state позволяет отразить состояние объекта на диаграмме в течение периода жизни

Для связи состояний на диаграмме используется инструмент StateTransition, который имеет вид направленной стрелки. Есть стрелка self transition (переход в себя).

- EnvironmentalController - контроллер управления исполнительными устройствами
- TemperatureSensor - датчик температуры
- pHSensor - датчик кислотности
- Heater - нагреватель
- Cooler - вентилятор для снижения температуры
- Light - осветитель
- WaterTank - хранилище для воды
- NutrientTank - хранилище для удобрений

Первой рассмотрим построение диаграммы статических объектов (StateChart).

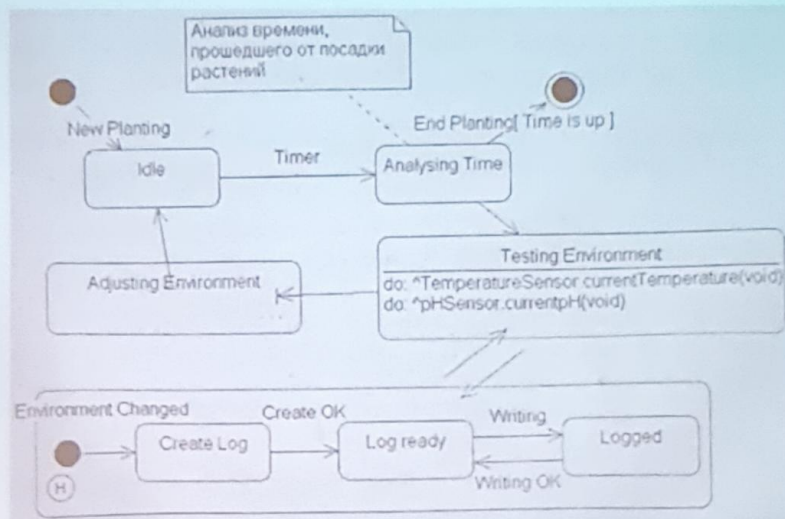
Диаграмма отражает статические состояние объекта контроллер.

Первым на диаграмме запускается состояние start state. После чего контроллер переходит в состояние ожидания. В определенный момент времени происходит анализ времени. Если время по плану выращивания вышло, то происходит переход на состояние end state, иначе - переходим в состояние тестирования окружающей среды (в этом состоянии включаются в работу датчики).

Когда время истечет, включаются исполнительные устройства и переходим в состояние ожидания.

Выбирая между State и Activity для менее активных объектов рекомендуется отдавать предпочтение диаграмме State Chart.

ДИАГРАММА STATECHART



17. Назначение, особенности и построение диаграммы Activity

Рассмотрим принцип построения динамических состояний (Activity) для объекта контроллер.

В основе ее построения лежит принцип работы конечных автоматов.

Все инструменты диаграммы Statechart доступны диаграмме Activity, кроме них добавляются еще инструменты

- activity - действие объекта или динамическое состояние (более закруглённые углы чем у State)
- swimlanes - плавательные дорожки (для структурирования области диаграмм)
- decision - решение (логика работы)
- synchronizations _ - синхронизация. | - синхронизация объектов глобально (более редкая)

Диаграмма activity для объекта контроллер может иметь такой вид.

Дополнительно к объекту контроллер были добавлены еще 2 объекта: таймер и план выращивания. Действия каждого из 3х объектов показаны в отдельных swimlane.

Диаграмма activity начинается с состояния start state. Запускается план выращивания и устанавливается таймер. Система переходит в состояние ожидания. Затем, в определенный момент времени контроллер запрашивает текущее время и план выращивания. Контроллер ждет ответа от каждого из объектов (показано в виде инструмента синхронизации). После анализируются ответы. Если время истекло - end state, иначе - переходим на activity тестирования состояния среды. После анализируем показания датчиков. Если изменения произошли - протоколируем. Переходим в состояние decision и включаем исполнительные устройства, если требуется. Идем в состояние ожидания.

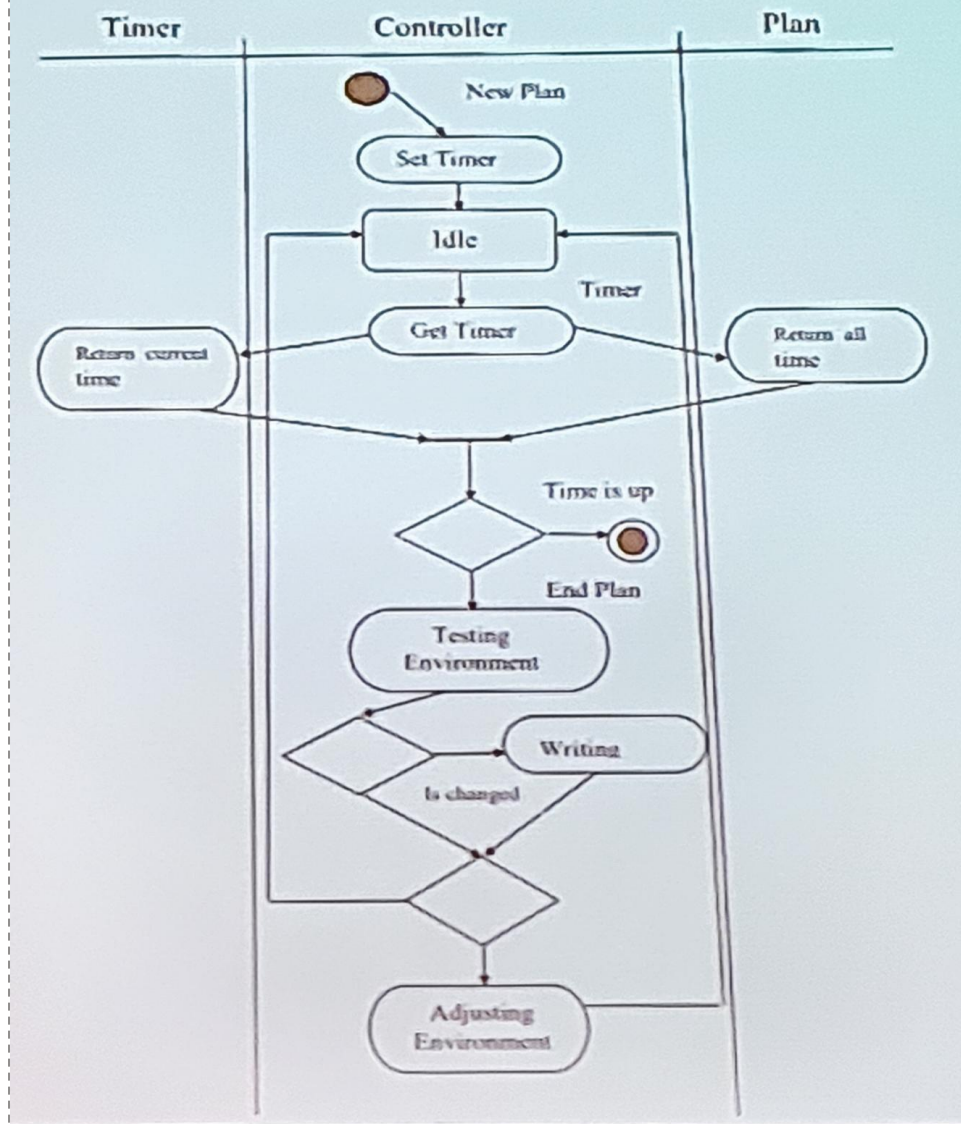
Вывод:

1. Поскольку контроллер - активный объект системы, его состояние лучше отображать с помощью Activity.
2. Обе диаграммы отображают одни и те же состояния.
3. Чтобы выбрать диаграмму для конкретного объекта рекомендуется для менее активных объектов отдавать предпочтение StateChart, для активных - Activity.

