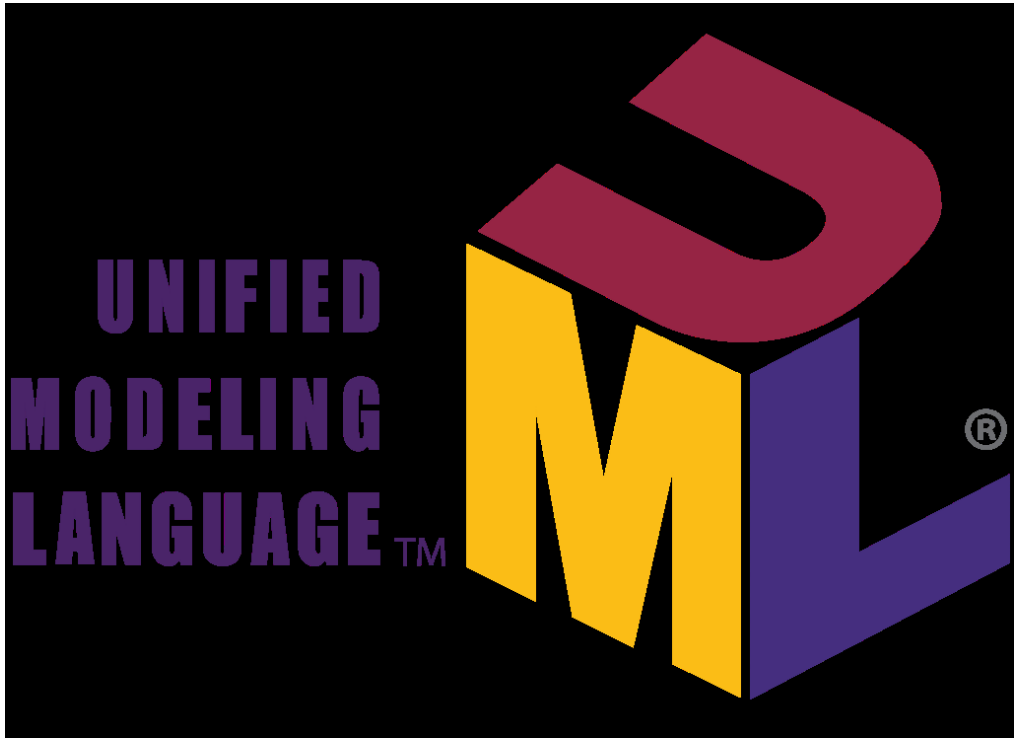
Лекция 5. Применение UML для проектирования сложных информационных систем



ПЛАН ЗАНЯТИЯ

- Краткий обзор истории UML.
- Синтаксис языка UML.
- Обзор основных диаграмм UML.
- Особенности разработки некоторых видов диаграмм.
- Примеры UML-диаграмм.
- Рекомендации по моделированию.
- Особенности использования некоторых диаграмм UML на разных этапах ЖЦ проектируемой системы.
- Обзор методологии RUP.

ОСНОВЫ UML

UML (**Unified Modeling Language**) представляет собой набор соглашений, которые предназначены для облегчения процесса моделирования и обмена информацией в проектной группе. Наличие стандартизированной нотации позволяет сократить время на усвоение информации, упрощает общение и взаимодействие, облегчает документирование.

UML представляет собой графическую нотацию, которая предназначена для моделирования и описания всех процессов, протекающих в процессе разработки.

Основу UML представляют диаграммы, которые различаются по типам и предназначены для моделирования различных аспектов разработки.

UML является языком широкого профиля, это — открытый стандарт, использующий графические обозначения для создания абстрактной модели системы, называемой UML-моделью. UML был создан для определения, визуализации, проектирования и документирования, в основном, программных систем. UML не является языком программирования, но на основании UML-моделей возможна генерация кода.

UML позволяет также разработчикам программного обеспечения достигнуть соглашения в графических обозначениях для представления общих понятий (таких как класс, компонент, обобщение (англ. generalization), агрегация (англ. aggregation) и поведение) и больше сконцентрироваться на проектировании и архитектуре.

Назначение языка UML: визуализация, специфицирование, конструирование и документирование объектно-ориентированных систем .

ИСТОРИЯ UML

Предпосылкой появления языка моделирования UML явилось бурное развитие во второй половине XX века объектно-ориентированных языков программирования (Simula 67, Smalltalk, Objective C, C++ и др).

В 1994 году **Гради Буч** и **Джеймс Рамбо**, работавшие в компании Rational Software, объединили свои усилия для создания нового языка объектно-ориентированного моделирования (OMT). За основу языка ими были взяты методы моделирования Object-Modeling Technique и Booch. OMT был ориентирован на анализ и разработку информационных систем, а Booch — на проектирование программных систем. В октябре 1995 года была выпущена предварительная версия 0.8 унифицированного метода (англ. Unified Method). Осенью 1995 года к компании Rational присоединился **Ивар Якобсон**, автор метода Object-Oriented Software Engineering — OOSE. OOSE обеспечивал превосходные возможности для спецификации бизнес-процессов и анализа требований при помощи сценариев использования. OOSE был также интегрирован в унифицированный метод.

На этом этапе основная роль в организации процесса разработки UML перешла к консорциуму **OMG** (Object Management Group, www.omg.org). Группа разработчиков в OMG, в которую также входили Буч, Рамбо и Якобсон («три амиго»), выпустила спецификации UML версий 0.9 и 0.91 в июне и октябре 1996 года.

Версия UML 2.4.1 принят в качестве международного стандарта ISO/IEC 19505-1, 19505-2.

Последняя версия UML 2.5.1 (по состоянию на февраль 2019 года, принята в декабре 2017 г.).

ИСТОРИЯ UML

Grady Booch
(Гради Буч)

Jim Rumbaugh
(Джим Рамбо)

Ivar Jacobson
(Ивар Якобсон)

ИСТОРИЯ UML. ПРОДОЛЖЕНИЕ

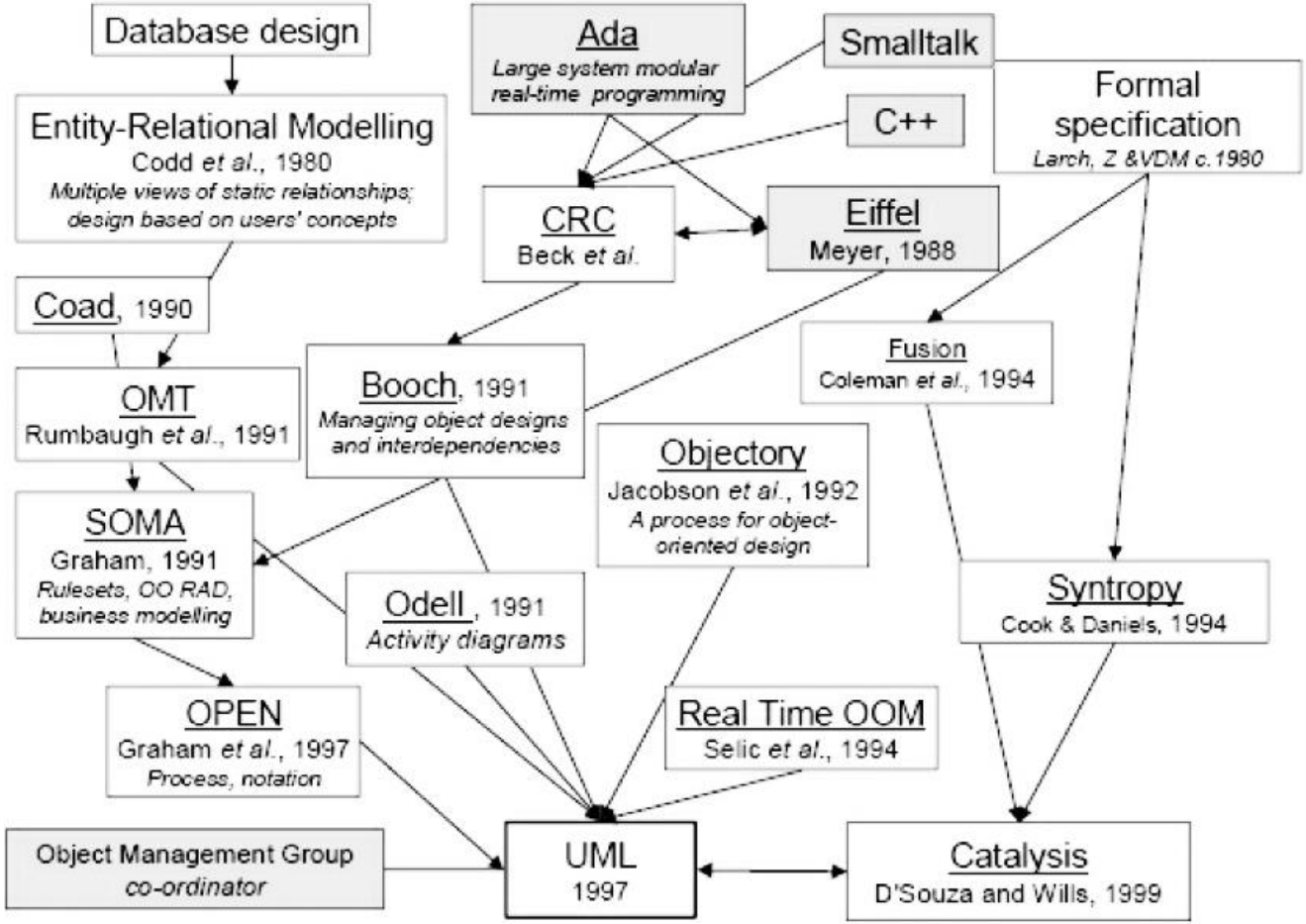
Главные цели разработчиков языка:

1. Моделировать системы целиком, от концепции до исполняемых компонентов, с помощью объектно-ориентированных методов;
2. Решить проблему масштабируемости, которая присуща сложным, критически важным системам;
3. Создать язык моделирования, который может использоваться не только людьми, но и компьютерами.

Задачи моделирования:

1. Визуализировать систему в ее текущем или желательном для нас состоянии;
2. Описать структуру или поведение системы;
3. Получить шаблон, позволяющий сконструировать систему;
4. Документировать принимаемые решения, используя полученные модели.

ИСТОРИЯ UML. ПРОДОЛЖЕНИЕ



ПРЕИМУЩЕСТВА UML

- UML позволяет описать систему с различных точек зрения и учитывать разные аспекты поведения системы;
- UML объектно-ориентирован, в результате чего методы описания результатов анализа и проектирования семантически близки к методам программирования на современных объектно-ориентированных языках;
- Диаграммы UML сравнительно просты для чтения после достаточно быстрого ознакомления с его синтаксисом;
- UML расширяет и позволяет вводить собственные текстовые и графические стереотипы, что способствует его применению не только в сфере программной инженерии;
- UML получил широкое распространение и динамично развивается.

