ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ОРЕНБУРГСКИЙ ГОСУДАРСТВЕННЫЙ АГРАРНЫЙ УНИВЕРСИТЕТ»

Кафедра «Автоматизированных систем обработки информации и управления»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ОБУЧАЮЩИХСЯ ПО ОСВОЕНИЮ ДИСЦИПЛИНЫ

Б1.В.ДВ.11.02 Основы компьютерного моделирования

Направление подготовки 27.03.04 Управление в технических системах

Профиль подготовки «Информационные управляющие комплексы систем безопасности объектов»

Квалификация (степень) выпускника бакалавр

Форма обучения заочная

Оренбург 201_ г.

СОДЕРЖАНИЕ

1. КОНСПЕКТ ЛЕКЦИЙ

1.1 Лекция № 1 (2 часа).
Тема: «Общие положения**»**
1.1.1 Вопросы лекции:
1. Объект проектирования.
2. Процесс проектирования.

1.1.2 Краткое содержание вопросов:

1. Объект проектирования.

Различные виды проектирования ориентированы на создание и преобразование разных объектов и предметов. *Объект проектирования* — это среда или процесс, в контексте которых находится предмет. *Предмет проектирования* — это предполагаемый продукт, образ которого первоначально представлен в проекте. Это то, созданию чего посвящена проектная деятельность. Объект и предмет проектирования соотносятся между собой как общее и частное.

2. Процесс проектирования.

Проектирование—это комплекс работ с целью получения описаний нового или модернизируемого технического объекта, достаточных для реализации или изготовления объекта в заданных условиях. Объектами проектирования могут быть изделия (например, обрабатывающий центр, двигатель внутреннего сгорания, ЭВМ) или процессы (например, технологические, вычислительные). Комплекс проектных работ включает в себя теоретические и экспериментальные исследования, расчеты, конструирование.

Проектирование, осуществляемое человеком при взаимодействии с ЭВМ, называют *автоматизированным.* Степень автоматизации может быть различной и оценивается долей s проектных работ, выполняемых на ЭВМ без участия человека. При s=0 проектирование называют *неавтоматизированным,* при s=1—*автоматическим.*

Автоматизированное проектирование осуществляется в рамках САПР. В соответствии с ГОСТом *система автоматизированного проектирования*—это организационно-техническая система, состоящая из комплекса средств автоматизации проектирования (АП), взаимодействующего с подразделениями проектной организации, и выполняющая автоматизированное проектирование.

Проектирование делится на стадии, этапы и процедуры. При проектировании сложных объектов выделяют стадии

- научно-исследовательских работ (НИР)
- опытно-конструкторских работ (ОКР)
- технического проекта
- рабочего проекта
- испытаний опытного образца.

Стадию *НИР* во многих случаях можно разделить на стадии

- предпроектных исследований
- технического задания
- технического предложения.

На этих стадиях последовательно изучаются потребности в получении новых изделий с заданным целевым назначением, исследуются физические, информационные, конструктивные и технологические принципы построения изделий. Далее исследуются возможности реализации этих принципов, прогнозируются возможные значения

характеристик и параметров объектов. *Результатом НИР является формулировка технического задания (ТЗ) на разработку нового объекта.*

На стадии *ОКР* разрабатывается эскизный проект изделия, проверяются, конкретизируются и корректируются принципы и положения, установленные на стадии НИР.

На стадии *технического проекта* принимаются подробные технические решения и прорабатываются все части проекта.

На стадии *рабочего проекта* создается полный комплект конструкторско-технологической документации, достаточный для изготовления объекта.

На стадии *испытаний опытного образца* (или пробной партии при крупносерийном производстве) получают результаты, позволяющие выявить возможные ошибки и недоработки проекта, принимаются меры к их устранению, после чего документация передается на предприятия, выделенные для серийного производства изделий.