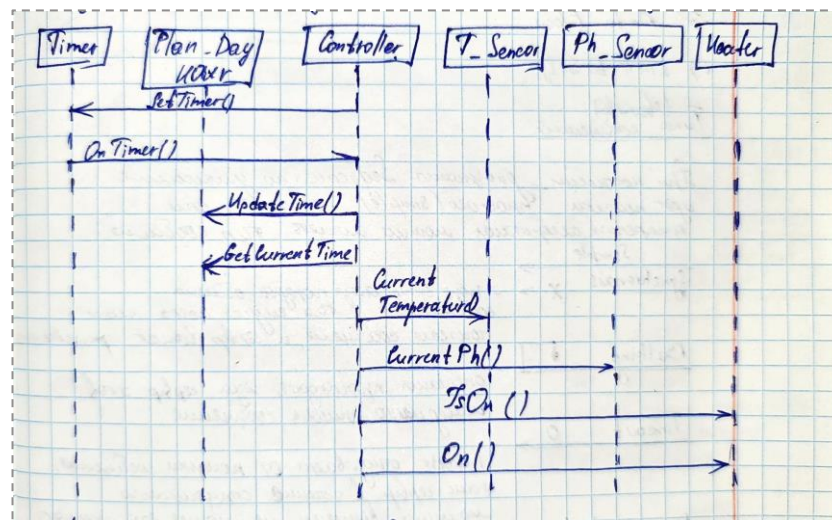
Выбирая между State и Activity для менее активных объектов рекомендуется отдавать предпочтение диаграмме State Chart.

Принципы ООП на диаграмме: параллелизм, абстрагирование.

ДИАГРАММА ACTIVITY



18. Назначение, особенности и построение диаграммы Sequence



То, что ниже - выполняется позже. Диаграмма показывает это время как относительную величину (никаких абсолютных значений!)

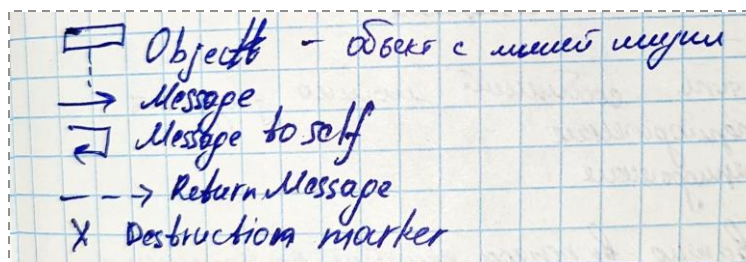
Вертикальные линии - "линии жизни".

Пара диаграмм отображает взаимодействие между объектами системы. Все объекты воспринимаются на объекты-клиенты и объекты-серверы. Обмен сообщениями происходит в определенном (заданном) порядке, в результате диаграмма позволяет получить взаимодействие во времени. Объекты-клиенты посылают сообщения,

серверы обрабатывают.

Обычно посылка сообщения связана с вызовом определенного метода.

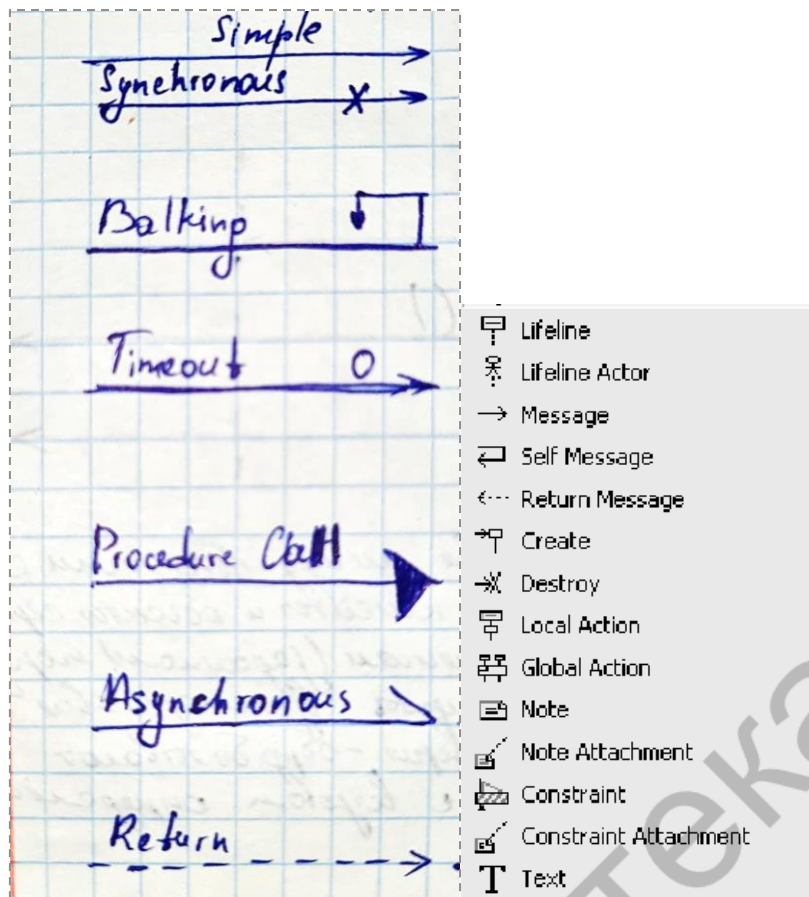
Типы и свойства сообщений
При построении диаграммы sequence по умолчанию идет посылки простого (simple)



сообщения, при конкретных алгоритмах можно менять тип сообщения

Инструменты Sequence:

- Объект
- Сообщение между объектами
- Сообщение самому себе
- Возврат сообщения
- Маркер уничтожения



При построении диаграммы **Sequence** по умолчанию между объектами прокладывается послылка простого сообщения, при необходимости можно поменять вид стрелки, и указать на отличный от первоначального алгоритма взаимодействия:

- **synchronous** — пара определяет порядок обмена сообщениями в том случае, когда клиент посылает сообщения, а сервер может принять его
- **balking** — операция происходит, когда сервер готов немедленно принять сообщения
- **timeout** — клиент отказывается от посылки сообщения, если сервер в течение определенного момента времени не может его принять
- **procedure call** — клиент вызывает процедуру сервера и полностью передает ему управление
- **asynchronous** — клиент отправляет сообщения и не ожидает ответа от сервера
- **return** — парное (ответное) сообщение
- **simple**

Кроме типа сообщения можно указать частоту отправки. Они могут быть периодическими или поступать нерегулярно (апериодически). Рекомендуется сначала строить диаграмму **Sequence**, а потом **Collaboration**, хотя можно превратить одну в другую. Просто тогда точно будет правильный порядок обмена сообщениями.

Сравнения Collaboration и Sequence: Collaboration может рассматриваться как состояние системы здесь и сейчас, в какой-то момент времени.

19. Назначение, особенности и построение диаграммы Collaboration

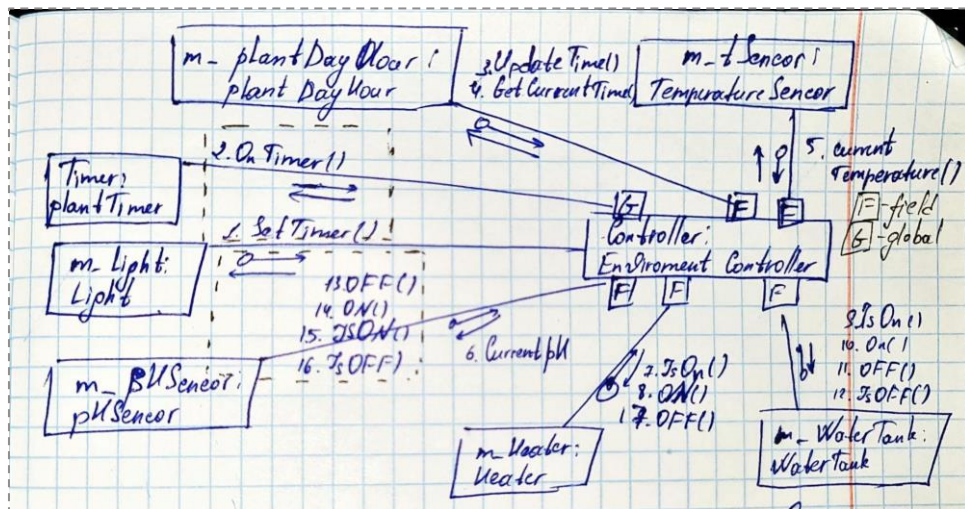


Диаграмма Collaboration - сиюминутный снимок диаграммы классов.