ПРИНЦИПЫ МОДЕЛИРОВАНИЯ

Буч Г., Рамбо Д., Якобсон И. Язык UML. Руководство пользователя. 2-е изд.: Пер. с англ. Мухин Н. – М.: ДМК Пресс, 2006. – 496 с.

1. Выбор модели оказывает определяющее влияние на подход к решению проблемы и на то, как будет выглядеть это решение.
2. Каждая модель может быть представлена с различной степенью точности. Необходима возможность рассматривать систему на разных уровнях детализации в разное время.
3. Лучшие модели – те, что ближе к реальности. Поскольку модель всегда упрощает реальность, задача в том, чтобы это упрощение не повлекло за собой какие-то существенные потери.
4. Нельзя ограничиваться созданием только одной модели. Наилучший подход при разработке любой нетривиальной системы – использовать совокупность нескольких моделей, почти независимых друг от друга. В зависимости от природы системы некоторые модели могут быть важнее других.

ОБЗОР ЯЗЫКА МОДЕЛИРОВАНИЯ UML

UML – это язык для визуализации, специфицирования, конструирования и документирования артефактов программных систем. Язык представляет словарь и правила комбинирования входящих в него слов в целях коммуникации.

Язык моделирования – это язык, словарь и правила которого сосредоточены на концептуальном и физическом представлении системы. UML – стандартное средство представления «чертежей» программного обеспечения.

При создании программных систем необходим язык, средствами которого можно описать архитектуру системы с различных точек зрения, причем на протяжении всего жизненного цикла ее разработки.

С точки зрения визуализации UML дает разработчику набор графических символов. Каждый из этих символов имеет четко определенную семантику, что обеспечивает однозначную интерпретацию этих символов всеми разработчиками.

В контексте специфицирования UML обеспечивает построение точных, недвусмысленных и полных моделей. В частности, UML позволяет специфицировать все важные решения, касающиеся анализа, дизайна и реализации, принимаемые в процессе разработки и внедрения программных систем.

В контексте специфицирования UML поддерживает прямое и обратное проектирование, что обеспечивает возможность работы как с графическим, так и с текстовым представлениями; при этом обеспечивается согласованность между ними.

UML предназначен для документирования архитектуры системы и всех ее деталей.

КОНЦЕПТУАЛЬНАЯ МОДЕЛЬ UML

Словарь UML включает **три вида строительных блоков**:

1. Сущности.
2. Связи.
3. Диаграммы.

Сущности (things) – это абстракции, которые являются основными элементами модели, связи (relationships) соединяют их между собой, а диаграммы (diagrams) группируют представляющие интерес наборы сущностей.

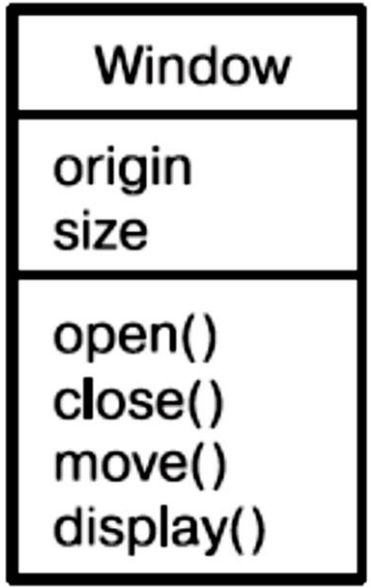
Четыре **вида сущностей UML**:

1. Структурные.
2. Поведенческие.
3. Группирующие.
4. Аннотирующие.

СТУКТУРНЫЕ СУЩНОСТИ UML. КЛАССЫ

Структурные сущности – «имена существительные» в моделях UML. Это в основном статические части модели, представляющие либо концептуальные, либо физические элементы. В совокупности структурные сущности называются классификаторами (classifiers).

Класс (class) – это описание набора объектов с одинаковыми атрибутами, операциями, связями и семантикой. Класс реализует один или несколько интерфейсов. Графически класс изображается в виде прямоугольника, обычно включающего имя, атрибуты и операции.

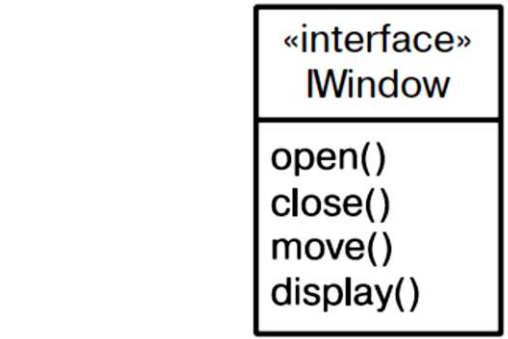


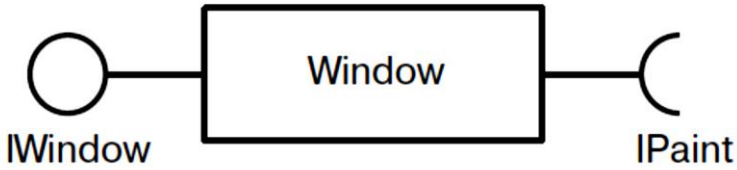
СТУКТУРНЫЕ СУЩНОСТИ UML. ИНТЕРФЕЙСЫ

Интерфейс (interface) – это набор операций, который специфицирует сервис (набор услуг) класса или компонента. Интерфейс описывает видимое извне поведение элемента. Он может представлять полное поведение класса либо только часть такого поведения. Определяет набор спецификаций операций (то есть их сигнатуру), но никогда не определяет детали их реализации.

Объявление интерфейса изображается как класс с ключевым словом «interface» над его именем.

Интерфейс, представляемый классом для внешнего мира, изображается в виде маленького круга, соединенного линией с рамкой класса. Интерфейс, запрашиваемый классом от некоторого другого класса, представлен маленьким полукругом, соединенным с рамкой класса линией.





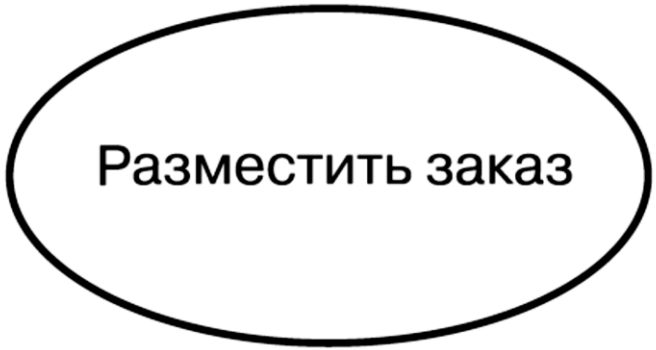
СТУКТУРНЫЕ СУЩНОСТИ UML. КООПЕРАЦИИ

Кооперация (collaboration) определяет взаимодействие и представляет собой совокупность ролей и других элементов, которые функционируют вместе, обеспечивая некоторое совместное поведение, представляющее нечто большее, чем сумма поведений отдельных элементов. Кооперации имеют как структурное, так и поведенческое измерения.

Конкретный класс или объект может участвовать в нескольких кооперациях. Последние представляют собой реализацию образцов (patterns) , составляющих систему. Кооперация изображается в виде эллипса, нарисованного пунктирной линией, включающего в себя ее имя.

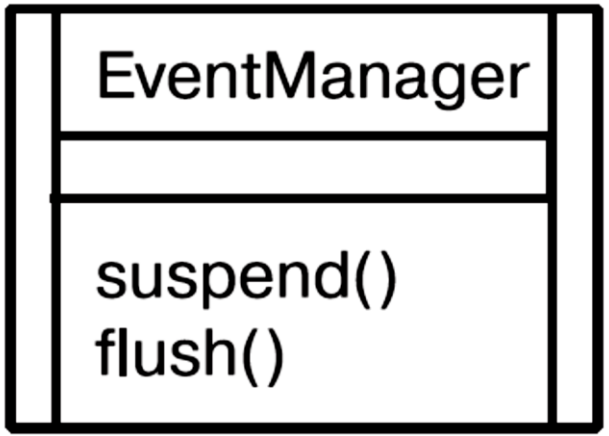
СТУКТУРНЫЕ СУЩНОСТИ UML. ВАРИАНТ ИСПОЛЬЗОВАНИЯ

Вариант использования (use case) – это описание последовательности действий, выполняемых системой и приносящих значимый результат конкретному действующему лицу (actor).
Варианты использования применяются для структурирования поведенческих сущностей модели. Реализуются посредством кооперации.
Графически вариант использования представлен эллипсом, нарисованным сплошной линией, обычно включающим в себя только имя.



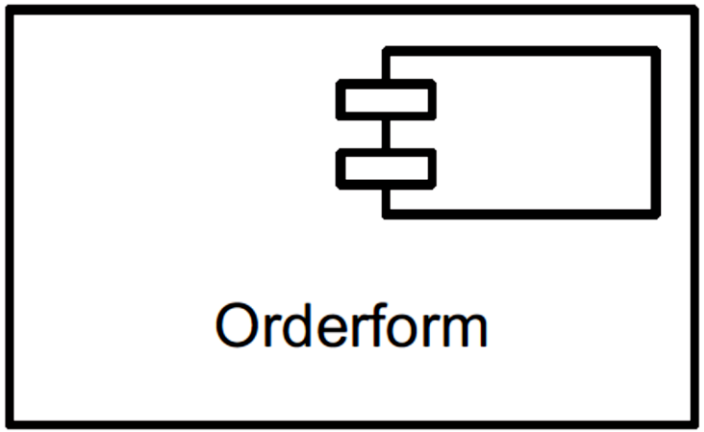
СТУКТУРНЫЕ СУЩНОСТИ UML. АКТИВНЫЕ КЛАССЫ

Активный класс – это класс, объекты которого являются владельцами одного или нескольких процессов или потоков (threads) и, таким образом, могут инициировать управляющие воздействия. Активный класс во всем подобен простому классу, за исключением того, что его объекты представляют собой элементы, поведение которых осуществляется параллельно с поведением других элементов. Изображается как класс с двойными боковыми линиями; обычно включает в себя имя, атрибуты и операции.



СТУКТУРНЫЕ СУЩНОСТИ UML. КОМПОНЕНТЫ

Компонент – это модульная часть системы, которая скрывает свою реализацию за набором внешних интерфейсов.
Компоненты системы, разделяющие общие интерфейсы, могут замещать друг друга, сохраняя при этом одинаковое логическое поведение.
Реализация компонента может быть выражена объединением частей и коннекторов; при этом части могут включать в себя более мелкие компоненты.
Графически компонент представлен как класс со специальной пиктограммой в правом верхнем углу.