Проектирование разделяется также на этапы. Используются при этом следующие понятия. *Проектное решение*—описание объекта или его составной части, достаточное для рассмотрения и принятия заключения об окончании проектирования или путях его продолжения. *Проектная процедура*—часть проектирования, заканчивающаяся получением проектного решения. Примерами проектных процедур служат синтез функциональной схемы устройства, оптимизация параметров функционального узла, трассировка межсоединений на печатной плате и т. п. *Этап проектирования* —это условно выделенная часть проектирования, сводящаяся к выполнению одной или нескольких проектных процедур, объединенных по признаку принадлежности получаемых проектных решений к одному иерархическому уровню и (или) аспекту описаний.

На любой стадии или этапе проектирования можно выявить ошибочность или неоптимальность ранее принятых решений и, следовательно, необходимость или целесообразность их пересмотра. Подобные возвраты характерны для проектирования и обусловливают его *итерационный характер*. Может быть также выявлена необходимость корректировки ТЗ. Вводят понятия *процедур внешнего и внутреннего проектирования.* К внешнему проектированию относят процедуры формирования или корректировки технического задания, а к внутреннему—процедуры реализации сформированного ТЗ. Тогда можно сказать, что происходит чередование процедур внешнего и внутреннего проектирования, что особенно характерно для ранних стадий (НИР, ОКР). При этом различают нисходящее (сверху вниз) и восходящее (снизу вверх) проектирование. При *нисходящем проектировании* задачи высоких иерархических уровней решаются прежде, чем задачи более низких иерархических уровней. При *восходящем проектировании* последовательность противоположная. Функциональное проектирование сложных систем чаще всего является нисходящим, конструкторское проектирование—восходящим.

1.2 Лекция №2 (2 часа).

Тема:« Основы унифицированного языка моделирования (UML)»

1.2.1 Вопросы лекции:

1. Классификация диаграмм, принятые обозначения.
2. Изображение ассоциаций на диаграммах классов.
3. Иерархии классов.

1.2.2 Краткое содержание вопросов:

1. Классификация диаграмм, принятые обозначения.

UML 2 описывает 13 официальных типов диаграмм, перечисленных в табл. 1.1, классификация которых приведена на рис. 1.2. Хотя эти виды диаграмм отражают различные подходы многих специалистов к UML и способ организации моей книги, авторы UML не считают диа граммы центральной составляющей языка. Поэтому диаграммы опре делены не очень строго. Часто вполне допустимо присутствие элементов диаграммы одного типа в другой диаграмме. Стандарт UML указывает, что определенные элементы обычно рисуются в диаграммах соответствующего типа, но это не догма.

Допустимый UML – это язык, определенный в соответствии со спецификацией. Однако на практике ответ несколько сложнее.

- Структурные диаграммы:
  - диаграммы классов (class diagrams) предназначены для моделирования структуры объектно-ориентированных приложений - классов, их атрибутов и заголовков методов, наследования, а также связей классов друг с другом;
  - диаграммы компонент (component diagrams) используются при моделировании компонентной структуры распределенных приложений; внутри каждая компонента может быть реализована с помощью множества классов;
  - диаграммы объектов (object diagrams) применяются для моделирования фрагментов работающей системы, отображая реально существующие в runtime экземпляры классов и значения их атрибутов;
  - диаграммы композитных структур (composite structure diagrams) используются для моделирования составных структурных элементов моделей - кооперaций, композитных компонент и т.д.;
  - диаграммы развертывания (deployment diagrams) предназначены для моделирования аппаратной части системы, с которой ПО непосредственно связано (размещено или взаимодействует);
  - диаграммы пакетов (package diagrams) служат для разбиения объемных моделей на составные части, а также (традиционно) для группировки классов моделируемого ПО, когда их слишком много.
- Поведенческие диаграммы:
  - диаграммы активностей (activity diagrams) используются для спецификации бизнес-процессов, которые должно автоматизировать разрабатываемое ПО, а также для задания сложных алгоритмов;
  - диаграммы случаев использования(use case diagrams) предназначены для "вытягивания" требований из пользователей, заказчика и экспертов предметной области;
  - диаграммы конечных автоматов (state machine diagrams) применяются для задания поведения реактивных систем;
  - диаграммы взаимодействий (interaction diagrams):
    - диаграммы последовательностей (sequence diagrams) используются для моделирования временных аспектов внутренних и внешних протоколов ПО;
    - диаграммы схем взаимодействия (interaction overview diagrams) служат для организации иерархии диаграмм последовательностей;

