Шпаргалка по ООАиП.

\\Author: Maxim Pestun
\\Date: 18.01.2011

Лекция 1.

Системный подход – это методология исследования объекта любой природы как системы.
Система – это совокупность взаимосвязанных частей, работающих совместно для достижения некоторого результата. Определяющий признак системы – поведение системы в целом не сводимо к совокупности поведения частей системы.
Программное обеспечение – это система, включающая в себя: компьютерные программы; документацию; данные, необходимые для корректной работы программ.
Проектирование ПО – это процесс создания спецификаций ПО на основе исходных требований к нему.
Проект ПО – совокупность спецификаций ПО (включающих модели и проектную документацию), обеспечивающих создание ПО в конкретной программно-технической среде.
Инженерия ПО (software engineering) – совокупность инженерных методов и средств создания ПО.
Жизненный цикл ПО (software lifecycle) – это период времени, который начинается с момента принятия решения о необходимости создания ПО и заканчивается в момент его полного изъятия из эксплуатации.
Процесс ЖЦ – набор взаимосвязанных действий, преобразующих некоторые входные данные и ресурсы в выходные.
Каждый процесс ЖЦ характеризуется задачами, методами их решения, действующими лицами, результатами. Процессы ЖЦ протекают параллельно. Каждый процесс разделен на набор действий, каждое действие – на набор задач. Каждый процесс, действие или задача инициируется и выполняется по мере необходимости, причем не существует заранее определенных последовательностей выполнения.

- ***основные*** (приобретение, поставка, разработка, эксплуатация, сопровождение);
- ***вспомогательные*** (документирование, управление конфигурацией, обеспечение качества, верификация, аттестация, совместная оценка, аудит, разрешение проблем);
- ***организационные*** (управление, создание инфраструктуры, усовершенствование, обучение).

Процессы ЖЦ согласно стандарту ISO/IEC 12207: 1995:

- Процесс приобретения;
- Процесс поставки;
- Процесс разработки;
- Процесс эксплуатации;
- Процесс сопровождения;
- Процесс документирования;
- Процесс управления конфигурацией;
- Процесс обеспечения качества;
- Процесс верификации;
- Процесс аттестации;
- Процесс совместной оценки;
- Процесс аудита;
- Процесс разрешения проблем;
- Процесс управления;
- Процесс создания инфраструктуры;
- Процесс усовершенствования;
- Процесс обучения.

Модель ЖЦ ПО – это структура, определяющая последовательность выполнения и взаимосвязи процессов, действий и задач на протяжении всего ЖЦ.

Стадия ЖЦ – это часть ЖЦ ограниченная временными рамками, по завершении которой достигается определенный важный результат в соответствии с требованиями для данной стадии ЖЦ.
Модели ЖЦ:

- каскадная (водопадная);
- эволюционная;
- основанная на формальных преобразованиях;
- итерационные (пошаговая и спиральная).

Лекция 2.

Декомпозиция является главным способом преодоления сложности разработки ПО.
Принципы декомпозиции:

- количество связей между подсистемами должно быть минимальным (**«низкая связанность»** или **«слабое зацепление»** – **Low Coupling**);
- степень взаимодействия внутри каждой подсистемы должна быть максимальной (**«сильная связность»** или **«высокая прочность»** – **High Cohesion**).

При разбиении системы на подсистемы необходимо выполнить следующие **требования:**

- каждая подсистема должна ***инкапсулировать*** свое содержимое (скрывать его от других подсистем);
- каждая подсистема должна иметь четко ***определенный интерфейс*** с другими подсистемами, устанавливающий стандартные ограничения на взаимодействие.

Два основных подхода к декомпозиции систем:

- **функционально-модульный**, основанный на функциональной декомпозиции, при которой структура системы описывается в терминах иерархии ее функций и иерархии структур данных;
- **объектно-ориентированный**, использующий объектную декомпозицию, при которой структура системы описывается в терминах объектов и связей между ними, а поведение системы описывается в терминах обмена сообщениями между объектами.

Архитектура ПО – набор ключевых правил, определяющих организацию системы:

- совокупность структурных элементов системы и связей между ними;
- поведение элементов системы в процессе их взаимодействия;
- иерархия подсистем, объединяющих структурные элементы;
- архитектурный стиль (типовой способ организации системы).