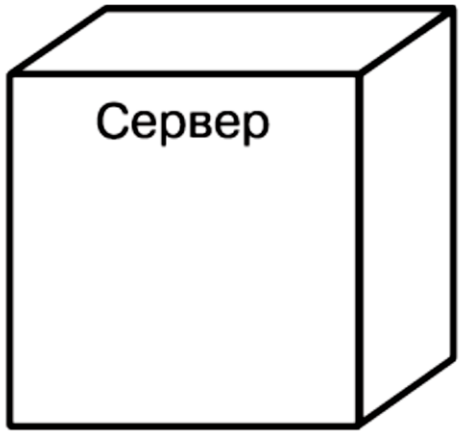


СТУКТУРНЫЕ СУЩНОСТИ UML. АРТЕФАКТЫ

Артефакт (artifact) – это физическая и замещаемая часть системы, несущая физическую информацию («биты»).
Возможны разные виды артефактов, таких как файлы исходного кода, исполняемые программы и скрипты.
Обычно артефакт представляет собой физический пакет с исходным или исполняемым кодом.
Изображается как прямоугольник, снабженный ключевым словом «artifact», расположенным над его именем.

СТУКТУРНЫЕ СУЩНОСТИ UML. УЗЛЫ

Узел (node) – это физический элемент, который существует во время исполнения и представляет вычислительный ресурс, обычно имеющий по меньшей мере некоторую память и часто – вычислительные возможности.
Набор компонентов может находиться на узле, а также мигрировать с одного узла на другой. Узел изображается в виде куба, обычно содержащего лишь его имя.



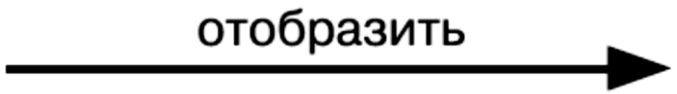
ПОВЕДЕНЧЕСКИЕ СУЩНОСТИ. ВЗАИМОДЕЙСТВИЯ

Поведенческие сущности – динамические части моделей UML. Это «глаголы» моделей, представляющие поведение во времени и пространстве. Всего существует три основных вида поведенческих сущностей.

Взаимодействие (interaction) – представляет собой поведение, которое заключается в обмене сообщениями между наборами объектов или ролей в определенном контексте для достижения некоторой цели. Поведение совокупности объектов или индивидуальная операция могут быть выражены взаимодействием.

Взаимодействие включает множество других элементов – таких как сообщения, действия (actions) и коннекторы (соединения между объектами).

Сообщение изображается в виде линии со стрелкой, почти всегда сопровождаемой именем операции.

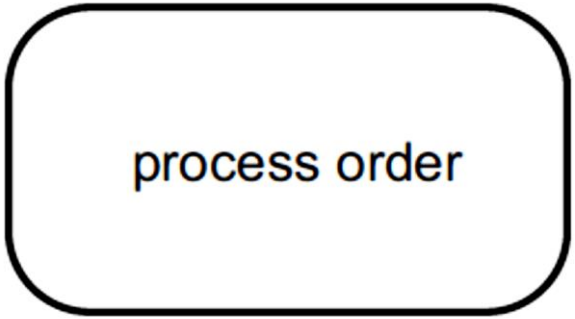


ПОВЕДЕНЧЕСКИЕ СУЩНОСТИ. АВТОМАТЫ

Автомат (state machine) – представляет собой поведение, характеризуемое последовательностью состояний объекта, в которых он оказывается на протяжении своего жизненного цикла в ответ на события, вместе с его реакцией на эти события.
Поведение индивидуального класса или кооперации классов может быть описано в терминах автомата.
Автомат включает в себя множество других элементов: состояния, переходы (из одного состояния в другое), события (сущности, которые инициируют переходы), а также действия (реакции на переходы).
Графически состояние представлено прямоугольником с закругленными углами, обычно с указанием имени и подсостояний, если таковые есть.

ПОВЕДЕНЧЕСКИЕ СУЩНОСТИ. ДЕЙСТВИЯ

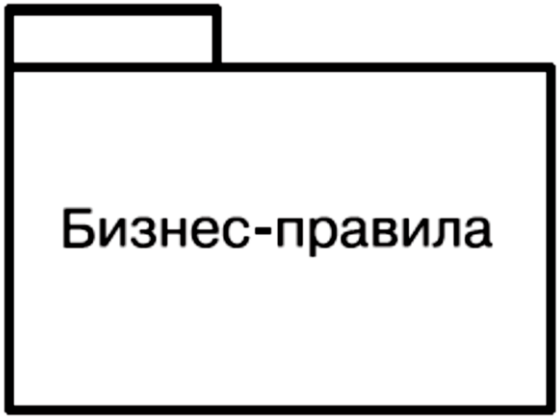
Деятельность (activity) – специфицирует последовательность шагов процесса вычислений.
Во взаимодействии внимание сосредоточено на наборе взаимодействующих объектов, в автомате – на жизненном цикле одного объекта; для деятельности же в центре внимания – последовательность шагов безотносительно к объектам, выполняющим каждый шаг.
Отдельный шаг деятельности называется **действием** (action). Изображается оно в виде прямоугольника с закругленными углами, включающего имя, которое отражает его назначение. Состояния и действия различаются по контекстам.
Семантически поведенческие сущности обычно связаны с различными структурными элементами – в первую очередь, классами, кооперациями и объектами.



ГРУППИРУЮЩИЕ СУЩНОСТИ. ПАКЕТЫ

Группирующие сущности – организационная часть моделей UML. Это «ящики», по которым можно разложить модель. Главная из группирующих сущностей – пакет.

Пакет (package) – это механизм общего назначения для организации проектных решений, который упорядочивает конструкции реализации. Структурные сущности, поведенческие сущности и даже другие группирующие сущности могут быть помещены в пакет. В отличие от компонентов (существующих только во время исполнения), пакеты полностью концептуальны, то есть существуют лишь на этапе разработки.



Пакет изображается в виде папки с закладкой, обычно только с указанием имени, но иногда и содержимого. Существуют и такие вариации группирующих сущностей, как каркасы (frameworks), модели, подсистемы (разновидность пакетов).

АННОТИРУЮЩИЕ СУЩНОСТИ

Аннотирующие сущности – это поясняющие части UML-моделей, комментарии, которые можно применить для описания, выделения и пояснения любого элемента модели.

Главная из аннотирующих сущностей – примечание (note). Это простой символ, служащий для описания ограничений и комментариев, относящихся к элементу либо набору элементов. Графически представлен прямоугольником с загнутым углом; внутри помещается текстовый или графический комментарий.

Существуют также разные вариации этого элемента, такие как требования, которые специфицируют некоторое желательное поведение с точки зрения, внешней по отношению к модели.

СВЯЗИ UML. ЗАВИСИМОСТИ

Существует **четыре типа связей в UML**:
1. Зависимость.
2. Ассоциация.
3. Обобщение.
4. Реализация.

Связи представляют собой базовые строительные блоки для описания отношений в UML, используемые для разработки хорошо согласованных моделей.

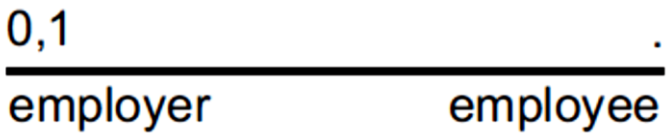
Зависимость (dependency) –представляет собой связь между двумя элементами модели, в которой изменение одного элемента (независимого) может привести к изменению семантики другого элемента (зависимого). Графически представлена пунктирной линией, иногда со стрелкой; может быть снабжена меткой.

СВЯЗИ UML. АССОЦИАЦИИ

Ассоциация (association) , – это структурная связь между классами, которая описывает набор связей, существующих между объектами – экземплярами классов.

Агрегация (aggregation) – особая разновидность ассоциации, представляющая структурную связь целого с его частями.

Изображается сплошной линией, иногда со стрелкой; иногда снабжена меткой и часто содержит другие пометки, такие как мощность и конечные имена.



СВЯЗИ UML. ОБОБЩЕНИЯ

Обобщение (generalization) – выражает специализацию или обобщение, в котором специализированный элемент (потомок) строится по спецификациям обобщенного элемента (родителя). Потомок разделяет структуру и поведение родителя.

Графически обобщение представлено в виде сплошной линии с пустой стрелкой, указывающей на родителя.

СВЯЗИ UML. РЕАЛИЗАЦИИ

Реализация (realization) – это семантическая связь между классификаторами, когда один из них специфицирует соглашение, которого второй обязан придерживаться.
Связи реализации используются между интерфейсами и классами или компонентами, которые реализуют эти интерфейсы, а также между вариантами использования и реализующими их кооперациями.
Связь реализации в графическом исполнении – гибрид связей обобщения и зависимости.

ДИАГРАММЫ UML

Диаграмма – это графическое представление набора элементов, чаще всего изображенного в виде связного графа вершин (сущностей) и путей (связей).

Диаграммы используются для визуализации системы с различных точек зрения, поэтому отдельная диаграмма – это проекция системы. Диаграмма представляет собой ограниченный взгляд на элементы, составляющие систему.

Диаграмма может включать в себя любую комбинацию сущностей и связей. На практике используется лишь небольшое число общих комбинаций, состоящих из наиболее часто применяемых представлений архитектуры программных систем, при этом допускается использование гибридных диаграмм.

UML включает 13 видов диаграмм:

1. Диаграмма классов.
2. Диаграмма объектов.
3. Диаграмма компонентов.
4. Диаграмма составной структуры.
5. Диаграмма вариантов использования.
6. Диаграмма последовательности.
7. Диаграмма коммуникации.
8. Диаграмма состояний.
9. Диаграмма деятельности.
10. Диаграмма размещения.
11. Диаграмма пакетов.
12. Временная диаграмма.
13. Диаграмма обзора взаимодействий.

ВИДЫ ДИАГРАММ UML

Все диаграммы UML можно условно разделить на **поведенческие** и **структурные.**
Поведенческие диаграммы отображают процессы, протекающие в моделируемой среде.
Структурные диаграммы отображают элементы, из которых состоит система. При этом одни и те же типы диаграмм могут использоваться как для моделирования бизнес-процессов, так и для непосредственного проектирования архитектуры.