▪ диаграммы коммуникаций (communication diagrams) являются аналогом диаграмм последовательностей, но по-другому изображаются (в привычной, графовой, манере);

▪ временные диаграммы (timing diagrams) являются разновидностью диаграмм последовательностей и позволяют в наглядной форме показывать внутреннюю динамику взаимодействия некоторого набора компонент системы.

Существенным в вопросе является то, на каких правилах базируется UML: описательных или предписывающих. Язык с **предписывающими правилами** (prescriptiverules) управляется официальной основой, которая устанавливает, что является, а что не является допустимым языком, и какое значение вкладывается в понятие высказывания языка. Язык с **описательными правилами** (descriptiverules) – это язык, правила которого распознаются по тому, как люди применяют его на практике. Языки программирования в основном имеют предписываю щие правила, установленные комитетом по стандартам или основными поставщиками, тогда как естественные языки, такие как английский, в основном имеют описательные правила, смысл которых устанавливается по соглашению.

UML – точный язык, поэтому можно было бы ожидать, что он основан на предписывающих правилах. Но UML часто рассматривают как программный эквивалент чертежей из других инженерных дисциплин, а эти чертежи основаны не на предписывающих нотациях. Никакой комитет не говорит, какие символы являются законными встроитель ной технической документации; эти нотации были приняты по соглашению, как и в естественном языке. Стандарты сами по себе еще ничего не решают, поскольку те, кто работает в этой области, не смогут следовать всему, что указывают стандарты; это то же самое, что спрашивать французов о французской академии наук. К тому же язык UML настолько сложен, что стандарты часто можно трактовать по разному. Даже ведущие специалисты по UML, которые рецензировали эту книгу, не согласились бы интерпретировать стандарты.

Этот вопрос важен и для меня, пишущего эту книгу, и для вас, применяющих язык UML. Если вы хотите понять диаграммы UML, важно уяснить, что понимание стандартов – это еще не вся картина. Люди принимают соглашения и в индустрии в целом, и в каких-то конкретных проектах. Поэтому, хотя стандарт UML и может быть первичным источником информации по UML, он не должен быть единственным.

Моя позиция состоит в том, что для большинства людей UML имеет описательные правила. Стандарт UML оказывает наибольшее влияние на содержание UML, но это делает не только он. Я думаю, что особенно верным это станет для UML 2, который вводит некоторые соглашения по обозначениям, конфликтующие или с определениями UML 1, или с использованием по соглашению, а также еще больше усложняет язык. Поэтому в данной книге я стараюсь обобщить UML так, как я его вижу: и стандарты, и применение по соглашению. Когда мне приходится указывать на некоторое отличие в этой книге, я буду употреблять термин **применение по соглашению** (conventionaluse), чтобы обо значить то, чего нет в стандарте, но, как я думаю, широко применяется. В случае если что то соответствует стандарту, я буду употреблять термин **стандартный** (standard) или **нормативный** (normative). (Нормативный – это термин, посредством которого люди обозначают утверждение, которое вы должны подтвердить, чтобы оно соответствовало стандарту.Поэтому выражение ненормативный UML – это своеобразный способ сказать, что нечто совершенно неприемлемо с точки зрения стандарта UML.)

Рассматривая диаграмму UML, необходимо помнить, что основной принцип UML заключается в том, что любая информация на конкрет ной диаграмме может быть подавлена. Это подавление может носить глобальный характер – скрыть все атрибуты – или локальный – не показывать какие-нибудь конкретные классы. Поэтому по диаграмме вы никогда не можете судить о чем-нибудь по его отсутствию. Даже если метамодель UML имеет поведение по умолчанию, например [1] для атрибутов, когда вы не видите эту

информацию на диаграмме, это может быть обусловлено либо поведением по умолчанию, либо тем, что она просто подавлена.