Модель “4+1”:

- ***представление функциональных возможностей ПО*** (**представление вариантов использования**, содержащее сценарии взаимодействия системы с внешней средой и роли, которые играют пользователи ПО и внешние системы);
- ***представление логической организации ПО*** (**логическое представление**, элементами которого являются пакеты, подсистемы, классы, связи между ними);
- ***представление физической структуры программных компонент, входящих в состав ПО*** (**представление реализации**, элементами которого являются компоненты, связи между ними);
- ***представление структуры потоков управления и аспектов параллельной работы ПО*** (**представление процессов**, элементы которого: потоки управления, нити, параллелизм, синхронизация);
- ***описание физического размещения компонент ПО по узлам вычислительной системы*** (**представление размещения**, элементы которого: узлы вычислительной системы, устройства, линии связи, задачи).

Архитектурное представление – это модель системы с определенной точки зрения, в которой отражены лишь существенные аспекты и опущено все, что несущественно при данном взгляде на систему.
Модель ПО – это формализованное описание системы ПО на определенном уровне абстракции.
Архитектурно значимый элемент – это элемент, значительно влияющий на структуру системы, её функциональность, производительность, надежность, защищенность, возможность развития. ***Подсистемы***, их ***интерфейсы***, ***процессы*** и ***потоки управления*** являются архитектурно значимыми элементами.
Основными **принципами построения ООП** являются:

- абстрагирование;

- инкапсуляция;
- модульность;
- иерархия.

Дополнительные принципы:

- типизация;
- параллелизм;
- устойчивость (persistence).

Элементы объектной модели: объект; класс; атрибут; операция; полиморфизм; наследование; компонент; пакет; подсистема; связь.

Объект – осязаемая сущность (tangible entity) – предмет или явление (процесс), имеющие четко выраженные границы, индивидуальность и поведение.

Атрибут – поименованное свойство класса, определяющее диапазон допустимых значений, которые могут принимать экземпляры данного свойства. Атрибуты могут быть: public (общий, открытый); private (закрытый, секретный); protected (защищенный).

Операция – это услуга, которую можно запросить у любого объекта данного класса.

Виды операций:

- Операции реализации (implementor operations) – реализуют требуемую функциональность.
- Операции управления (manager operations) – управляют созданием и уничтожением объектов (конструкторы и деструкторы).
- Операции доступа (access operations) – так называемые, get-теры, set-теры – дают доступ к закрытым атрибутам.
- Вспомогательные операции (helper operations) – непубличные операции, служат для реализации операций других видов.

Компонент – это относительно независимая и замещаемая часть системы, выполняющая четко определенную функцию в контексте заданной архитектуры.

ООП – способ создания программных компонентов, базирующихся на объектах.

Пакет – это общий механизм для организации элементов в группы.

Подсистема – это комбинация пакета (может включать другие элементы модели) и класса (обладает поведением). Подсистема реализует один или более интерфейсов, определяющих ее поведение. Она используется для представления компонента в процессе проектирования.

Лекция 3.

Унифицированный язык моделирования UML (Unified Modeling Language) – это язык для определения, представления, проектирования и документирования программных систем, организационно-экономических систем, технических систем и других систем различной природы. UML содержит стандартный набор диаграмм и нотаций самых разнообразных видов.