СТРУКТУРА ДИАГРАММ UML 2.3 ПРЕДСТАВЛЕННАЯ НА ДИАГРАММЕ КЛАССОВ UML

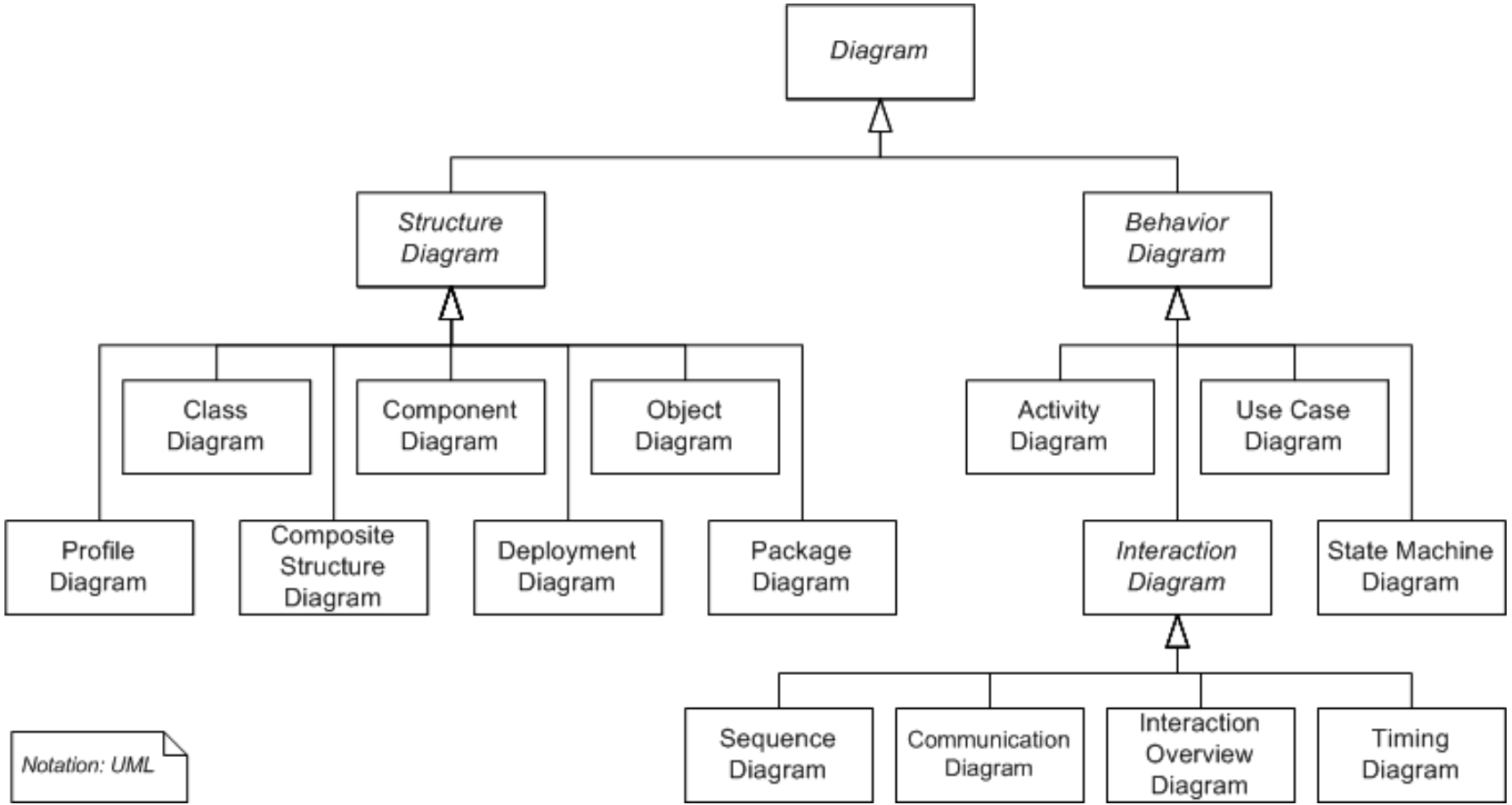


ДИАГРАММА ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ (USE-CASE DIAGRAM)

Диаграмма вариантов использования является отправной точкой в процессе моделирования.

Она **предназначена** для описания взаимодействия проектируемой системы с любыми внешними или внутренними объектами - пользователями, другими системами и т.п.

Цели создания диаграмм прецедентов:

- определение границы и контекста моделируемой предметной области на ранних этапах проектирования;
- формирование общих требований к поведению проектируемой системы;
- разработка концептуальной модели системы для ее последующей детализации;
- подготовка документации для взаимодействия с заказчиками и пользователями системы.

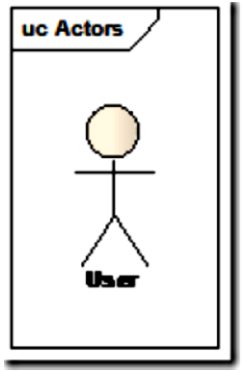
Основными понятиями при работе с диаграммой вариантов использования являются **Актор** (Actor) и **Вариант использования** (Use case).

Актор – это роль, которую выполняет пользователь или другая система, при взаимодействии с проектируемой системой.

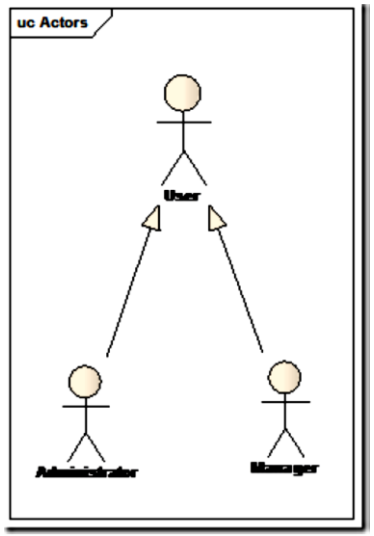
Вариант использования – это единица взаимодействия актора и системы. **Совокупность всех вариантов использования** полностью определяет поведение системы.

ДИАГРАММА ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ. АКТОРЫ

Проектирование диаграммы вариантов использования начинается с определения списка Акторов.
Каждый Актор обладает уникальным именем.

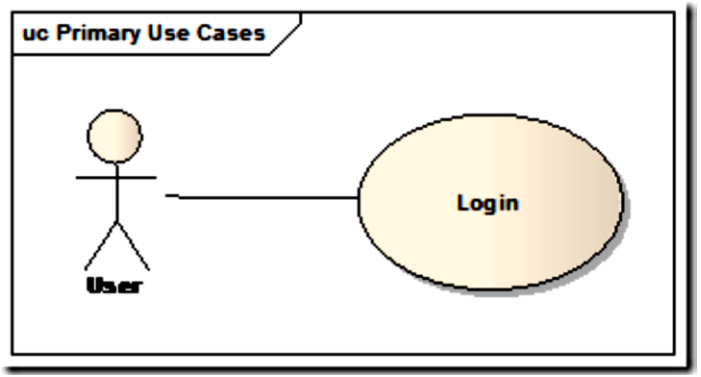


Друг с другом акторы могут быть связаны различного рода отношениями.
Например, акторы могут наследоваться друг от друга. В этом случае акторы-наследники наследуют характеристики базовых акторов.

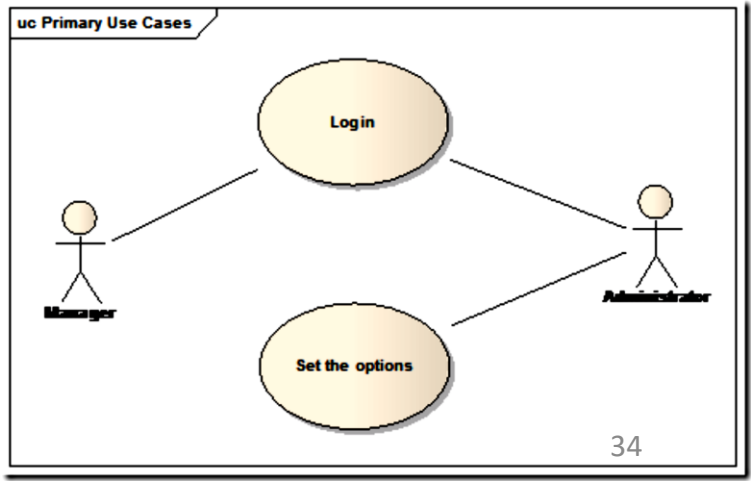


ВАРИАНТЫ ИСПОЛЬЗОВАНИЯ

Каждый **вариант использования** относится к каком-либо актору.
Такое отношение обозначает, что данный актор инициирует данный вариант использования.



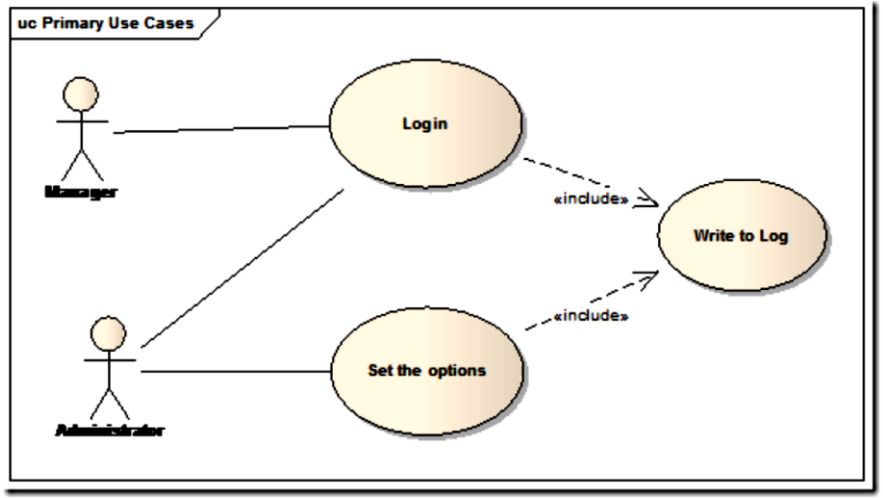
Один и тот же вариант использования может использоваться несколькими акторами.
Например вариант использования Login используется двумя акторами.



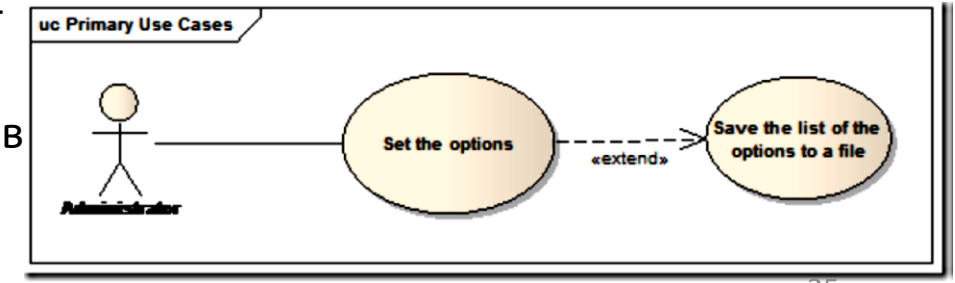
ОТНОШЕНИЯ МЕЖДУ ВАРИАНТАМИ ИСПОЛЬЗОВАНИЯ

Варианты использования также могут быть связаны друг с другом различными отношениями.

1. «**Включение**» одного варианта использования в другой. Означает, что один вариант использования инициируется в процессе другого.