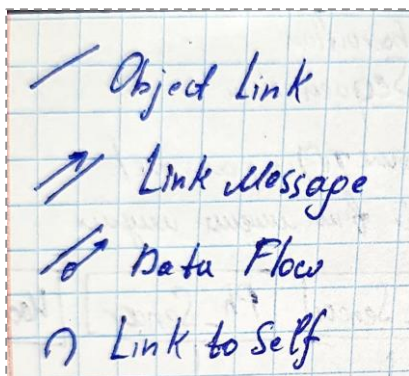
Можно Sequence -> Collaboration - рекомендуют.

Collaboration -> Sequence - не рекомендуется, поскольку временная составляющая может повредиться.

В отличие от Sequence Collaboration не содержит временную шкалу и линии жизни объектов. Collaboration показывает взаимодействие объектов в принципе, а также, кроме посылки сообщений, обмен данными.

Поскольку отсутствует линия жизни, диаграмма получается более компактной. Все сообщения или данные отображаются пакетом (списком).

В диаграмме Collaboration важно само взаимодействие и менее важно конкретное время. Если мы хотим посмотреть взаимодействие между всеми объектами, то это диаграмма Collaboration.

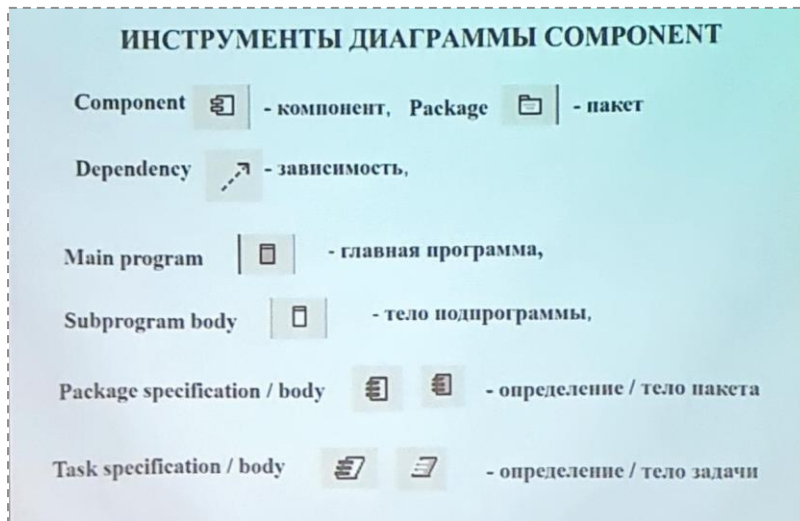


Инструменты Collaboration:

- Связь объекта
- Передача сообщения
- Передача данных
- Связь с самим собой

Принципы ООП на диаграмме: абстрагирование.

20. Назначение, особенности и построение диаграммы Component



Инструменты:

- Component - отражает какую-то часть программной системы
- Dependency - зависимость
- Main program - главная программа
- Subprogram body - подпрограмма
- Package specification/body - определение/тело пакета - эта пара элементов позволяет отразить на диаграмме определение и описание пакета
- Task specification/body - определение/тело задачи

Основной элемент Component - отражает часть создаваемой программной системы

Пары элементов позволяют отразить на диаграмме определение пакета и описание пакета.

Определение пакета можно рассматривать как заголовочный файл, а тело пакета как реализация.

Назначение диаграммы Component:

Представление структуры, разрабатываемой программной системы с точки зрения программных элементов (файлы, модули)

Обычно строится на завершающем этапе проектирования перед диаграммой классов.

Эта диаграмма позволяет создать физическое изображение текущей модели будущей системы. Эту диаграмму часто сравнивают с Deployment, которая отражает структуру аппаратной части.

Сравнение Deployment и Component

- Физическая структура ПС – Component
- Физическая структура аппаратной части - Deployment

На диаграмме компонент отражаются следующие **принципы ООП**: *модульность, инкапсуляция* (т.к. компоненты связаны через интерфейс).

Инструментов у этой диаграммы больше чем в любых других средах.

Строится на основе всех ранее построенных диаграмм.

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

Диаграмма Class является самой важной и многофункциональной в наборе диаграмм среды разработки. Главное назначение диаграммы Class - генерация по ней кода.

Однако, кроме этого, с помощью инструментов диаграммы можно строить и другие модели, по которым не рекомендуют генерировать код, но эти модели существенно помогают проектированию программной системы.

Для диаграммы Class предусмотрено 2 основных инструмента:

- **класс.** Class на диаграмме представляется прямоугольником (по умолчанию), разделенным на 3 части: имя класса, атрибуты, методы.
- **связь между классами.** Тип связи влияет на структуру сгенерированного кода

Чаще всего классы связаны на диаграмме.

Значок *Association* показывает ассоциации классов. Поскольку направление не указано, Rational XDE не может определить, какой класс первичен в установленной ассоциации, поэтому считает оба класса равноправными. Этот вид связи не используется для проектирования классов, по которым планируется получить исходный код, т.к. он никак на него не влияет. Значок *Directed Association* позволяет создать направленную ассоциацию между классами. Этот тип связи показывает, что объект одного класса включается в другой класс для получения доступа к его методам. Rational XDE определяет название переменной для объекта класса, и при генерации исходного кода эта переменная будет включена в него перед определением методов и атрибутов класса.

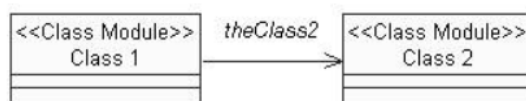


Рисунок 8.7 – Однонаправленная ассоциация

Значок *Aggregation association* позволяет отразить на диаграмме агрегацию элементов. Этот тип связей показывает, что один элемент входит в другой как часть. Агрегация используется при моделировании как дополнительное средство, показывающее, что элемент состоит из отдельных частей.

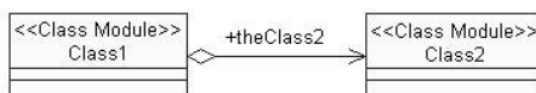


Рисунок 8.8 – Агрегирование класса

Значок *Dependency* показывает на диаграмме такое отношение зависимости, когда один класс использует объекты другого. Для классов C# этот тип связи не имеет широкого применения и на создаваемый код не влияет.



Рисунок 8.9 – Связь Dependency

Значок *Association Class* используется для отображения класса, ассоциированного с двумя другими. Фактически данный тип связи отражает то, что некоторый класс со своими атрибутами включается как элемент в два других, при этом никак не отражаясь на создаваемом исходном коде.

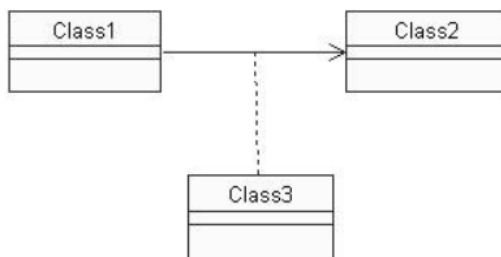


Рис. 8.10 – Связь Association class

Значок *Generalization* указывает, что один класс является родительским по отношению к связанному, при этом будет создан код наследования класса.



Рисунок 8.11 – Связь Generalization

Значок *Realization* отражает на диаграмме отношение между классом и интерфейсом, который этим классом реализуется.



Рисунок 8.12 – Связь Realization

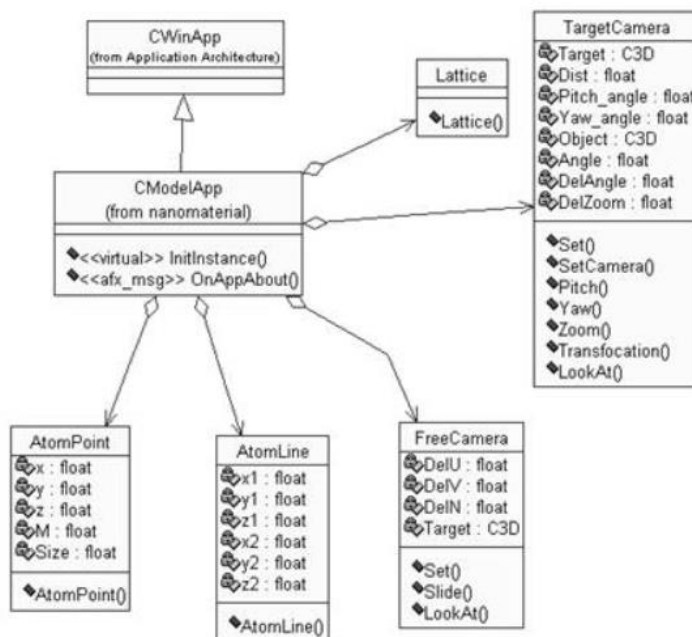


Рисунок 8.13 – Фрагмент диаграммы классов

Значок *Interface* позволяет создать элемент интерфейса. Он представляет собой абстрактный контейнер абстрактных операций, которые должны быть реализованы в классе.