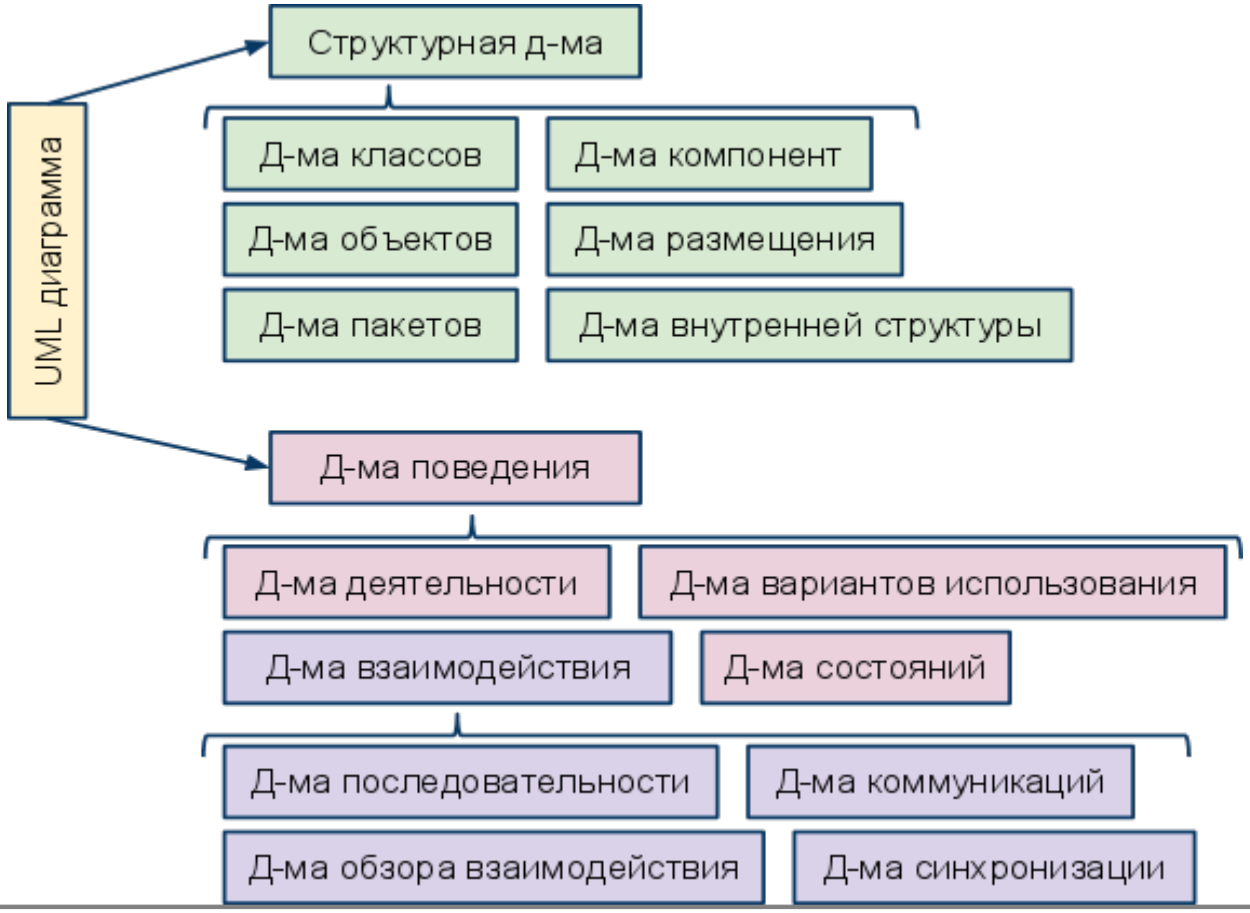


Диаграмма вариантов использования.

В диаграммах вариантов использования может присутствовать несколько типов связей:

- ***связь коммуникации*** (*линия со стрелкой*, обозначающая связь между вариантом использования и действующим лицом, по сути такая связь – путь для передачи запросов или данных);
- ***связь включения*** (*пунктирная линия со стрелкой*, обозначающая включение многократно используемой функциональности, представленной в виде отдельного варианта использования);
- ***связи расширения*** (*пунктирная линия со стрелкой*, соединяющая вариант – особый случай – с базовым вариантом использования);
- ***связь обобщения*** (*сплошная линия с треугольным концом*, используемая в иерархиях наследования действующих лиц или вариантов использования).

Диаграмма деятельности.

Основным элементом диаграмм деятельности является узел действия. Каждый такой узел представляет собой элементарную единицу работы (это может быть решение некоторой задачи, которую необходимо выполнить вручную или автоматизированным способом, или выполнение метода класса). Узел действия изображается в виде закругленного прямоугольника с текстовым описанием. Диаграмма деятельности может иметь **входной узел**, вообще говоря, не один, (в UML 1 должен быть ровно один), определяющий начало потока управления. **Финальный узел** необязателен. На диаграмме может быть несколько **финальных узлов**. На диаграмме могут присутствовать объекты и потоки объектов, в тех случаях, если объект используется или изменяется в одном из узлов действий. Поток объектов отмечается сплошной или пунктирной стрелкой от узла действия к объектному узлу или от объектного узла к узлу действия, использующего объект. **Ребра** (сплошные стрелки) между узлами действий показывают потоки управления или потоки объектов. **Узел разветвления** (узел принятия решения), а также **узел объединения** потоков изображается ромбом. Если необходимо показать, что две или более ветвей потока выполняются параллельно, используются «линейки синхронизации» – **узлы разделения** и **узлы слияния** (жирные горизонтальные линии).