Говоря это, следует упомянуть, что существуют основные соглашения, например о том, что многозначные свойства должны быть множествами.

Не надо слишком зацикливаться на допустимом UML, если вы занимаетесь эскизами или моделями. Важнее составить хороший проект системы, и я предпочитаю иметь хороший дизайн в недопустимом UML, чем допустимый UML, но плохой дизайн. Очевидно, хороший и допустимый предпочтительнее, но лучше направить свою энергию на разработку хорошего проекта, чем беспокоиться о секретах UML. (Конечно, в случае применения UML в качестве языка программирования необходимо соблюдать стандарты, иначе программа будет работать неправильно!)

Смысл UML

Одним из затруднений в UML является то, что хотя спецификация по дробно описывает определение правильно сформированного UML, но этого недостаточно, чтобы определить значение UML вне сферы изысканного выражения «метамодель UML». Не существует формальных описаний того, как UML отображается на конкретные языки программирования. Вы не можете посмотреть на диаграмму UML и *точно* сказать, как будет выглядеть соответствующий код. Однако у вас может быть *приблизительное представление* о виде программы. На практике этого достаточно. Команды разработчиков часто формируют собст венные локальные соглашения, и, чтобы их использовать, вам придет ся с ними познакомиться.

2. Изображение ассоциаций на диаграммах классов.

Диаграммы классов используются при моделировании ПС наиболее часто. Они являются одной из форм статического описания системы с точки зрения ее проектирования, показывая ее структуру. Диаграмма классов не отображает динамическое поведение объектов изображенных на ней классов. На диаграммах классов показываются классы, интерфейсы и отношения между ними.

Представление классов

Класс – это основной строительный блок ПС. Это понятие присутствует и в ОО языках программирования, то есть между классами UML и программными классами есть соответствие, являющееся основой для автоматической генерации программных кодов или для выполнения реинжиниринга. Каждый класс имеет название, атрибуты и операции. Класс на диаграмме показывается в виде прямоугольника, разделенного на 3 области. В верхней содержится название класса, в средней – описание атрибутов (свойств), в нижней – названия операций – услуг, предоставляемых объектами этого класса.

Атрибуты класса определяют состав и структуру данных, хранимых в объектах этого класса. Каждый атрибут имеет имя и тип, определяющий, какие данные он представляет. При реализации объекта в программном коде для атрибутов будет выделена память, необходимая для хранения всех атрибутов, и каждый атрибут будет иметь конкретное значение в любой момент времени работы программы. Объектов одного класса в программе может быть сколь угодно много, все они имеют одинаковый набор атрибутов, описанный в классе, но значения атрибутов у каждого объекта свои и могут изменяться в ходе выполнения программы.

Для каждого атрибута класса можно задать видимость (visibility). Эта характеристика показывает, доступен ли атрибут для других классов. В UML определены следующие уровни видимости атрибутов:

- Открытый (public) – атрибут виден для любого другого класса (объекта);
- Защищенный (protected) – атрибут виден для потомков данного класса;
- Закрытый (private) – атрибут не виден внешними классами (объектами) и может использоваться только объектом, его содержащим.

Последнее значение позволяет реализовать свойство инкапсуляции данных. Например, объявив все атрибуты класса закрытыми, можно полностью скрыть от внешнего мира его данные, гарантируя отсутствие несанкционированного доступа к ним. Это позволяет сократить число ошибок в программе. При этом любые изменения в составе атрибутов класса никак не скажутся на остальной части ПС.

Класс содержит объявления **операций**, представляющих собой определения запросов, которые должны выполнять объекты данного класса. Каждая операция имеет **сигнатуру**, содержащую имя операции, тип возвращаемого значения и список параметров, который может быть пустым. Реализация операции в виде процедуры – это метод, принадлежащий классу. Для операций, как и для атрибутов класса, определено понятие «видимость». Закрытые операции являются внутренними для объектов класса и недоступны из других объектов. Остальные образуют интерфейсную часть класса и являются средством интеграции класса в ПС.