Значок *Signal* позволяет создать элемент сигнала, который отражает асинхронное сообщение от одного класса к другому, т.е. без ожидания ответа.

Значок *Enumeration* создает элемент перечисление. Это тип данных, состоящий из набора констант базового типа. Перечисления используются для создания набора именованных страниц.

Классы не могут существовать обособленно. Основная их задача – взаимодействовать друг с другом при помощи обмена сообщениями. При построении модели классы связываются друг с другом связями различных видов. Согласно спецификации UML таких связей довольно много, однако в отличие от C++ в C# многие виды связей дают одинаковые результаты при генерации исходного кода.

Значок *Abstraction* определяет, что элементы модели представляют одно понятие, но на разных уровнях абстракции.

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

При реализации проектов по разработке информационных систем и моделировании бизнес процессов часто встречаются ситуации, когда решение задач в разных проектах имеют схожие структурные черты. Попытки выявить схожие схемы или структуры в рамках ООП и анализа привели к появлению понятия шаблон. *Если есть одна нотация UML и мы как диаграммы, так и шаблоны отображаем с помощью одних примитивов, не лучше ли выбрать что-то одно? Нет, диаграммы - проектирование, а когда мы на этапе перехода к программе, мы можем что-то уточнить с помощью шаблонов.* Паттерны различаются степенью детализации и уровнем абстракции:

- 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
- паттерны анализа - специальные схемы для представления общей организации процесса моделирования
- паттерны тестирования - специальных схемы для представления общей организации процесса тестирования программных системы
- паттерны реализации - совокупность компонентов и других элементов реализации, используемых в структуре модели при написании программного кода

Общее у диаграмм и шаблонов:

- UML
- На этапе проектирования
- Ни по тому, ни по другому не генерируют код (влияет на кодогенерацию только диаграмма классов)

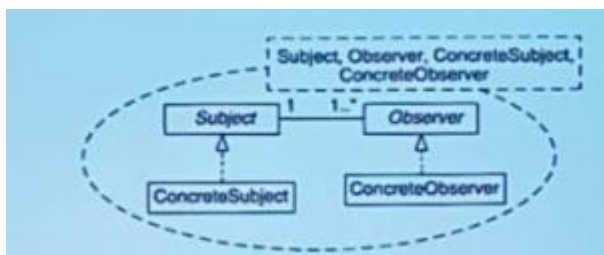
Различия:

- Диаграммы строят раньше, у них выше уровень абстракции

Паттерны проектирования

Паттерны проектирования представляются в наглядной форме с использованием обозначений языка UML

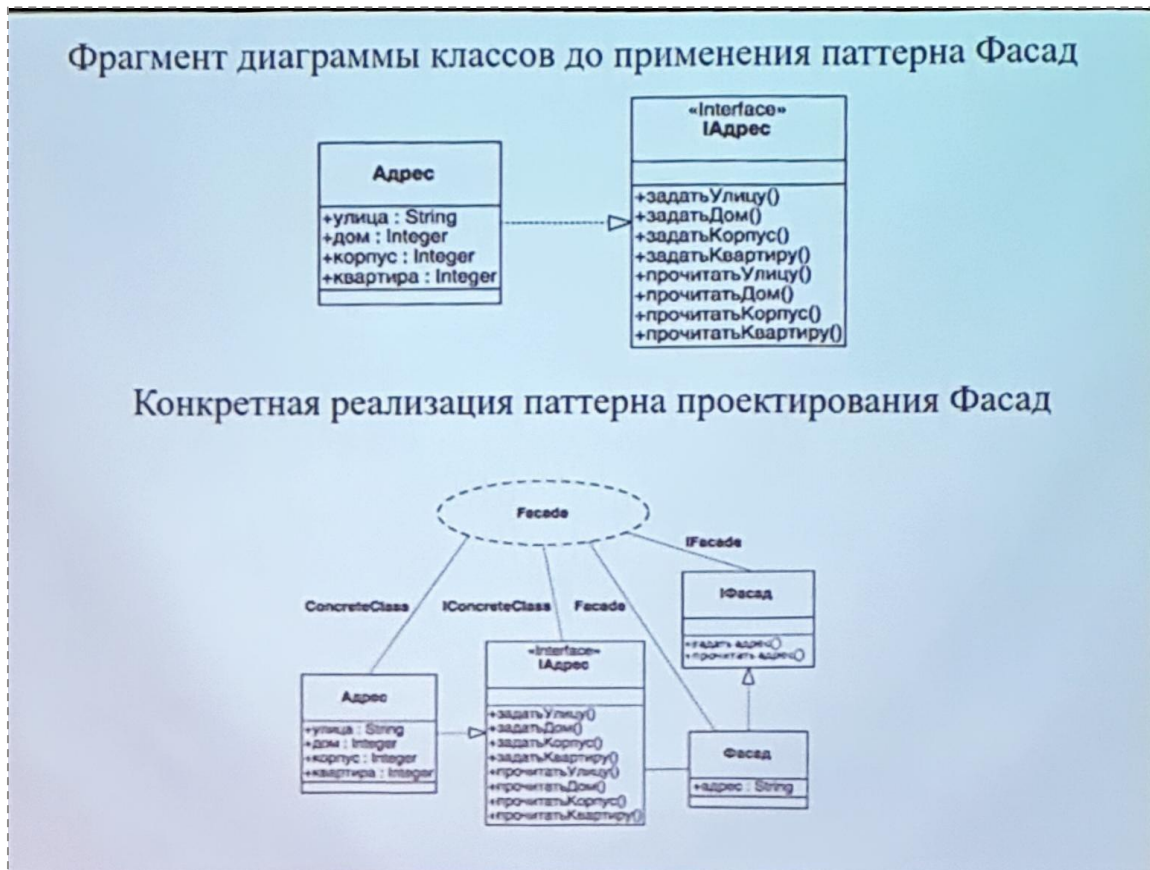
При изображении шаблона используется обозначение параметризованной кооперации языка UML.



В овале - параметризованная кооперация, в прямоугольнике - параметры кооперации.

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

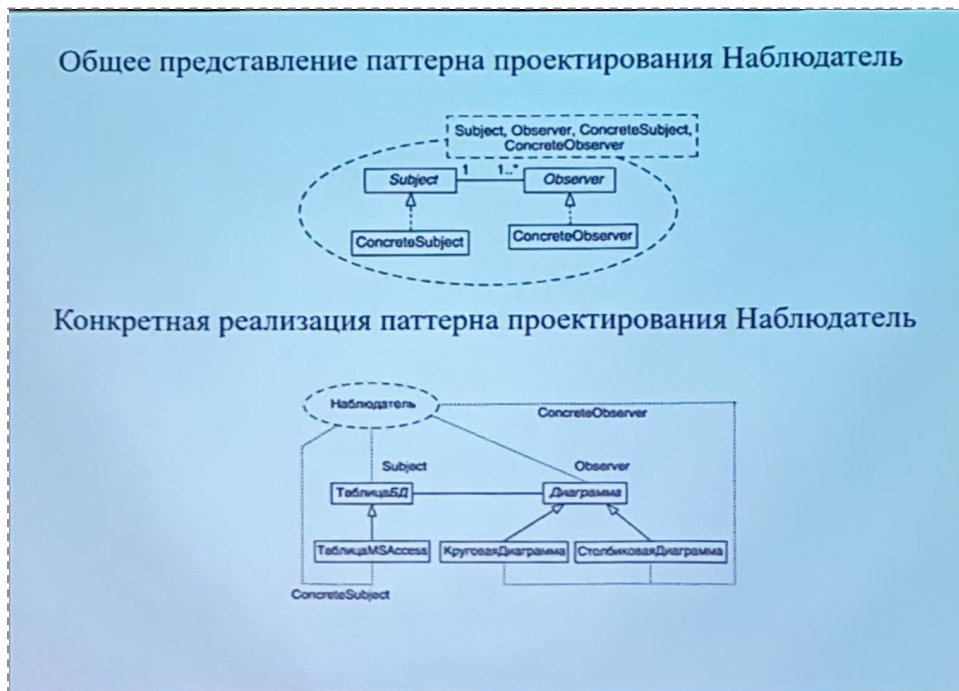
Шаблон фасад предназначен для замены нескольких разнотипных интерфейсов доступа к определенной подсистеме некоторым унифицированным интерфейсом, что существенно упрощает использование рассматриваемой подсистемы