Стереотип – это новый тип элемента модели, который определяется на основе уже существующего элемента. Стереотипы расширяют нотацию модели, могут применяться к любым элементам модели и представляются в виде текстовой метки или пиктограммы.
Теги (именованные значения) – специальные термины, используемые спецификации ограничений и свойств, такие как disjoint, complete, incomplete и др., могут сопровождаться указанием значения свойства, например, author=Вася или location=server.
Примечание – элемент диаграммы для комментария или другой текстовой информации. Примечание может содержать дополнительные сведения об элементах модели (с ними его соединяет пунктирная линия).
Ограничение – это семантическое ограничение, имеющее вид текстового выражения на естественном или формальном языке (OCL – Object Constraint Language), которое невозможно выразить с помощью нотации UML. Он может быть использован для решения следующих задач:

- описание инвариантов классов и типов в модели классов;
- описание пред- и постусловий в операциях и методах;
- описание ограничивающих условий элементов модели;
- навигация по структуре модели;
- спецификация ограничений на операции.

Лекция 4.

Классификация ограничений:

- ***Инвариант класса*** – условие, которое всегда справедливо для всех экземпляров класса (ключевое слово **inv:**).
- ***Предусловие операции*** – условие, которое должно быть истинно перед выполнением операции (ключевое слово **pre:**).
- ***Постусловие операции*** – условие, истинное всегда после выполнения операции (ключевое слово **post:**).
- ***Тело запроса*** – описание результата операции-запроса, не модифицирующей объекты (ключевое слово **body:**)
- ***Начальное значение атрибута или соединения*** (ключевое слово **init:**)
- ***Правило вывода***, описывающее производные атрибуты, связи или классы (ключевое слово **derive:**).
- **Определение** (ключевое слово **def:**).

Характеристики OCL:

- текстовый (невизуальный) язык описания ограничений;
- формальный язык, часть стандарта UML;
- язык со строгой типизацией;
- декларативный язык (для ограничений не определяется конкретная процедура их проверки);
- платформо-независимый.

Синтаксис OCL-выражения:

- <OCL-выражение> ::= <указание контекста> [(inv | pre | post | body | init | derive | def) : <тело выражения>]
- <указание контекста> ::= context <имя элемента модели>

В теле выражения используются:

- выражения простых типов (boolean, integer, string, real);
- элементы модели, для которой составлено ограничение;
- коллекции.

Примеры использования и описание операций:

- **context Airline inv: name.toLower = ‘klm’** – здесь ‘klm’ – строка, toLower – стандартная операция над строкой, дающая как результат строку в нижнем регистре. Смысл ограничения: у любого экземпляра класса Airline значение атрибута name, записанное строчным буквами совпадает со строкой ‘klm’.

- **context Passenger inv: age >= ((9.6 - 3.5)* 3.1).floor implies mature = true** – здесь floor – «округление вниз». Смысл ограничения: у любого экземпляра класса Passenger значения атрибутов age и mature таковы, что если age > 18, то mature = true.
- **context Flight inv: self.maxNrPassengers <= 1000** – в любом рейсе максимальное количество пассажиров не превышает 1000 (использован атрибут объекта – экземпляра класса Flight).
- **context Flight:maxNrPassengers:Integer init: 1000** – в любом рейсе максимальное количество пассажиров по умолчанию = 1000.
- **context Passenger inv: self.age >= Passenger::minAge** – у любого пассажира возраст больше минимального (использован атрибут age экземпляра класса Passenger и атрибут того же класса minAge).
- **context Airport inv: self.arrivingFlights -> collect(airLine) -> nonEmpty** – для любого аэропорта множество авиакомпаний, выполняющих прибывающие рейсы, не пусто.
- **context Airport inv: self.departingFlights->select(duration<4)->notEmpty** – для любого аэропорта есть хоть один отправляющийся рейс длительностью менее 4 часов.
- **context Airport inv: self.departingFlights->forAll(maxNrPassengers < 1000)** – для любого аэропорта справедливо, что у любого отправляющегося рейса максимальное количество пассажиров < 1000.

Некоторые дополнительные операции над коллекциями:

- exists: истина, если хотя бы для одного элемента коллекции истинно;
- isEmpty: истина, если коллекция пуста;
- notEmpty: истина, если коллекция непуста;
- size: количество элементов коллекции;
- sum: сумма элементов коллекции чисел;
- count(<элемент>): количество вхождений элемента;
- includes(<элемент>): истина, если <элемент> входит в коллекцию;
- excludes(<элемент>): истина, если <элемент> отсутствует в коллекции;
- includesAll(<коллекция>): истина, если все элементы параметра <коллекция> входят в коллекцию.