Отношения

На диаграммах классов обычно показываются ассоциации и обобщения (см. предыдущую статью).

Каждая **ассоциация** несет информацию о связях между объектами внутри ПС. Наиболее часто используются бинарные ассоциации, связывающие два класса. Ассоциация может иметь название, которое должно выражать суть отображаемой связи (см. рис. 2). Помимо названия, ассоциация может иметь такую характеристику, как **множественность**. Она показывает, сколько объектов каждого класса может участвовать в ассоциации. Множественность указывается у каждого конца ассоциации (полюса) и задается конкретным числом или диапазоном чисел. Множественность, указанная в виде звездочки, предполагает любое количество (в том числе, и ноль).

Ассоциация сама может обладать свойствами класса, то есть, иметь атрибуты и операции. В этом случае она называется класс-ассоциацией и может рассматриваться как класс, у которого помимо явно указанных атрибутов и операций есть ссылки на оба связываемых ею класса. В примере на рис.2 ассоциация «включает» по существу есть класс-ассоциация, укоторой есть атрибут «Количество», показывающий, сколько единиц каждого товара входит в набор.

Обобщение на диаграммах классов используется, чтобы показать связь между классом-родителем и классом-потомком. Оно вводится на диаграмму, когда возникает разновидность какого-либо класса (например, при развитии ПС – см. рис.4), а также в тех случаях, когда в системе обнаруживаются несколько классов, обладающих сходным поведением.

Как уже говорилось ранее, UML позволяет строить модели с различным уровнем детализации.

Стереотипы классов

При создании диаграмм классов часто пользуются понятием «стереотип». В дальнейшем речь пойдет о стереотипах классов. **Стереотип** класса – это элемент расширения словаря UML, который обозначает отличительные особенности в использовании класса. Стереотип имеет название, которое задается в виде текстовой строки. При изображении класса на диаграмме стереотип показывается в верхней части класса в двойных угловых скобках. Есть четыре стандартных стереотипа классов, для которых предусмотрены специальные графические изображения (см. рис.5).

Стереотип используется для обозначения классов-сущностей (классов данных), стреотип описывает пограничные классы, которые являются посредниками между ПС и внешними по отношению к ней сущностями – актерами, обозначаемыми стереотипом <>. Наконец, стереотип описывает классы и объекты, которые управляют взаимодействиями. Применение стереотипов позволяет, в частности, изменить вид диаграмм классов.

Применение диаграмм классов

Диаграммы классов создаются при логическом моделировании ПС и служат для следующих целей:

- Для моделирования данных. Анализ предметной области позволяет выявить основные характерные для нее сущности и связи между ними. Это удобно моделируется с помощью диаграмм классов. Эти диаграммы являются основой для построения концептуальной схемы базы данных.
- Для представления архитектуры ПС. Можно выделить архитектурно значимые классы и показать их на диаграммах, описывающих архитектуру ПС.
- Для моделирования навигации экранов. На таких диаграммах показываются пограничные классы и их логическая взаимосвязь. Информационные поля моделируются как атрибуты классов, а управляющие кнопки – как операции и отношения.
- Для моделирования логики программных компонент (будет описано в последующих статьях).
- Для моделирования логики обработки данных.

3. Иерархии классов.

Самым распространенным видом отношения зависимости является соединение между классами, когда один класс использует другой в качестве параметра операции.

Для моделирования такого отношения изобразите зависимость, направленную от класса с операцией к классу, используемому в качестве ее параметра.

Одиночное наследование

Моделируя словарь системы, вам часто придется работать с классами, похожими на другие по структуре и поведению. В принципе их можно моделировать как различные, независимые друг от друга абстракции. Но лучше выделить одинаковые свойства и сформировать на их основе общие классы, которым наследуют специализированные.