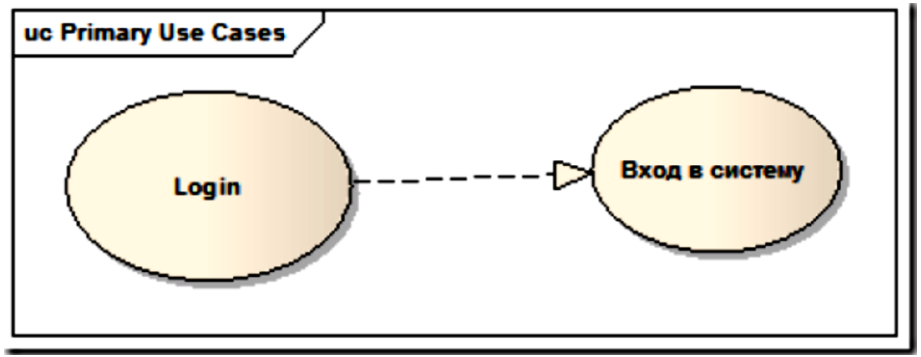


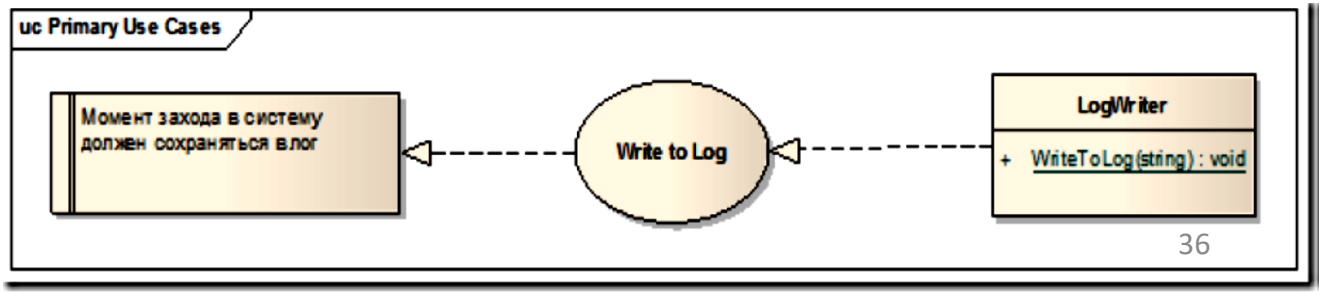
2. «**Расширение**». Означает, что один вариант использования является дополнением или уточнением другого варианта использования в случае наступления некоторых условий.



ОТНОШЕНИЯ МЕЖДУ ВАРИАНТАМИ ИСПОЛЬЗОВАНИЯ. ПРОДОЛЖЕНИЕ

3. «**Реализация**». Означает, что один вариант использования является реализацией другого варианта использования. Например, если один из них описан в терминах бизнес-процессов, а другой – в терминах проектируемой системы. Кроме того, варианты использования могут быть связаны отношением «Реализация» с требованиями к системе и с классами. При наличии таких связей есть возможность проследить в каких классах реализованы требования и какие классы могут быть затронуты при изменении требований или вариантов использования.

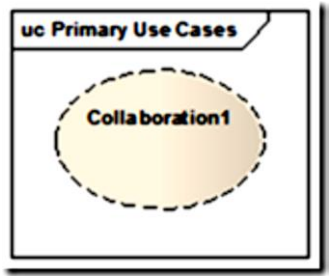


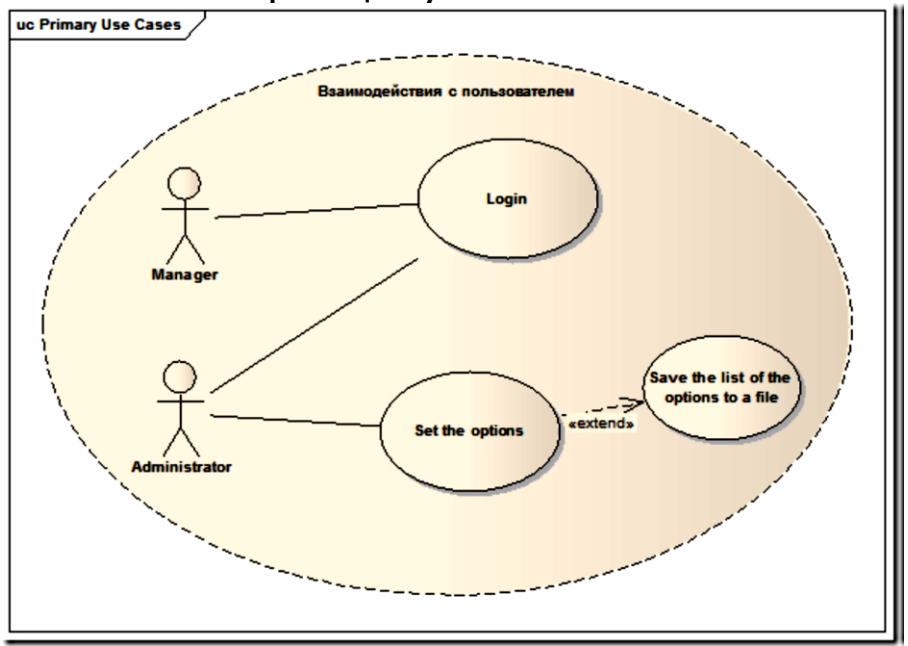


ДОПОЛНИТЕЛЬНЫЕ ЭЛЕМЕНТЫ ДИАГРАММЫ

Кроме Акторов и Вариантов использования на диаграмме также могут находиться следующие элементы:

1. «**Collaboration**» – элемент, предназначенный для визуальной группировки объектов – акторов и вариантов использования – по принципу их совместной работы.

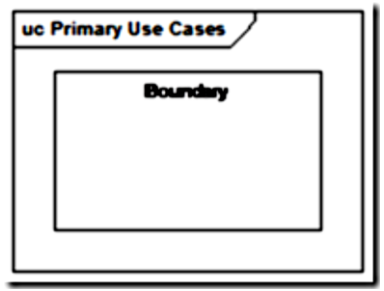


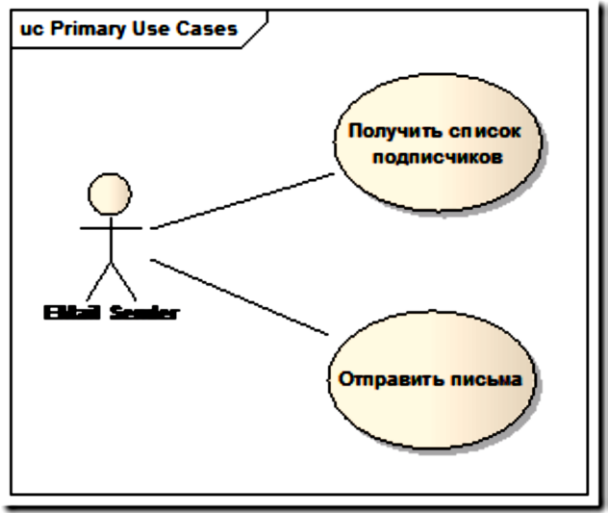


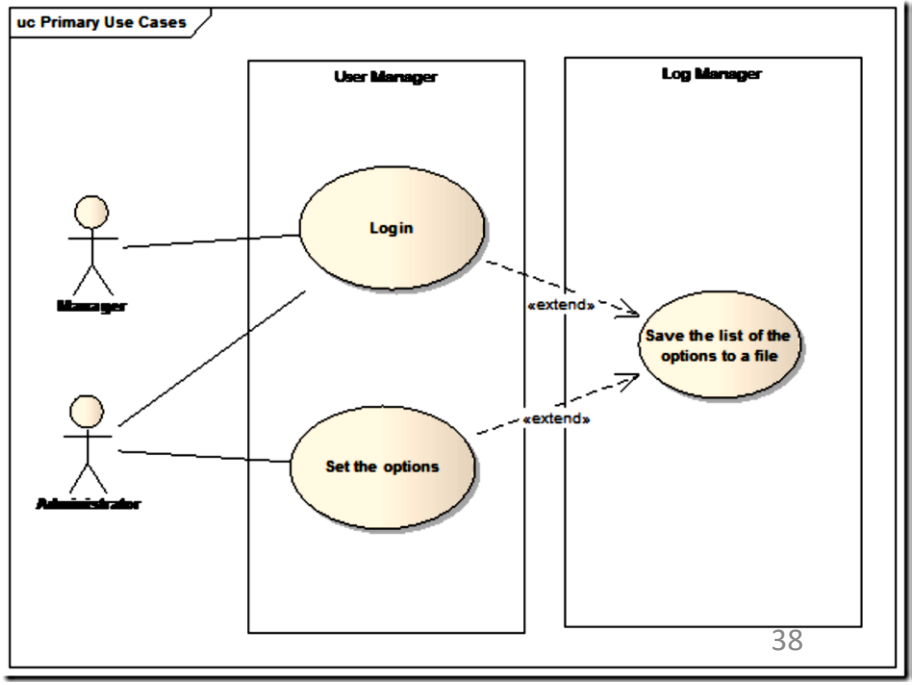
ДОПОЛНИТЕЛЬНЫЕ ЭЛЕМЕНТЫ ДИАГРАММЫ. ПРОДОЛЖЕНИЕ

2. «**Boundary**» - элемент, предназначенный для визуальной группировки объектов – акторов и вариантов использования – по принципу их распределения на подсистемы или компоненты.

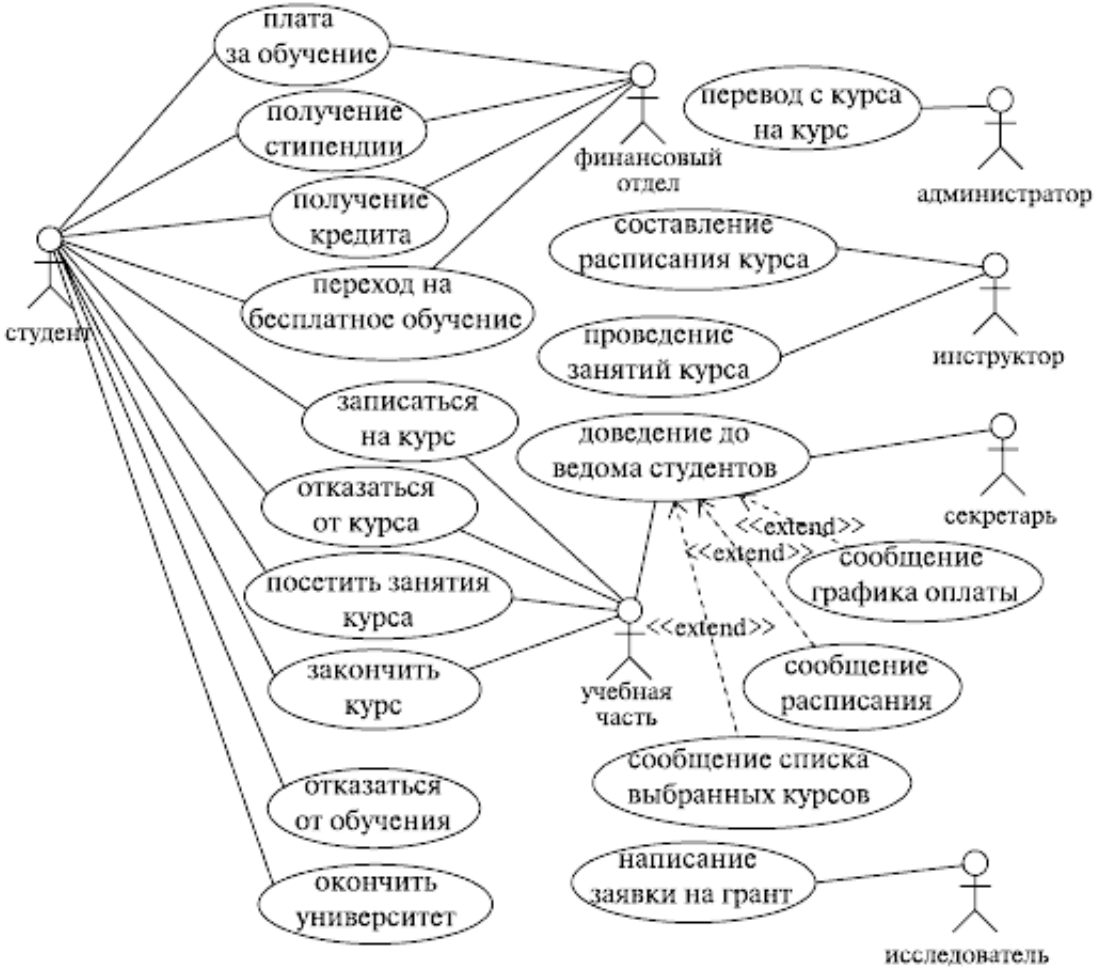
Примечание. Среди акторов могут быть не только пользователи, но и внешние системы и внутренние подсистемы.