Лекция 5.

Бизнес-процесс (производственный процесс) определяется как логически завершенный набор взаимосвязанных и взаимодействующих видов деятельности, поддерживающий деятельность организации и реализующий ее политику, направленную на достижение поставленных целей. Бизнес-процесс использует определенные ресурсы (финансовые, материальные, человеческие, информационные). Выделяют следующие классы процессов:

- основные процессы (производство товаров и услуг, приносят доход, составляют основную деятельность компании);
- обеспечивающие процессы (обеспечение основных процессов финансами, кадрами, комплектующими, тех. обслуживанием, администрирование и юридическое обеспечение);
- процессы управления (планирование и контроль бизнес-процессов других видов).

Бизнес-модель – это формализованное описание бизнес-процессов предприятия, фиксирующее существующее положение дел (модель AS-IS «как есть») или устанавливающее новые усовершенствованные способы осуществления деятельности (модель AS-TO-BE «как будет»).

Модель бизнес-процессов (Business Use Case Model) – модель, описывающая бизнес-процессы организации в терминах ролей и их потребностей. Она представляет собой расширение модели вариантов использования UML за счет введения набора стереотипов **Business Actor** (деловое действующее лицо – стереотип действующего лица) и **Business Use Case** (бизнес-процесс – стереотип варианта использования). Из этой модели видно в каком контексте работает предприятие, но не видно как именно протекает его работа (это описывает модель бизнес-анализа).

Каждый **бизнес-процесс сопровождается спецификацией**, в которой содержится:

- наименование;
- краткое описание бизнес-процесса;
- цели и результаты;
- описание сценариев (основного и альтернативных);

- специальные требования (время и стоимость);
- расширения (исключительные ситуации);
- связи;
- диаграммы деятельности (моделирующие сценарии бизнес-процесса).

Модель бизнес-анализа – это объектная модель, элементами которой являются исполнитель (business worker) и бизнес-сущность (business entity).

Лекция 6.

Требование – это условие, которому должно удовлетворять программное обеспечение, или свойство, которым оно должно обладать, чтобы:

- удовлетворить потребность пользователя в решении некоторой задачи;
- удовлетворить требования контракта, спецификации или стандарта.

Методы выявления требований:

- собеседование (интервьюирование);
- анкетирование;
- проведение совещаний;
- сессии по выявлению требований (мозговой штурм);
- раскадровка (storyboard);
- создание и демонстрация работающих прототипов;
- ролевые игры.

Лекция 7.

При построении диаграмм взаимодействия возникают проблемы правильного распределения обязанностей между классами. Для их решения существует ряд образцов: Information Expert, Creator, Low Coupling, High Cohesion.

Образец “Information Expert”. Проблема: Нужно определить наиболее общий принцип распределения обязанностей между классами. Решение: Следует назначить обязанность информационному эксперту – классу, у которого имеется информация, требуемая для выполнения обязанности.

Образец "Creator". Проблема: Нужно определить, кто должен отвечать за создание нового экземпляра некоторого класса. Создание новых объектов в объектно-ориентированной системе является одним из стандартных видов деятельности. Следовательно, при назначении обязанностей, связанных с созданием объектов, полезно руководствоваться некоторым основным принципом. Решение: Следует назначить классу B обязанность создавать экземпляры класса A, если выполняется одно из следующих условий:

- класс B агрегирует, содержит или активно использует объекты класса A;
- класс B обладает данными инициализации, которые будут передаваться объектам класса A при их создании (т.е. класс B является информационным экспертом).

Класс B при этом определяется как создатель (creator) объектов класса A.

Если несколько классов удовлетворяют этим условиям, то предпочтительнее использовать в качестве создателя класс, агрегирующий или содержащий класс A.

Образец "Low Coupling" (низкая связанность). Проблема: Нужно распределить обязанности между классами таким образом, чтобы снизить взаимное влияние изменений в них и повысить возможность повторного использования. Решение: Следует распределить обязанности таким образом, чтобы обеспечить низкую связанность.

Образец "High Cohesion" (высокая функциональная прочность или сильное зацепление обязанностей). Проблема: Нужно распределить обязанности между классами таким образом, чтобы каждый класс не выполнял много разнородных функций или несвязанных между собой обязанностей. Такие классы создавать нежелательно, поскольку они приводят к возникновению таких же проблем, как у классов с сильной связанностью. Решение: Следует распределить обязанности таким образом, чтобы обеспечить высокую функциональную прочность.