Моделирование отношений наследования осуществляется в таком порядке:

1. Найдите атрибуты, операции и обязанности, общие для двух или более классов из данной совокупности
2. Вынесите эти элементы в некоторый общий класс (если надо, создайте новый, но следите, чтобы уровней не оказалось слишком много).
3. Отметьте в модели, что более специализированные классы наследуют более общим, включив отношение обобщения, направленное от каждого потомка к его родителю.

Структурные отношения

Отношения зависимости и обобщения применяются при моделировании классов, которые находятся на разных уровнях абстракции или имеют различную значимость. Что касается отношения зависимости, один класс зависит от другого, но тот может ничего не "знать" о наличии первого. Когда речь идет об отношении обобщения, класс-потомок наследует своему родителю, но сам родитель о нем не осведомлен. Другими словами, отношения зависимости и обобщения являются односторонними.

Ассоциации предполагают участие равноправных классов. Если между двумя классами установлена ассоциация, то каждый из них каким-то образом зависит от другого, и навигацию можно осуществлять в обоих направлениях. В то время как зависимость - это отношение использования, а обобщение - отношение "является", ассоциации определяют структурный путь, обусловливающий взаимодействие объектов данных классов. По умолчанию ассоциации являются двунаправленными, но вы можете оставить только одно направление.

Моделирование структурных отношений производится следующим образом:

1. Определите ассоциацию для каждой пары классов, между объектами которых надо будет осуществлять навигацию. Это взгляд на ассоциации с точки зрения данных.

2. Если объекты одного класса должны будут взаимодействовать с объектами другого иначе, чем в качестве параметров операции, следует определить между этими классами ассоциацию. Это взгляд на ассоциации с точки зрения поведения.

3. Для каждой из определенных ассоциаций задайте кратность (особенно если она не равна *, то есть значению по умолчанию) и имена ролей (особенно если это помогает объяснить модель).

4. Если один из классов ассоциации структурно или организационно представляет собой целое в отношении классов на другом конце ассоциации, выглядящих как его части, пометьте такую ассоциацию как агрегирование.

Как узнать, когда объекты данного класса должны взаимодействовать с объектами другого класса? Для этого рекомендуется воспользоваться CRC-карточка-ми и анализом прецедентов (см. главу 16). Эти методы очень помогают при рассмотрении структурных и поведенческих вариантов функционирования системы. Если в результате обнаружится взаимодействие между классами, специфицируйте ассоциацию.

2. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ПРОВЕДЕНИЮ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

2.1 Практическое занятие №1 (2 часа).

Тема: «Общие принципы проектирования информационных систем»

2.1.1 Задание для работы:

1. Изучить объект проектирования.
2. Изучить процесс проектирования.
3. Понять формальную логику.
4. Изучить процесс образования понятия.
5. Изучить диалектическую логику.
6. Изучить содержательно-генетическую логику.

2.1.2 Краткое описание проводимого занятия:

Рассматриваются понятия проектирование, объекта, предмета проектирования. Стадии и этапы проектирования. Понятие формальной логики и ее отличие от неформальной. Процесс образования понятия. Понятие и предмет диалектической логики. Положения концепции содержательно-генетической логики.

2.1.3 Результаты и выводы:

Различные виды проектирования ориентированы на создание и преобразование разных объектов и предметов. Объект и предмет проектирования соотносятся между собой как общее и частное.

Комплекс проектных работ включает в себя теоретические и экспериментальные исследования, расчеты, конструирование.

Проектирование разделяется на этапы.

Формальная логика, в отличие от неформальной, организована как формальная система, обладающая высоким уровнем абстракции и чётко определёнными методами, правилами и законами.

Приемами образования понятия являются: абстрагирование, анализ, синтез, сравнение и обобщение. Все логические приемы образования понятий имеют важнейшее значение. Они связаны между собой, их невозможно представить один без другого. Часто применяются вместе или предшествуют один другому.