Рассмотрен пример для применения шаблона фасад для выполнения операций по записи и считыванию адресов из БД сотрудников. Начальный фрагмент содержит два класса: адрес и интерфейс к операциям этого класса. При задании адреса нового сотрудника необходимо обратиться к этому интерфейсу и последовательно выполнить ряд операций используя в качестве аргумента ID сотрудника. Для получения информации об адресе сотрудника необходимо также обратиться к интерфейсу и прочитать по ID информацию о сотруднике. Очевидно, отслеживать при каждом обращении правильность выполнения этих последовательных операций не всегда удобно, поэтому данному фрагменту рекомендуют ввести еще один интерфейс: реализацию шаблона фасад.

При задании адреса нового сотрудника, в этом случае достаточно обратиться к интерфейсу IFасад и выполнить одну операцию: задать адрес, используя в качестве аргумента ID сотрудника. Для извлечения также выполняется одна операция, используя в качестве аргумента ID сотрудника. Реализацию данных операций данных операций необходимо предусмотреть в классе Facade.

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



Шаблон наблюдатель предназначен для контроля изменений состояния объекта и передачи информации об изменении этого состояния множеству клиентов. Конкретный класс следит за изменениями в БД и информирует об этом абстрактного наблюдателя. Тот информирует конкретных наблюдателей.

Использование шаблона наблюдатель не только упрощает взаимодействие между объектами соответствующих классов, но и позволяет вносить изменения в реализацию операций классов субъекта и наблюдателей независимо друг от друга. При этом процесс добавления или удаления наблюдателей никак не влияет на особенности реализации класса субъекта.

25. Декомпозиция программной системы на модули. Принцип модульности. Оценка сложности программной системы через принцип модульности. Затраты на работу с модулями

Два подхода для декомпозиции:

- модель потока данных
 - разбиение по функциям. При выборе этой модели получим набор функциональных модулей и определим связи между ними. Например, результат может быть похож на реализацию диаграммы потоков данных.
- объектно-ориентированная модель
 - основана на объектах (слабо сцепленных сущностях, имеющих собственные наборы данных, состояние, наборы операций) и их описаниях - классах. Объекты представляют классы.

Любую сложную проблему проще понять, разделив ее на части, каждую из которых можно решать и оптимизировать независимо.

Крайне редко программная система проектируется как единая неделимая система. Чаще всего проектирование предполагает разбиение системы на части, которые называются подсистемами или модулями. Декомпозиция или разбиение системы на модули, достаточно сложный и ответственный этап.

В основе декомпозиции - принцип модульности. **Принцип модульности** - наиболее общая демонстрация разделения понятий. Программная система делится на именуемые и адресуемые компоненты, часто называемые модулями, которые затем интегрируются для совместного решения проблемы.

Модуль - фрагмент программного текста, являющийся строительным блоком для физической структуры системы. Как правило, модуль состоит из интерфейсной части и части-реализации. Опишем сложность представив систему в виде иерархии

Первичные характеристики - количество модулей (n) и количество ребер/связей (e)

Высота - количество уровней в иерархии

Ширина - максимальное из количеств модулей в одном уровне управления

Введя те определения можно оценить сложность программной системы (это один из способов) /



Сложность программной системы оценим через модули и попытаемся её уменьшить за счёт декомпозиции. Характеристики сложности ПС в виде иерархии. Первичные характеристики ПС: количество модулей, количество ребер. Дополнительные характеристики: ширина и высота.

$N \approx n_1 * \log_2(n_1) + n_2 * \log_2(n_2)$, где n_1 - число различных операций, n_2 - число различных операндов.

$$V = N * \log_2(n_1 + n_2)$$

$$V(G) = E - N + 2$$

26. Определение модуля. Связность и сцепление модулей. Типы связности и сцепления модулей

Модуль - фрагмент программного текста, являющийся строительным блоком для физической структуры системы. Как правило, модуль состоит из интерфейсной части и части-реализации. Модуль как структурную единицу программной системы чаще всего оценивают связностью и сцеплением. Программная система делится на именуемые и адресуемые компоненты – модули.

Модуль - часть кода, состоит из интерфейсной части и реализации.

Связность - внутренняя характеристика модуля. Чем выше связность модуля, тем лучше результат проектирования (модуля).

Выделяют 7 типов связностей

1. связность по совпадению = 0: в модуле отсутствуют явно выраженные внутренние связи
2. логическая связность = 1: части модуля объединены по принципу функционального подобия
 - большая вероятность внесения ошибок при изменении сопряжения ради одной из функций
3. временная связность = 3: части модуля не связаны, но необходимы в один и тот же период работы системы
 - сильная взаимная связь с другими модулями => высокая чувствительность к внесению изменений
4. процедурная связность = 5: части модуля связаны порядком выполняемых ими действий, реализующих некоторый "сценарий поведения"
5. коммуникативная связность = 7: части модуля связаны по данным, работают с одной и той же структурой данных
6. информационная (последовательная) связность = 9: выходные данные одной части используются как входы следующей
7. функциональная связность = 10: части модуля вместе реализуют одну функцию

Связность 1-3 неправильное планирование проектного решения, тип связности 4 - небрежный.

Сцепление - мера взаимозаменяемости модулей в рамках программной системы, внешняя характеристика модуля, которую желательно уменьшить:

1. сцепление по данным = 1: все входные и выходные параметры вызываемого модуля обычные простые элементы данных
2. сцепление по образцу = 3: в качестве параметров используются структуры данных
3. сцепление по управлению = 4: модуль А явно управляет функционированием модуля В, посылая ему управляющие сигналы
4. сцепление по внешним ссылкам = 5: модуль А и В ссылаются на одни и те же элементы глобальных данных
5. сцепление по общей области = 7: если есть несколько модулей, то в данном случае они разделяют одну и ту же глобальную структуру данных
6. сцепление по содержанию = 9: один модуль прямо ссылается на содержание другого модуля.

27. Создание модели предметной области программной системы с помощью диаграммы классов

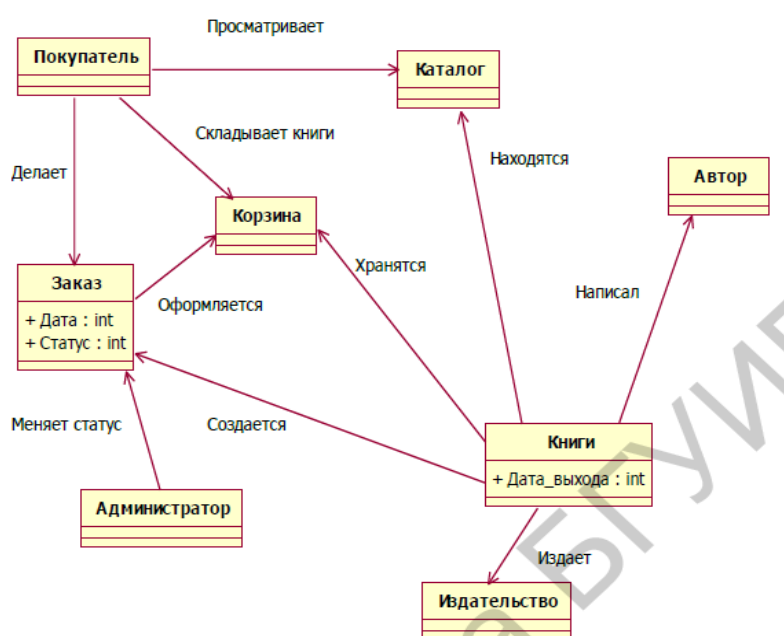
Одной из первых моделей, которую строят с помощью диаграммы class является модель предметной области. На ней показывают основные понятия, относящиеся к данной предметной области (примитив class). В каждом понятии указывается его имя и, где известно, атрибуты. Методы не заполняются. Поскольку классы на диаграмме должны быть связаны в диаграмму, используются самые общие связи для связывания классов.