Лекция 8.

Процесс – это ресурсоемкий поток управления, который может выполняться параллельно с другими потоками. Он выполняется в независимом адресном пространстве и в случае высокой сложности может разделяться на два или более потоков.
Поток (нить) – это облегченный поток управления, который может выполняться параллельно с другими потоками в рамках одного и того же процесса в общем адресном пространстве.

Лекция 9.

База данных – долговременное самодокументированное хранилище данных.
Структура реляционных данных – совокупность таблиц. В любой таблице фиксированное количество столбцов и произвольное – строк. Совокупность значений ячеек одной строки – запись.
Оператор SQL – предложение SQL для манипуляции данными (выборка, изменение, добавление, удаление).
Отображение классов:

- Каждый класс переводится в отдельную таблицу, атрибуты становятся столбцами таблицы. Операции на структуру таблицы не влияют. Они могут переводиться в хранимые процедуры, если мы ходим переложить часть работы по манипулированию данными на СУБД.
- Уникальный идентификатор устойчивого класса превращается в первичный ключ таблицы. Если имеется несколько альтернативных уникальных идентификаторов, выбирается наиболее используемый. Если в модели для устойчивого класса явно не указан идентификатор, то в таблицу добавляется столбец ID – первичный ключ.

Отображение бинарных ассоциаций:

- **«1 к 1му»** – возможны различные решения, лучше создать общую таблицу для 2х классов. Столбцы – совокупность атрибутов. Первичный ключ – любой ID (первого или второго класса). Достигается максимально возможная экономия: одна таблица представляет оба класса и ассоциацию между ними.
- **«1 к 0..1»** – внешний ключ добавляется к таблице необязательного класса.
- **«0..1 к 0..1»** – рекомендуется отдельная таблица для связи. Ее столбцы – внешние ключи для таблиц классов, связанных ассоциацией. Основной ключ – комбинация этих столбцов.
- **«* к *»** – для ассоциации заводится отдельная таблица. Ее столбцы – внешние ключи для таблиц классов, связанных ассоциацией. Основной ключ – комбинация этих столбцов.
- **«1 к 1..*»** – к таблице класса у полюса «1..*» добавляется столбец – внешний ключ для таблицы класса у полюса «один».
- **«1 к 0..*»** – как «1 к 1..*».
- **«0..1 к *»** – применяются те же решения, что и в «0..1 к 0..1».

Отображение классов связанных N-арной ассоциацией:

- Требуется таблица для хранения связи. Например, в случае тернарной (N=3) связи формируются четыре таблицы, по одной для каждого класса и одна для связи. Таблица связи будет иметь среди своих атрибутов ключи каждой из 3х других таблиц.

Отображение классов-ассоциаций:

- Атрибуты класса-ассоциации добавляются либо в создаваемую для связи таблицу, либо (если дополнительная таблица не требуется) в ту таблицу, куда добавляется внешний ключ.

Лекция 10.

Образец (паттерн) – это типовое проектное решение конкретной задачи проектирования, описанное специальным образом, чтобы облегчить его повторное применение.
Абстрактная фабрика (Abstract Factory)

- Классификация: образец порождения объектов.
- Назначение: предоставляет интерфейс для создания взаимосвязанных и взаимозависимых объектов, не определяя их конкретных классов.

- Мотивация: часто встает задача проектирования программной системы независимой от конкретной реализации GUI.

Фабричный метод (Factory method)

- Классификация: образец порождения объектов.
- Назначение: определяет интерфейс для создания объектов, но оставляет реализациям решение о том, какой объект какого именно класса создавать.
- Мотивация: нужно создать экземпляр одной из реализаций интерфейса, но заранее неизвестно какой. Например, в текстовом редакторе работающем с разными форматами при создании нового документа не известен его тип.

Адаптер (Adapter)

- Классификация: структурный образец.
- Назначение: преобразует один интерфейс в другой, обеспечивая совместимость.
- Мотивация: есть библиотечный класс, который можно было бы использовать, но он не поддерживает требуемый интерфейс.

Composite (Компоновщик)

- Классификация: структурный образец.
- Назначение: организует объекты в древовидные структуры, позволяет клиентам единообразно работать с составными и элементарными объектами.
- Мотивация: есть совокупность объектов-контейнеров, состоящих из элементарных объектов и других контейнеров.