Диалектическая логика - это учение о познании, о философском постижении объективной истины. Она описывает процесс познания не реальной сферы действительности, а абстрактного объекта. Содержание диалектической логики показывает диалектический метод философского познания в чистом, наиболее общем, абстрактном виде. Формальная логика исследует отношения между мыслями, выраженные в стабильных, неизменных структурах.

Процесс мышления протекает согласно логическим законам независимо от того, знаем мы об их существовании или нет. Вследствие своей объективности логические законы, так же как и физические, нельзя нарушить, отменить или переиначить.

2.2 Практическое занятие №2 (2 часа).

Тема: «Виды систем проектирования АСОИ**»**

2.2.1 Задание для работы:

1. Изучить унифицированный процесс – управляемый вариантами использования.
2. Изучить унифицированный процесс - ориентирован на архитектуру.
3. Изучить унифицированный процесс - итеративный и инкрементный.

2.2.2 Краткое описание проводимого занятия:

Унифицированный процесс разработки программного обеспечения с точки зрения управляемости вариантами использования.

Унифицированный процесс с точки зрения ориентации на архитектуру. Итеративный и инкрементный унифицированный процесс.

2.2.3 Результаты и выводы:

Унифицированный процесс есть процесс разработки программного обеспечения. Процесс разработки программного обеспечения - это сумма различных видов деятельности, необходимых для преобразования требований пользователей в программную систему. Однако Унифицированный процесс - это больше, чем единичный процесс, это обобщенный каркас процесса, который может быть специализирован для широкого круга программных систем, различных областей применения, уровней компетенции и размеров проекта.

Унифицированный процесс компонентно-ориентирован. Это означает, что создаваемая программная система строится на основе программных компонентов, связанных хорошо определенными интерфейсами.

Для разработки чертежей программной системы Унифицированный процесс использует Унифицированный язык моделирования.

Архитектура - это представление всего проекта с выделением ключевых составляющих и затушевывание деталей. Архитектура вырастает из требований к результату, в том виде, как их понимает пользователь и другие заинтересованные лица.

Каждый продукт имеет функции и форму, причем одно без другого не существует. В нашем случае функции, как мы ранее отмечали, соответствуют вариантам использования, а форма – архитектуре. Согласно знаниям, заложенным в методологии (унифицированного процесса),сначала должны быть разработаны варианты использования, то есть функции, а потом для того чтобы обеспечить выполнение этих функций разрабатывается архитектура системы. С другой стороны архитектура должна обеспечить реализацию необходимых сейчас и в будущем функций, то есть вариантов использования. Реально архитектура и варианты использования разрабатываются параллельно.

Таким образом, архитектор придает системе форму и архитектор, проектируя форму, должен заложить такие решения, которые бы позволили системе развиваться не только в момент начальной разработки, но и в будущих поколениях системы.

На каждой итерации разработчики определяют и описывают уместные варианты использования, создают проект, использующий выбранную архитектуру в качестве направляющей, реализуют проект в компоненты и проверяют соответствие компонентов вариантам использования. Если итерация достигла своей цели, процесс разработки переходит на следующую итерацию. Если итерация не выполнила своей задачи, разработчики должны пересмотреть свои решения и попробовать другой подход.

2.3 Практическое занятие №3 (2 часа).

Тема: «Типы диаграмм в языке UML**»**

2.3.1 Задание для работы:

1. Изучить классификацию диаграмм, принятые обозначения.
2. Изучить изображение ассоциаций на диаграммах классов.
3. Иерархии классов.
4. CRC-карточки.
5. Диаграмма классов.
6. Статические (static) и динамические классы.
7. Диаграммы объектов.
8. Диаграммы прецедентов.
9. Диаграмма состояний (конечных автоматов).
10. Диаграммы последовательностей.

2.3.2 Краткое описание проводимого занятия:

Диаграмма UML. Структурные и поведенческие диаграммы. Ассоциации на диаграмме классов. Порядок наследования отношений в иерархии классов. Назначение CRC-карточек. Цель построения диаграммы классов. Описание статических и динамических классов. Определение диаграммы объектов, назначение и представление диаграммы. Понятие и цели создания диаграммы прецедентов. Цель создания диаграммы состояний. Понятие диаграммы последовательности.