На этапе построения модели предметной области вводятся основные понятия, связи между ними, словарь терминов, которые дальше будет использоваться. Модель предметной области является основой для следующих этапов.

Создание модели предметной области начинается уже на этапе определения требований (диаграмма Use Case) и завершается на этапе анализа. Для ее построения используется диаграмма классов, на которой отображены не классы разрабатываемой системы, а понятия предметной области и связи между ними. При этом достаточно использовать всего один тип связей: Directed Association.

Следует помнить, что найденные понятия - это только кандидаты на создание классов, поэтому модель предметной области нельзя напрямую преобразовать в диаграмму классов. В дальнейшем на этапе анализа и проектирования у выявленных объектов предметной области находят общие черты, свойства, иерархию, которая позволяет создавать непротиворечивую модель проектирования и реализации.

Модель предметной области отражает не все понятия, которые могут понадобиться для создания приложения. Для создания ПС определяется то подмножество объектов, которое необходимо для реализации прецедентов на текущей итерации. Возможно, впоследствии их количество будет изменено.



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

Модель анализа можно рассматривать как совокупность модели реализации прецедентов.

Модель анализа, как и модель предметной области, необходима для создания надежной и устойчивой архитектуры, а также понимания требований, предъявляемых к системе. Фактически модель анализа - это набор диаграмм, показывающих, как планируется реализовать в системе каждый прецедент. И хотя на этой модели уже присутствуют классы и связи между ними, они еще далеки до окончательного варианта системы.

Рассмотрим один из примеров реализации прецедента. На данном фрагменте показаны стереотипы классов:

- граничный класс - на границе взаимодействия внешними актерами и системы
- управление - координирует работу других классов приложения (чем-то управляют (круг со стрелкой/цикл))
- сущность - классы БД

Модель анализа является основой для декомпозиции разрабатываемой системы. Первый этап декомпозиции - выделение пакетов (частей) анализа. Пакеты выделяются согласно тем стереотипам, которые были определены на предыдущем шаге.

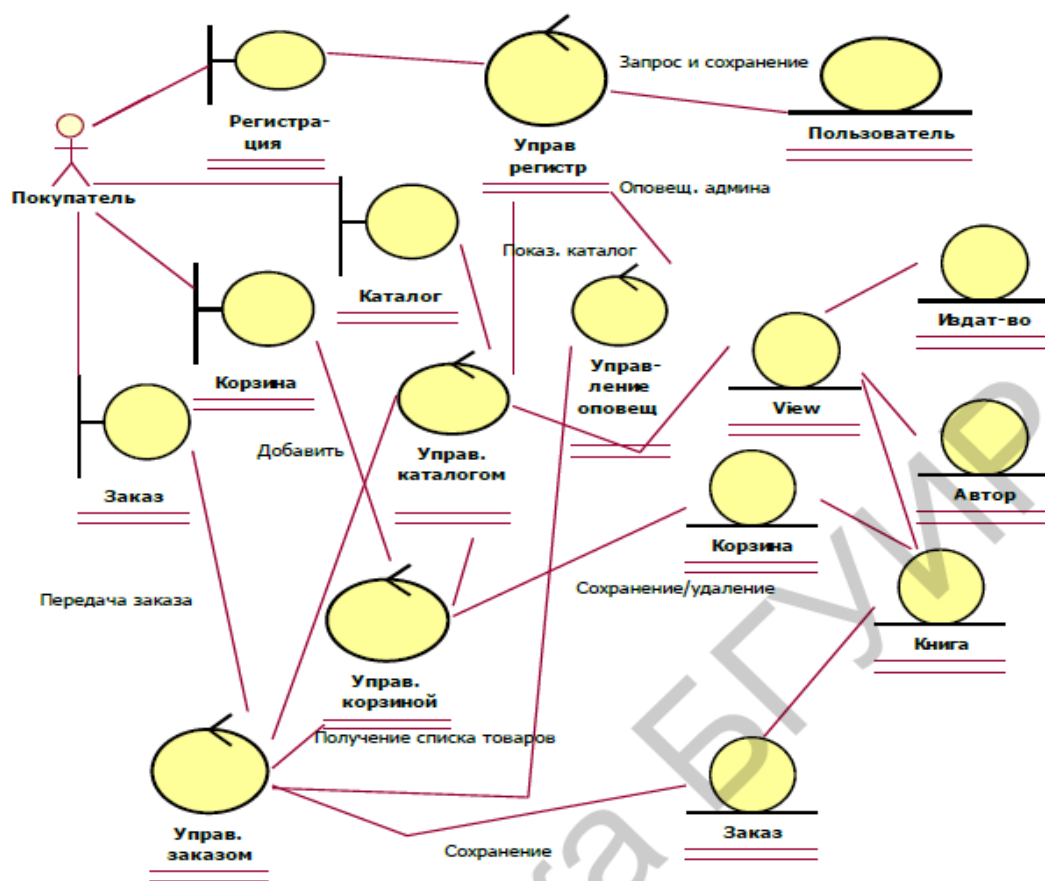
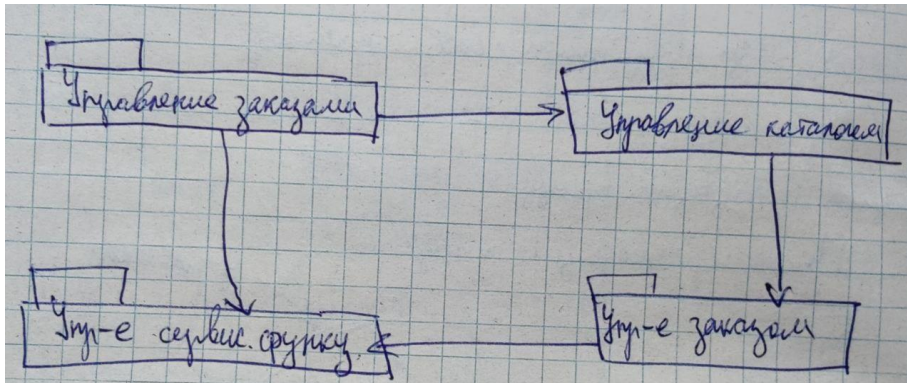


Рис. 2.14. Модель анализа книжного магазина

Пакеты анализа



Управление здесь != управлению в стереотипах

4 части: управление регистрацией, каталогом, сервисными функциями, заказом

Управление регистрацией - включает все классы, связанные с регистрацией, процесс регистрации обособлен и мало связан с другими процессами

Пакет управления регистрацией будет включать в себя все классы, которые относятся к регистрации. Управление каталогом включает все классы, позволяющие пользователю взаимодействовать с каталогом.

29. Декомпозиция системы. Правила выделения подсистем

Подсистема — это не просто пакеты, объединяющие проектируемые классы, в них ещё включаются реализации вариантов использования, интерфейсы и другие подсистемы. Разделение на подсистемы упрощает установку, разработку и конфигурацию. Создание подсистем позволяет проще устанавливать различные категории доступа к информации.