Мост (Bridge)

- Классификация: структурный образец.
- Назначение: отделить абстракцию от реализации.
- Мотивация: наследование жестко привязывает реализацию к абстракции, поэтому лучше иметь иерархию наследования для интерфейсов и отдельно их реализации.

Фасад (Facade)

- Структурный паттерн, идея которого в том, чтобы предоставить унифицированный интерфейс к подсистеме (или пакету) в виде прокси-класса SubsystemProxy (в случае пакета – в виде фасада пакета).

Proxy (Заместитель)

- Классификация: структурный образец.
- Назначение: для объекта создается суррогат, контролирующий доступ.
- Мотивация: есть «тяжелый» класс, объекты которого разумно создавать по требованию – эта обязанность возлагается на легкие суррогаты.

Цепочка обязанностей (Chain of Responsibility)

- Классификация: образец поведения.
- Назначение: избежать привязки отправителя запроса к получателю, давая возможность передать запрос через многих (заранее неизвестно скольких) посредников.

Iterator (Итератор)

- Классификация: образец поведения.
- Назначение: дать последовательный доступ к набору однородных объектов, не раскрывая его внутреннего представления.
- Ситуация применимости: есть структура данных, для которой нужно реализовать один или несколько способов обхода – каждый осуществляется отдельным итератором.

Strategy (Стратегия)

- Классификация: образец поведения.
- Назначение: Определяет семейство алгоритмов, инкапсулирует каждый из них и делает их взаимозаменяемыми.
- Мотивация: есть несколько алгоритмов решения одной задачи, которые нежелательно «зашивать» в клиентский класс.

Decorator (Декоратор)

- Классификация: структурный образец.
- Назначение: добавление объекту новых обязанностей в динамике. Альтернатива подклассам.

- Мотивация: Например, хотим, чтобы библиотека GUI могла добавлять свойства (рамку) или новое поведение (прокрутку) к любому элементу GUI. Для этого «оборачиваем» элемент GUI в объект-декоратор. Декоратор имеет тот же интерфейс. Он переадресует запросы элементу, который в него «завернут».

Лекция 11.

Технология создания ПО (ТС ПО) – это упорядоченная совокупность взаимосвязанных технологических процессов в рамках ЖЦ ПО.
Технологический процесс – это совокупность взаимосвязанных технологических операций.
Технологическая операция – это основная единица работы, выполняемая определенной ролью, которая:

- подразумевает четко определенную ответственность роли;
- дает четко определенный результат (набор рабочих продуктов), базирующийся на определенных исходных данных (другом наборе рабочих продуктов);
- представляет собой единицу работы с жестко определенными границами, которые устанавливаются при планировании проекта.

Рабочий продукт – информационная или материальная сущность, которая создается, модифицируется или используется в некоторой технологической операции (модель, документ, код, тест и т.п.).
Роль – определение поведения и обязанностей отдельного лица или группы лиц в среде организации-разработчика ПО, осуществляющих деятельность в рамках некоторого технологического процесса и ответственных за определенные рабочие продукты.
Руководство – практическое руководство по выполнению одной или совокупности технологических операций. Руководства включают методические материалы, инструкции, нормативы, стандарты и критерии оценки качества рабочих продуктов.
Инструментальное средство (CASE-средство) – программное средство, обеспечивающее автоматизированную поддержку деятельности, выполняемой в рамках технологических операций.
Согласно нормативам, ТС ПО должна поддерживать полный **набор процессов ЖЦ**, к которым относятся:

- управление требованиями;
- анализ и проектирование ПО;
- разработка ПО;
- эксплуатация;
- сопровождение;
- документирование;
- управление конфигурацией и изменениями;
- тестирование;
- управление проектом.

RUP в значительной степени соответствует указанным выше требованиям. Ее основными **принципами являются**:

- Раннее определение рисков и выработка ответных мер.
- Выполнение требований заказчиков.
- Концентрация на работающем коде.
- Готовность к изменениям с самого начала проекта и управление ими.
- Сборка системы из компонентов.
- Визуальное моделирование.
- Обеспечение качества в течение всего ЖЦ.

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

- начальная стадия (inception);
- стадия разработки (elaboration);
- стадия конструирования (construction);
- стадия ввода в действие (transition).

Статический аспект.
Статический аспект RUP представлен четырьмя основными элементами:

- роли;
- виды работ;
- рабочие продукты;
- дисциплины (процессы).