ПРИМЕР ДИАГРАММЫ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ



ПРИМЕНЕНИЕ ДИАГРАММ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ. МОДЕЛИРОВАНИЕ КОНТЕКСТА СИСТЕМЫ

В UML можно **моделировать контекст системы с помощью диаграммы вариантов использования**, изображающей действующие лица, окружающие систему.

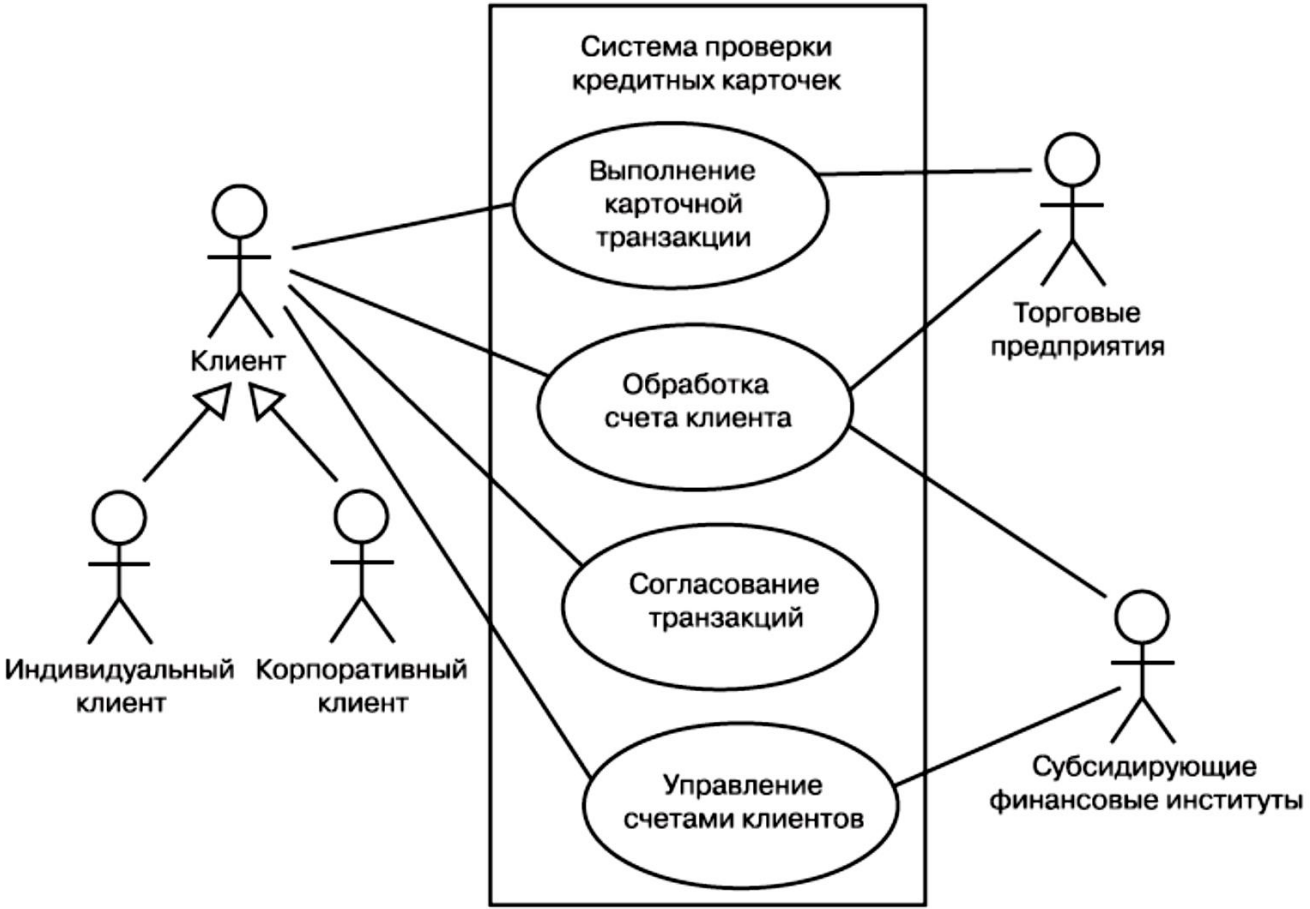
Решение о том, что именно включить или не включать в диаграмму в качестве действующих лиц, важно постольку, таким образом специфицируете класс сущностей, взаимодействующих с системой.

Для моделирования контекста системы, необходимо:

- Идентифицировать границы системы, приняв решение о том, какое поведение является ее частью, а какое осуществляют внешние сущности. Этим определяется субъект.
- Идентифицировать действующие лица, окружающие систему, рассмотрев при этом, какие группы требуют помощи со стороны системы в выполнении своих задач, какие необходимы для обеспечения функционирования системы, какие взаимодействуют с внешним оборудованием или другими программными системами и какие осуществляют вторичные функции по администрированию и поддержке.
- Организовать схожие действующие лица в иерархии обобщения–специализации.
- Там, где это требуется для лучшего понимания модели, представить стереотип для каждого из действующих лиц.

Диаграммы вариантов использования применимы для **моделирования контекста подсистем** в составе системы.

ПРИМЕР МОДЕЛИРОВАНИЯ КОНТЕКСТА СИСТЕМЫ.
Контекст системы проверки кредитных карт



МОДЕЛИРОВАНИЕ ТРЕБОВАНИЙ К СИСТЕМЕ

Требования могут быть выражены в различных формах – от неструктурированного текста до выражений формального языка. Большинство функциональных требований к системе могут быть выражены в виде вариантов использования, поэтому для управления этими требованиями используются диаграммы вариантов использования UML.

Для моделирования требований к системе, необходимо:

- Установить контекст системы, идентифицируя действующие лица, окружающие ее.
- Рассмотреть поведение системы, которого от нее ожидает или требует каждое действующее лицо.
- Обобщить это поведение в виде вариантов использования.
- Выделить общее поведение в новые варианты использования, на которые ссылаются другие варианты; выделить разновидности поведения в новые варианты использования, расширяющие основные потоки.
- Смоделировать эти варианты использования, действующие лица и их связи на диаграмме вариантов использования.
- Снабдить варианты использования примечаниями или ограничениями, которые задают нефункциональные требования, рассмотреть целесообразность включения некоторых из них в проектируемую систему.

ПРИМЕР МОДЕЛИРОВАНИЯ ТРЕБОВАНИЙ К СИСТЕМЕ.
Структурная схема ИС управления торговыми операциями