Правила выделения подсистем

1. ищут части системы, которые могут входить в нее как факультативные (необязательные) и помещают их в отдельную подсистему
2. подсистемы можно создавать по принципу горизонтального или вертикального разделения. В первом случае в подсистему попадает граничный класс и относящиеся к нему классы управления и классы-сущности. Во втором случае - в одну подсистему попадают все граничные классы, во вторую - все управления, в третью - все сущности
3. разделение на подсистемы может осуществляться для реализации функции взаимодействия с конкретным актером
4. подсистема может содержать все сильно связанные классы. Если в подсистему не укладывается какой-либо класс, то его делят на части, которые будут общаться между собой через интерфейс
5. если необходимо реализовать несколько уровней сервиса, которые реализуются различными интерфейсами, то каждый уровень сервиса попадает в отдельную подсистему
6. если необходимо реализовать выполнение системы для различных типов технических средств или ОС, то необходимо выделить функции, зависящие от конкретных программно-аппаратных средств в отдельные подсистемы

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

Процесс создания приложения - логическое продолжение построения кода класса. Мастер создания приложений 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*;
- класс просмотра *C***View*;
- класс оконной рамки *CMainFrame*.

Все приложения *VC++ MFC* являются объектами. Поэтому приложение – это главный класс, который включает в себя все необходимые для работы классы. Соблюдая соглашение об именах, мастер создает главный класс приложения с именем проекта, прибавляя к нему в начале букву *C*, а в конце *App* (в нашем случае это – *C***App*, где ***** – имя проекта). *C***App* наследуется из библиотечного класса *CWinApp*. Класс документа *C***Doc*, в котором должна проходить обработка данных, наследуется из библиотечного класса *CDocument*. Класс просмотра *C***View*, отображающий данные на экране компьютера, наследуется из библиотечного класса *CEditView*. Данные отображаются в окне класса *CMainFrame*, наследуемого из библиотечного класса *CFrameWnd*.

Таким образом, на основе стандартных классов документа, предоставляемых MFC, строится приложение, в котором необходимо будет только добавить функциональность, а за отображение документа на экране отвечает библиотека.

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

Одна из важных функциональных возможностей Rational Rose - обновление кода по модели и модели по коду. Обновление возможно только в ручном режиме. При необходимости обновления кода/модели пользователь открывает вкладку Update Code/Model и выбирает для каких классов он желает выполнить обновление (все или какие-то конкретные).

Model Assistant представляет собой окно, в котором создаются атрибуты, операции, а также задаются их свойства. В окне имеется ряд полей:

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

Инструмент *Component Assignment Tool* - специальное окно, компоненты, созданные в котором, будут содержать всю необходимую информацию для генерации кода на выбранном языке программирования. Кроме того, инструмент позволяет просмотреть классы, которые еще не назначены в компоненты.

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

Синхронизация м б автоматическая и не автоматическая (код и диаграммы)

Одним из отличий новых сред является возможность выполнения синхронизации моделей и кода, как в ручном, так и автоматическом режиме. По умолчанию - в ручном

Синхронизация:

- В момент сохранения модели
- В момент активизации модели
- В момент сохранения кода после изменения
- В момент активизации окна кода

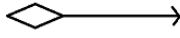
В Rational XDE синхронизация модели и кода возможны как в ручном, так и в автоматическом режиме. Каждый вариант синхронизации удобен для своего случая:

1. На этапе анализа и проектирования, когда основную работу по моделированию выполняет аналитик, синхронизации в момент сохранения модели позволит программистам, работающим в проекте, пользоваться самой последней версией структуры приложения;
2. На этапе реализации, когда основная работа ложится на программистов, установка синхронизации после сохранения внесения изменений в файлы кода позволит поддерживать модель системы в актуальном состоянии;
3. Когда же разработка закончена или создается новая версия уже работающей системы, чтобы не нарушить работу программы, с может быть вообще отключена.

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

WebModeler - подключаемый модуль, предназначенный для автоматизации разработки Web-приложений с помощью Rational Rose. Чтобы подключить WebModeler необходимо преобразовать диаграмму классов в веб-нотацию. (Никакой новой диаграммы не появляется!). В веб-нотации имеется 3 основных инструмента:

1. Клиентская страница
2. Серверная страница
3. Форма

Также появляется новая связь - "Link" - однонаправленная ассоциация и агрегирование. Она показывает, что форма принадлежит странице (). - то, что она на ЛК говорила. (Дальше из эруда) другие новые связи:

- *Submit* определяет, что форма взаимодействует со страницей на сервере и передает ей данные.
- *Build* показывает, что страница создается сервером.
- *Redirect* определяет передачу управления одной серверной странице от другой и используется для страниц ASP.
- *Includes* означает, что одна страница включается в другую.
- *Forward* похожа на связь Redirect, только используется для страниц JSP.

Основной недостаток WebModeler'а в Rational Rose — возможность строить только статические веб-страницы. Также WebModeler позволяет генерировать код для модели.

34. Возможности и особенности построения Web-модели в среде Rational XDE

WebModeler предназначен для базового проектирования веб-приложений. После подключения переводим среду в веб-нотацию, после этого диаграмма классов(инструментарий) преобразуется в веб-нотацию.

В Rational XDE, как и в Rational Rose имеется модуль WebModeler, который предназначен для непосредственного проектирования веб приложений. Набор инструментов модуля существенно *расширен*. Кроме статических страниц можно проектировать и *динамические* страницы.

При помощи сложных инструментов создаются наборы элементов, связанных между собой и содержащих необходимые элементы для создания .NET приложения.

- формы
- серверные страницы
- классы

1 связь - однонаправленная ассоциация. Разнообразить связь можно через свойства. После построения по диаграмме можно выполнить кодогенерацию

- DataModeler - позволяет спроектировать БД

35. Понятие проектных рисков. Действия по управлению рисками

Управление рисками:

$$RE = P(UO) * L(UO)$$

где

RE - показатель риска

P(UO) - вероятность неудовлетворительного результата

L(UO) - потеря при неудовлетворительном результате

Выполнение любого проекта сопровождается рисками, которым надо уметь управлять.

Действия по управлению рисками:

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

Идентификация риска:

- проектный риск
- технический риск
- коммерческий риск

Источники проектного риска:

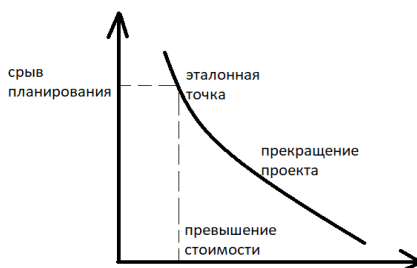
- выбор бюджета, плана, человеческий ресурс
- формирование требований к программному проекту
- сложность, размер и структура программного проекта
- методика взаимодействия с заказчиком

Источники технического риска:

- трудности проектирования, реализации формирования интерфейса, тестирования и сопровождения
- неточность спецификаций
- техническая неопределенность или отсталость принятого решения

Источники коммерческого риска:

- создание продукта, не требующегося на рынке
- создание продукта, опережающего требования рынка
- потеря финансирования



Если отношение времени и денег в эталонной точке ниже графика, то ещё всё нормально и можно работать. На кривой - граничное состояние (можно подумать). Выше кривой - надо прекращать проект.

36. Статический и динамический аспекты Rational Unified Process (RUP)

Статические аспекты - этапы ЖЦ.

Динамические аспекты - временные промежутки (минуты, дни, часы и т. д.)

Элементы статического аспекта:

1. **Роль** определяет поведение и ответственность личности или группы личностей, составляющих проектную команду. Одна личность может играть в проекте много различных ролей
2. **Вид деятельности** — единица выполняемой работы, соответствует понятию технологической операции.
3. **Артефакты** (рабочие продукты) — некоторые продукты проекта, порождаемые или используемые в нем при работе над окончательным продуктом
4. **Дисциплина** соответствует понятию технологического процесса и представляет собой последовательность действий, приводящую к получению значимого результата

Элементы динамического аспекта:

1. Циклы
2. Фазы
3. Итерации

Основные дисциплины:

1. построение бизнес-моделей;
2. определение требований;
3. анализ и проектирование;
4. реализация;
5. тестирование;
6. развертывание.

Вспомогательные дисциплины:

1. управление конфигурацией и изменениями;
2. управление проектом;
3. создание инфраструктуры.

Динамическая срань:

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.

37. Принципы и стадии разработки ПС в технологии Rational Unified Process

Rational Unified Process - технология, соответствующая стандартам и нормативным документам, связанным с процессами ЖЦ ПО и оценкой технологической зрелости организаций-разработчиков.