Понятие **«роль» (role)** определяет поведение и ответственность личности или группы личностей, составляющих проектную команду. Одна личность может играть в проекте много различных ролей.
Видом работы конкретного исполнителя понимается элементарная работа – технологическая опреация, выполняемая им.
Дисциплина (discipline) соответствует понятию технологического процесса (или процесса ЖЦ) и представляет собой последовательность работ, приводящую к получению значимого результата.
В рамках RUP определены шесть **основных дисциплин**:

- построение бизнес-моделей;
- определение требований;
- анализ и проектирование;
- реализация;
- тестирование;
- развертывание;

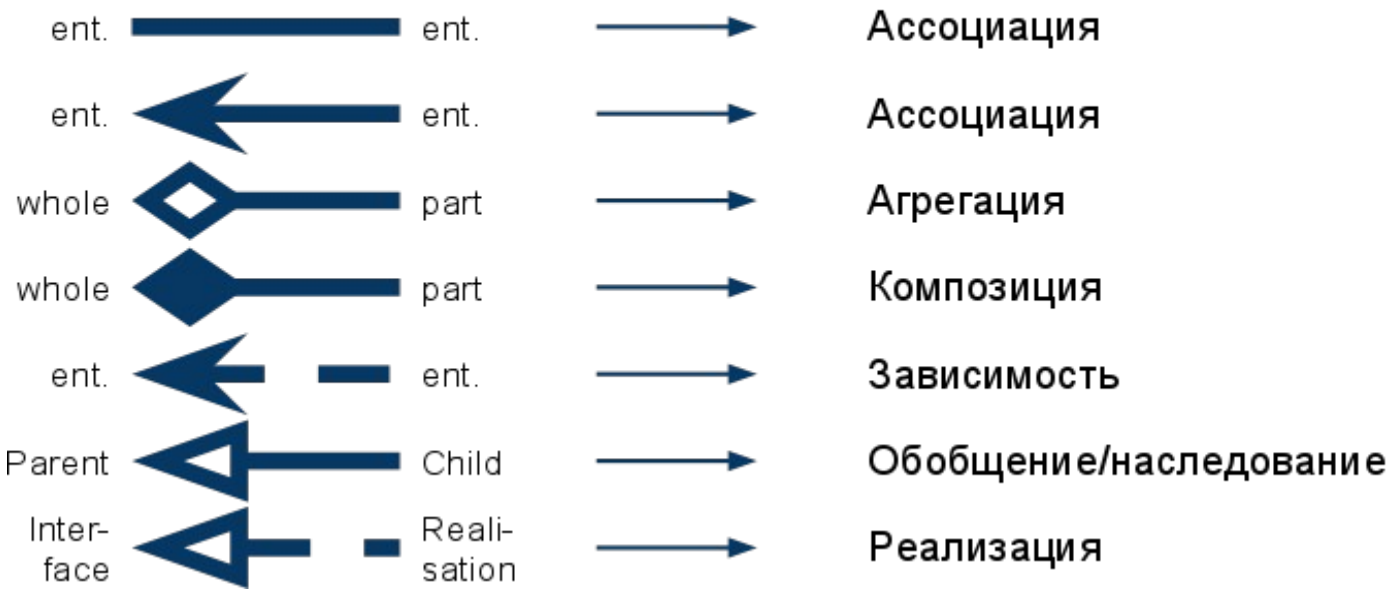
и три вспомогательных:

- управление конфигурацией и изменениями;
- управление проектом;
- создание инфраструктуры.

Виды тестирования:

- ***регрессионное*** (при котором новые сборки проверяются на тестах для предыдущих сборок, поскольку при интеграции новых компонент может быть нарушено правильное функционирование старых и системы в целом);
- ***инсталляционное*** (проверка возможности установки системы на вычислительной среде и правильность работы инсталлятора);
- ***конфигурационное*** (проверка работы системы в разных конфигурациях);
- ***отрицательное*** (проверка устойчивости системы на заведомо неверных данных, при неверных действиях пользователя – «защита от дурака» – при недостаточных ресурсах);
- ***нагрузочное*** (проверка нефункциональных свойств, например, производительности, пропускной способности).

Соединения


ent.
ent.
Ассоциация
ent.
ent.
Ассоциация
whole
part
Агрегация
whole
part
Композиция
ent.
ent.
Зависимость
Parent
Child
Обобщение/наследование
Inter-
face
Reali-
sation
Реализация