Распределение основных потоков

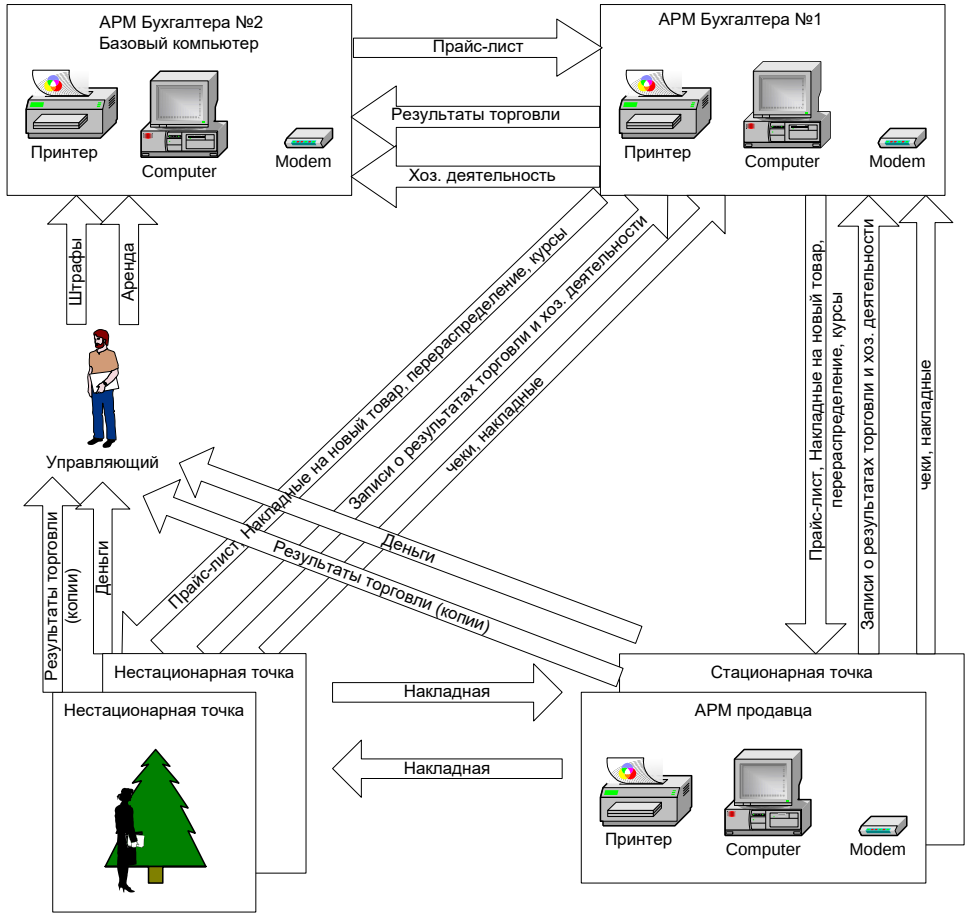
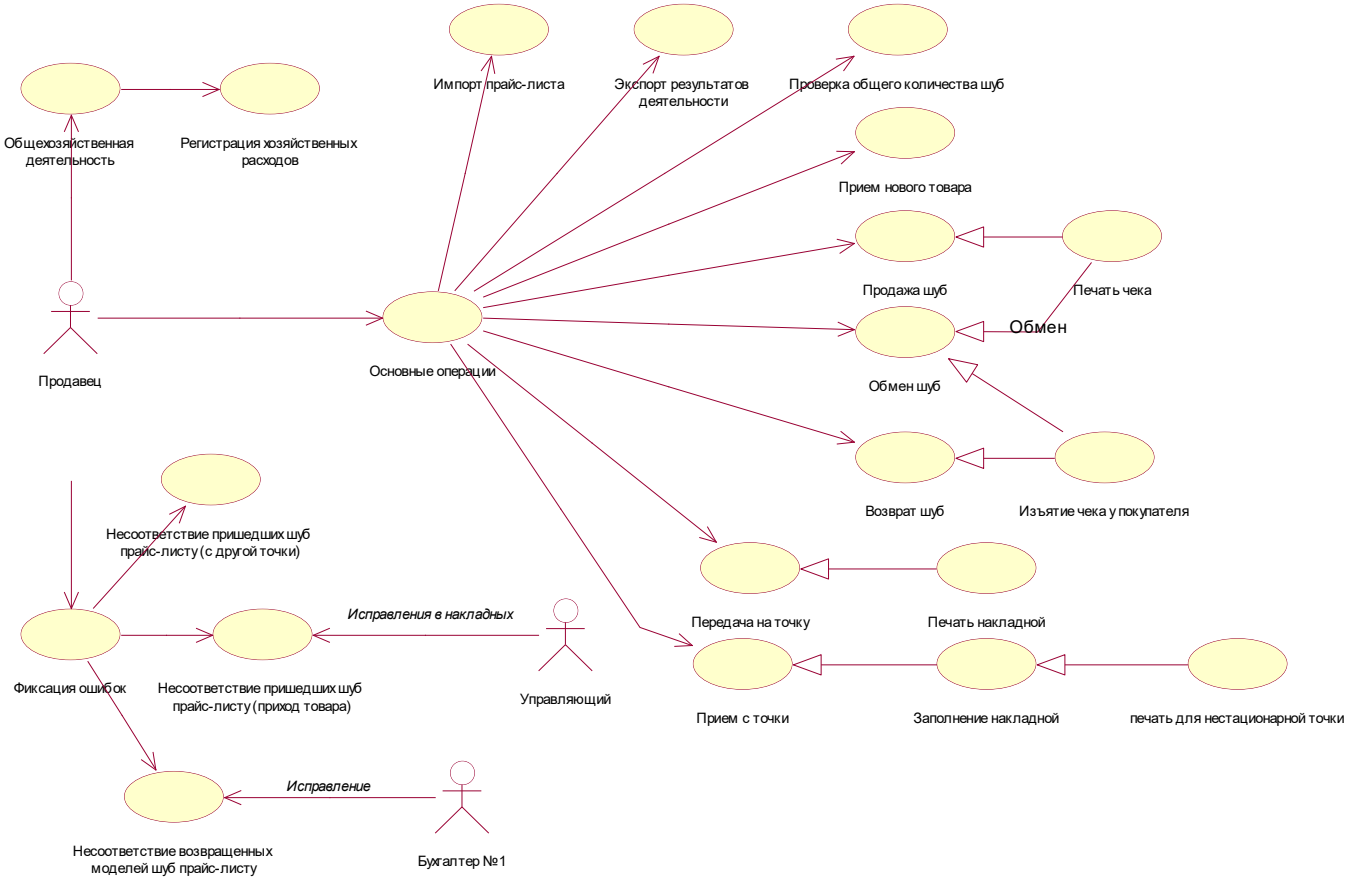


ДИАГРАММА ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ АРМА ПРОДАВЦА



ОСОБЕННОСТИ ПРОЕКТИРОВАНИЯ С ИСПОЛЬЗОВАНИЕ ДИАГРАММ

Прямое проектирование (forward engineering) – это процесс трансформации модели в код путем ее отображения на язык реализации. **Диаграмма вариантов использования** может быть подвергнута прямому проектированию с целью формирования тестов (а также контрольных примеров) для того элемента, который она описывает. Каждый вариант использования на такой диаграмме специфицирует поток событий с его вариациями, и все эти потоки описывают ожидаемое поведение элемента, которое подлежит тестированию. Для каждого варианта использования из диаграммы вы можете создать сценарий тестирования (test case), который будет выполняться при выпуске каждой новой версии данного элемента.

Чтобы осуществить прямое проектирование диаграммы вариантов использования, необходимо:

- Идентифицировать объекты, которые взаимодействуют с системой и роли, которые может играть каждый внешний объект.
- Создать действующее лицо, представляющее каждую из отдельных взаимодействующих ролей.
- Для каждого варианта использования на диаграмме идентифицировать основной и исключительный потоки событий.
- Сгенерировать соответствующий сценарий для каждого потока, используя предусловие потока как начальное состояние теста и постусловие – как критерий успешности.
- При необходимости сгенерировать тестовую инфраструктуру, которая будет представлять каждое из действующих лиц, взаимодействующих с вариантом использования.
- Использовать инструментальные средства для запуска каждого из этих тестов всякий раз, при реализации элементов, относящихся к диаграмме вариантов использования.

ОСОБЕННОСТИ ПРОЕКТИРОВАНИЯ С ИСПОЛЬЗОВАНИЕ ДИАГРАММ. ОБРАТНОЕ ПРОЕКТИРОВАНИЕ

Обратное проектирование (reverse engineering) – процесс трансформации кода в модель посредством его отображения из определенного языка реализации. Автоматически выполнить обратное проектирование диаграммы вариантов использования не представляется возможным, потому что на пути от спецификации поведения элемента к его реализации происходит потеря информации. Необходимо провести экспертный анализ реализации системы.

Чтобы **осуществить обратное проектирование диаграммы вариантов использования, необходимо**:

- Идентифицировать каждое действующее лицо, взаимодействующее с системой.
- Исследовать манеру взаимодействия каждого из них с системой, изменение ее состояния или состояния среды, реакцию на события.
- Проследить поток событий в исполняемой системе относительно каждого действующего лица, начав с основных потоков и во вторую очередь изучив альтернативные.
- Объединить в группы взаимосвязанные потоки, определяя соответствующие варианты использования. Рассмотреть варианты моделирования с применением связей расширения и моделирование похожих потоков с применением связей включения.
- Отобразить эти варианты использования и действующие лица на диаграмме вариантов использования и установить их связи.

РЕКОМЕНДАЦИИ ПО МОДЕЛИРОВАНИЮ

Диаграммы вариантов использования, взятые в совокупности, формируют статическое представление вариантов использования системы; каждая из них в отдельности выражает только какой-то один аспект.

Хорошо структурированная **диаграмма вариантов использования обладает следующими характеристиками**:

- сконцентрирована на передаче одного аспекта представления системы с точки зрения вариантов использования;
- содержит только те варианты использования и действующие лица, которые важны для понимания данного аспекта;
- обеспечивает представление, соответствующее ее уровню абстракции: показывает только те дополнения (в частности, точки расширения), которые важны для понимания диаграммы;
- отражает важную семантику.

При **отображении диаграммы вариантов использования** необходимо:

- присваивать ей имя, соответствующее ее назначению;
- располагать ее элементы так, чтобы пересекающихся линий было как можно меньше;
- располагать рядом семантически близкие по поведению и ролям элементы;
- использовать примечания и цветовое выделение для того, чтобы привлечь внимание читателя к важным особенностям;
- показывать не слишком много видов связей. Сложные связи включения и расширения, целесообразно выносить в отдельную диаграмму.

РЕКОМЕНДАЦИИ ПО МОДЕЛИРОВАНИЮ. ПРОДОЛЖЕНИЕ

Хорошо описанный вариант использования имеет **следующие атрибуты**:

- Имя, ясно говорящее о назначении варианта использования.
- Описание. Несколько предложений, описывающих этот вариант использования.
- Частота. Насколько часто данный вариант использования возникает.
- Предусловия. Все условия запуска варианта использования.
- Постусловия. Все условия, которые должны быть выполнены после успешного выполнения варианта использования.
- Основной сценарий работы, который используется в большинстве случаев.
- Альтернативные сценарии, возникающие иногда. Для каждого альтернативного сценария указываются условия его запуска.
- Задействованные действующие лица.
- Расширяемые варианты использования.
- Включаемые варианты использования.
- Статус: "в разработке", "готов к проверке", "в процессе проверки", "подтвержден", "отвергнут".
- Допущения об окружении и ходе работы системы, использованные при разработке данного варианта. В полностью готовом варианте эти допущения либо должны быть подтверждены и стать ограничениями системы, либо должны давать начало различным сценариям работы.

Варианты использования могут дополняться диаграммами других видов — прежде всего диаграммами последовательностей, описывающими последовательности действий участвующих компонентов, диаграммами состояний и переходов компонентов и диаграммами классов этих компонентов, и др.

ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТЕЙ (SEQUENCE DIAGRAM)

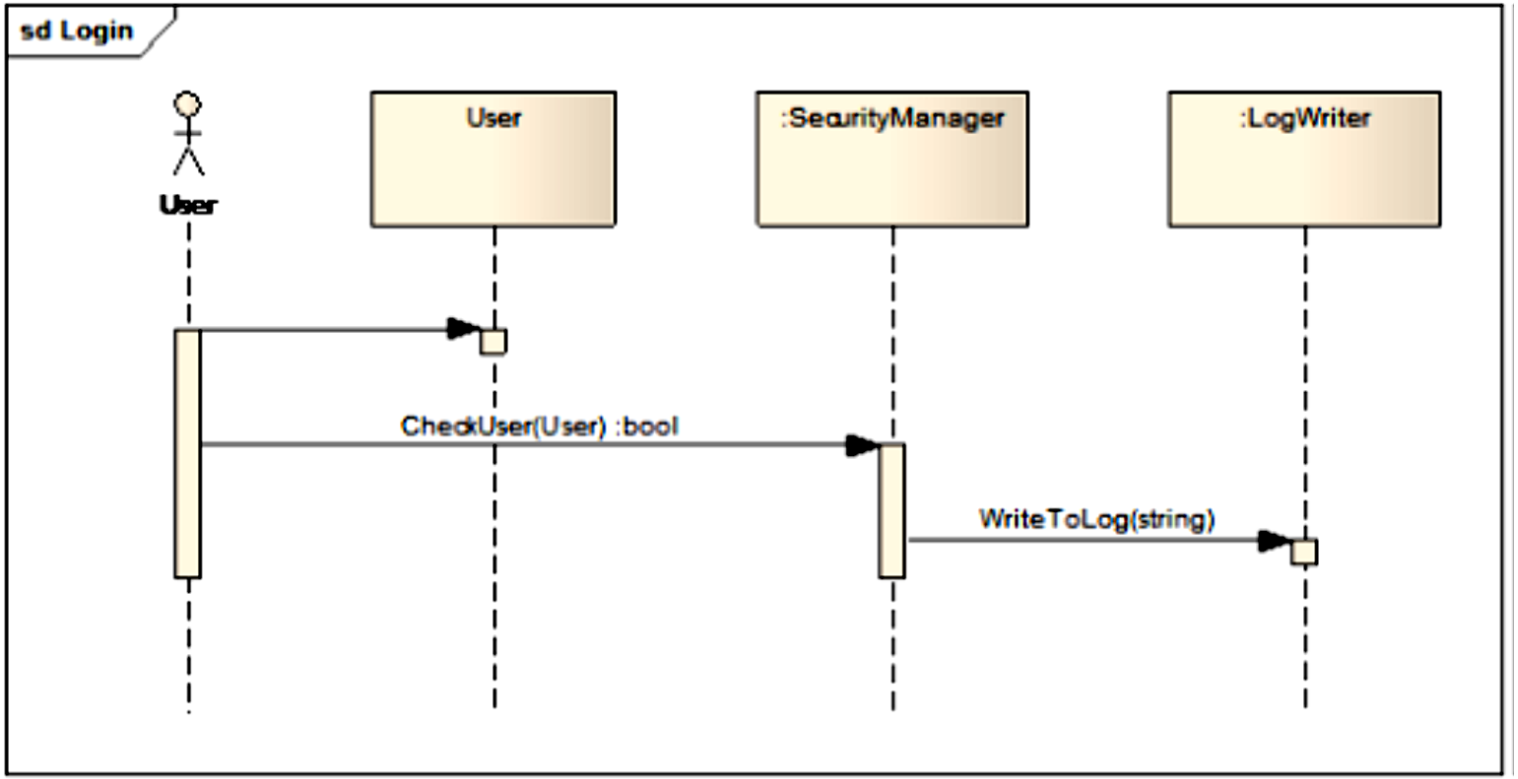
Диаграмма последовательностей служит основным способом расшифровки последовательности действий в процессе выполнения того или иного варианта использования. Она отвечает на вопрос «Как работает система при выполнении данного варианта использования?».