2.3.3 Результаты и выводы:

Рассматривая диаграмму UML, необходимо помнить, что основной принцип UML заключается в том, что любая информация на конкретной диаграмме может быть подавлена. Это подавление может носить глобальный характер – скрыть все атрибуты – или локальный – не показывать какие-нибудь конкретные классы. Поэтому по диаграмме вы никогда не можете судить о чем-нибудь по его отсутствию. Даже если метамодель UML имеет поведение по умолчанию, например для атрибутов, когда вы не видите эту информацию на диаграмме, это может быть обусловлено либо поведением по умолчанию, либо тем, что она просто подавлена.

Диаграммы классов используются при моделировании ПС наиболее часто. Они являются одной из форм статического описания системы с точки зрения ее проектирования, показывая ее структуру. Диаграмма классов не отображает динамическое поведение объектов изображенных на ней классов. На диаграммах классов показываются классы, интерфейсы и отношения между ними.

Диаграммы классов используются при моделировании ПС наиболее часто. Они являются одной из форм статического описания системы с точки зрения ее проектирования, показывая ее структуру. Диаграмма классов не отображает динамическое поведение объектов изображенных на ней классов. На диаграммах классов показываются классы, интерфейсы и отношения между ними.

Самым распространенным видом отношения зависимости является соединение между классами, когда один класс использует другой в качестве параметра операции.

Моделирование отношений наследования осуществляется в таком порядке:

1. Найти атрибуты, операции и обязанности, общие для двух или более классов из данной совокупности
2. Вынести эти элементы в некоторый общий класс (если надо, создайте новый, но следите, чтобы уровней не оказалось слишком много).
3. Отметить в модели, что более специализированные классы наследуют более общим, включив отношение обобщения, направленное от каждого потомка к его родителю.

CRC-карточки - эффективный способ анализа сценариев. Карточки можно раскладывать так, чтобы представить формы сотрудничества объектов. С точки зрения динамики сценария, их расположение может показать поток сообщений между объектами, с точки зрения статики они представляют иерархии классов.

Диаграмма классов описывает типы объектов системы и различного рода статические отношения, которые существуют между ними. На диаграммах классов отображаются также свойства классов, операции классов и ограничения, которые накладываются на связи между объектами.

Структурная классификация описывает системные сущности и их отношения между собой. В число классификаторов, имеющихся в моделях UML, входят классы, варианты использования, компоненты и узлы. Классификаторы являются базой, на которой строится динамическое поведение системы.

Динамическое поведение описывает поведение системы во времени. Поведение можно определить как ряд изменений в мгновенных снимках системы, полученных со статической точки зрения.

Диаграмма объектов показывает взаимосвязи экземпляров некоторых классов. Она используется для пояснения некоторых частей системы со сложными отношениями между объектами, особенно в случае использования рекурсивных отношений.

Диаграммы прецедентов относятся к той группе диаграмм, которые представляют динамические или поведенческие аспекты системы. Это отличное средство для достижения взаимопонимания между разработчиками, экспертами и конечными пользователями продукта. Такие диаграммы очень просты для понимания и могут восприниматься и, что немаловажно, обсуждаться людьми, не являющимися специалистами в области разработки ПО.

Главное назначение диаграммы состояний - описать возможные последовательности состояний и переходов, которые в совокупности характеризуют поведение моделируемой системы в течение всего ее жизненного цикла. Диаграмма состояний представляет динамическое поведение сущностей, на основе спецификации их реакции на восприятие некоторых конкретных событий. Системы, которые реагируют на внешние действия от других систем или от пользователей, иногда называют реактивными. Если такие действия инициируются в произвольные случайные моменты времени, то говорят об асинхронном поведении модели.

Диаграммы последовательностей - это отличное средство документирования поведения системы, детализации логики сценариев использования; но есть еще один способ - использовать диаграммы взаимодействия.