Основные принципы:

1. итерационный и инкрементный подход к созданию ПО
2. планирование и управление проектом на основе функциональных требований к системе
3. построение системы на базе архитектуры ПО

Согласно RUP, ЖЦ ПО разбивается на отдельные циклы, в каждом из которых создается новое поколение продукта. Каждый цикл, в свою очередь разбивается на четыре последовательные стадии:

1. начальная стадия
2. стадия разработки
3. стадия конструирования
4. стадия ввода в действие

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

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

Начальная стадия может принимать множество разных форм. Для крупных проектов она связана со всесторонним изучением всех возможностей реализации проекта. В это же время вырабатывается бизнес-план проекта: определяется, сколько приблизительно он будет стоить и какой доход принесет. Кроме того, выполняется начальный анализ для оценки размеров проекта.

Результаты:

- общее описание системы: основные требования к проекту, его характеристики и ограничения
- начальная модель вариантов использования (степень готовности 10-20%)
- начальный проектный глоссарий (словарь терминов)
- начальный бизнес-план
- план проекта, отражающий стадии и итерации
- один или несколько прототипов

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

Результаты:

- модель вариантов использования (завершения на 80%), 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
 - модель предметной области, которая отражает понимание бизнеса и является отправным пунктом для формирования основных классов предметной области
 - технологическая платформа, определяющая основные элементы технологии реализации системы и их взаимодействие
- работающий прототип
- уточненный бизнес-план
- план разработки всего проекта, отражающий итерации и критерии оценки для каждой итерации

Основными признаками завершения стадии разработки являются два события:

- разработчики в состоянии оценить с достаточно высокой точностью, сколько времени потребуется на реализацию каждого варианта использования
- идентифицированы все наиболее серьезные риски, и степень понимания наиболее важных из них такова, что известно, как справиться с ними.

39. Содержание и результаты третьей и четвертой стадий в технологии Rational Unified Process

Стадия конструирования:

- итерации являются инкрементными в соответствии с той функцией, которую они выполняют. Каждая итерация добавляет очередные конструкции к вариантам использования, реализованным во время предыдущих итераций
- итерации являются повторяющимися по отношению к разрабатываемому коду. На каждой итерации некоторая часть существующего кода переписывается с целью сделать его более гибким

Результат:

продукт, готовый к передачи конечным пользователям

Как минимум он содержит следующее:

- ПО, интегрированное на требуемых платформах
- руководства пользователя
- описание текущей реализации

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

- Бета-тестирование, позволяющее убедиться, что новая система соответствует ожиданиям пользователей;
- Параллельное функционирование с существующей системой, которая подлежит постепенной замене;
- Конвертирование БД;
- Оптимизацию производительности;
- Обучение пользователей и специалистов службы сопровождения.

40. Этапы создания программных средств в технологии Oracle.

Методическая основа - метод Oracle - комплекс методов, охватывающий большинство процессов ЖЦ ПО. В данный комплекс входят компоненты разработки прикладного ПО CDM (Custom Development Method), управления проектом (PJM), внедрения прикладного ПО (AIM), реинжиниринг бизнес-процессов (BPR), управление изменениями (OCM). 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, каждый из которых выполняется в течение нескольких этапов:

1. **стратегия** (определение требований)
2. **анализ** (формулирование детальных требований к системе)
3. **проектирование**. Разрабатывается подробная архитектура системы, проектируются схема реляционной БД и программные модули, устанавливаются перекрестные ссылки между компонентами системы для анализа их взаимного влияния и контроля за изменениями
4. **реализация**. Создается БД, строятся прикладные системы, производится их тестирование, проверка качества и соответствия требованиям пользователей. Создается системная документация, материалы для обучения и руководства пользователей
5. **внедрение**
6. **эксплуатация**

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

41. Процессы создания программных средств в технологии Oracle.

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

42. Сравнительный анализ технологий создания ПС Rational Unified Process, Oracle, нахуй этот ваш Borland

В **RUP** разработка ведётся циклами (в каждом 4 стадии: начальная, разработка, конструирование, ввод в действие) + *есть статические (роль, вид деятельности, дисциплина, продукт или артефакт) и динамические аспекты (цикл, фазы, итерации).*

В Oracle существует множество компонентов, но основной - **CDM (Custom Development Method)**. Там **11 процессов** (*определение бизнес-требований, изучение существующих систем, определение технической архитектуры, проектирование и разработка БД, проектирование и разработка модулей, конвертация данных из старой системы, документирование, тестирование, обучение, внедрение в эксплуатацию, поддержка и сопровождение*), **6 возможных этапов - аналоги стадий из RUP** (*стратегия, анализ, проектирование, реализация, внедрение, эксплуатация*).

Процессы CDM являются аналогом статических аспектов RUP. Этапы являются аналогом динамических аспектов RUP.

43. Понятия CASE-средство, CASE-система, CASE-технология, CASE-индустрия и различия между ними

CASE-средства – ПС, обеспечивающие поддержку многочисленных технологий проектирования информационных систем, охватывая всевозможные средства автоматизации и часть ЖЦ ПО.

CASE-система – набор CASE-средств, имеющих определенное функциональное предназначение и выполненных в рамках единого ПП. Обеспечивает автоматизацию всего ЖЦ ПО.

CASE-технология представляет собой **совокупность методологий** анализа, проектирования, разработки и сопровождения сложных систем и поддерживается комплексом взаимосвязанных средств автоматизации. Основная цель CASE-подхода – разделить и максимально автоматизировать все этапы разработки ПС.

CASE-индустрия объединяет **сотни фирм и компаний** различной направленности деятельности. Практически все серьезные зарубежные программные проекты осуществляются с использованием CASE-средств, а общее число распространяемых пакетов превышает 500 наименований.

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.

То, что она спрашивает на экзамене

Спрашивает про диаграмму в связке с принципами ООП. Например, **какие принципы ООП можно увидеть на диаграмме?**

UseCase: освобождена от всех принципов ООП

Deployment: освобождена от всех принципов ООП

Statechart: параллелизм, абстрагирование

Activity: параллелизм, абстрагирование

Sequence: абстрагирование

Component: модульность, инкапсуляция, абстрагирование

Class: наследование, абстрагирование

Collaboration: абстрагирование

Главный принцип объектно-ориентированного подхода?

Ответ: абстрагирование

Самая важная диаграмма - CLASS диаграмма классов!!!

Что нужного показано на 4-х диаграммах для диаграммы классов?

Ответ: Объекты системы.

Диаграммы по важности:

1. Class
2. UseCase

Если есть одна нотация UML и мы как диаграммы, так и шаблоны отображаем с помощью одних примитивов, не лучше ли выбрать что-то одно?

Ответ: диаграммы - на этапе проектирования, а когда мы на этапе перехода к программированию, то мы можем что-то уточнить с помощью шаблонов.

Как выстроить модули (я так понимаю, вопрос про кол-во модулей, когда мы декомпозируем подсистему на модули)?

Ответ: ~10 (но вообще, сколько посчитаем нужным из собственного опыта).

Спрашивает про наши проектные решения □

Какого принципа мы придерживаемся?

Ответ: унифицированного процесса разработки:

- выбор архитектуры (на раннем этапе)
- планирование и управление проектом на основе функциональных требований к системе
- итерационность и инкрементность

Почему в такой последовательности?

Зачем мы вообще строим диаграммы?

Ответ?: unified process предполагает построение архитектуры как основной шаг, так как на нее опирается программный код. Плюс так происходит автоматизация (генерация кода и тд) и реализация принципов ООП (инкапсуляция и т.д.)

Мы их строим применительно к объектам системы.

В наборе диаграмм должна появиться еще одна? Web diagram? (Про Rational XDE)

Ответ: нет, это просто дополнительная опция (модуль) и с его помощью мы переводим диаграмму классов в веб-нотацию.

Знать надо про Case-среды и особенности работы, диаграммы желательно со своими примерами.

На что мы опираемся, выбирая технологии?

Ответ: автоматизация, ЖЦ, ООП подход.

Процессы CDM являются аналогом статических аспектов RUP. Этапы являются аналогом динамических аспектов RUP.