Диаграмма последовательностей всегда создается в привязке к варианту использования. Каждый вариант использования может содержать несколько диаграмм последовательностей, на тот случай, если они описывают несколько альтернативных вариантов развития событий.

Диаграмма последовательностей, так же, как и вариант использования, может быть **реализована** как в терминах бизнес-объектов, так и в терминах физических сущностей, таких как компоненты или классы.

Диаграмма последовательностей всегда начинается с актора, инициирующего процесс. Вверху диаграммы располагаются элементы, классы или компоненты, которые задействованы в процессе работы.

ПРИМЕРЫ ДИАГРАММ ПОСЛЕДОВАТЕЛЬНОСТЕЙ



ПРИМЕРЫ ДИАГРАММ ПОСЛЕДОВАТЕЛЬНОСТЕЙ

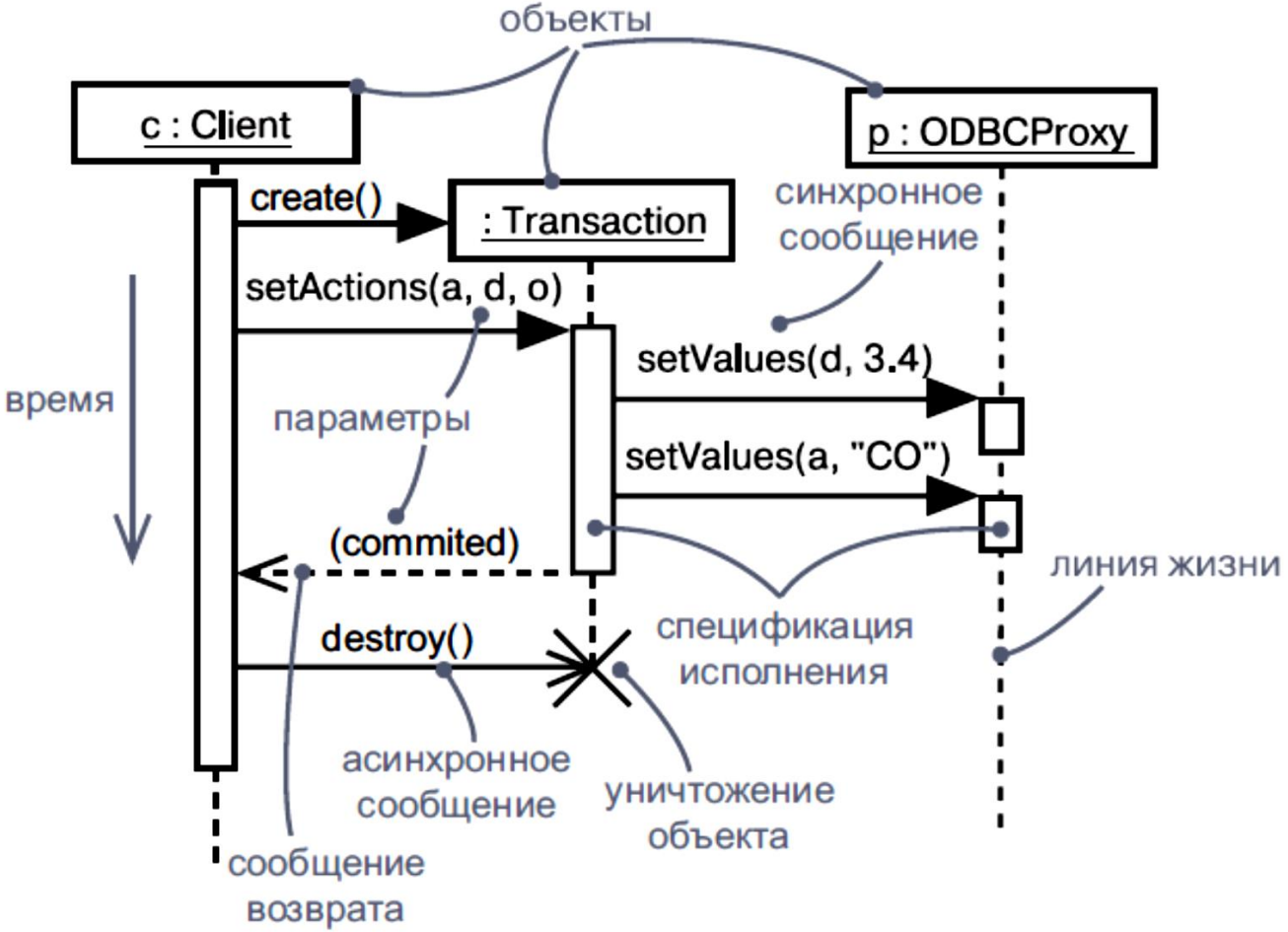


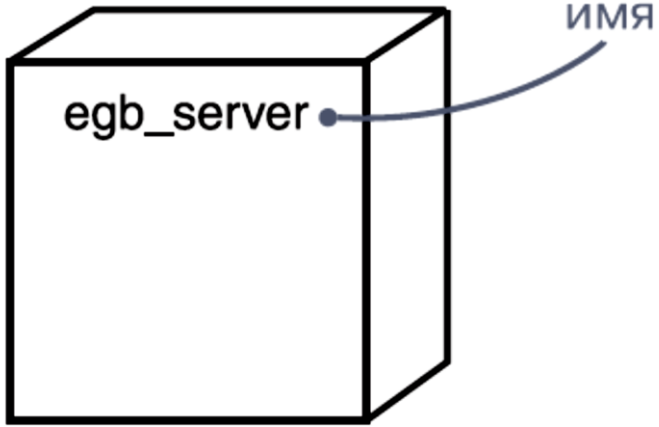
ДИАГРАММА РАЗМЕЩЕНИЯ (DEPLOYMENT DIAGRAM)

Диаграмма размещения отражает физические взаимосвязи между программными и аппаратными компонентами системы. Она позволяет показать маршруты перемещения объектов и компонентов в распределенной системе.

Диаграмма размещения показывает физическое расположение телекоммуникационной сети и местонахождение в ней различных компонентов.

Каждый **узел** на диаграмме размещения представляет собой некоторый тип вычислительного устройства – в большинстве случаев, часть аппаратуры.

Используя каноническое обозначение узла на диаграмме можно визуализировать его, не конкретизируя стоящей за ним аппаратуры.



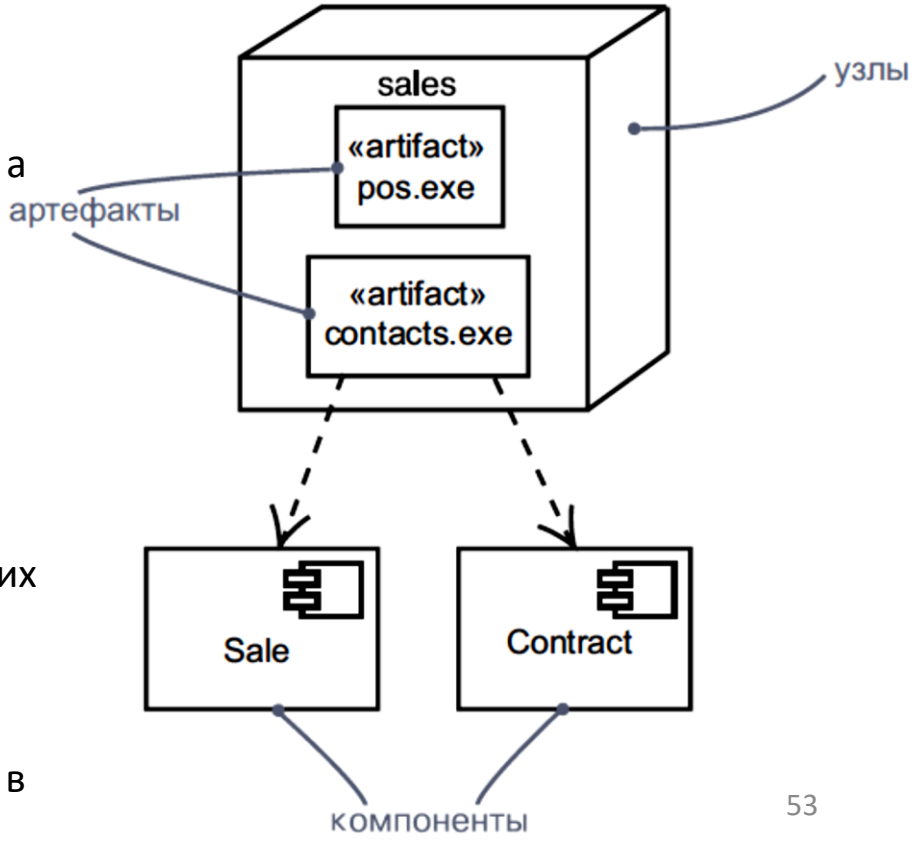
УЗЛЫ И АРТЕФАКТЫ

Во многих отношениях узлы подобны артефактам. Те и другие наделены именами, могут участвовать в связях зависимости, обобщения и ассоциации, бывают вложенными, могут иметь экземпляры и вступать во взаимодействия. Однако между ними есть и существенные **различия**:

- артефакты принимают участие в работе системы, а узлы – это сущности, на которых работают артефакты;
- артефакты представляют физическую упаковку логических элементов, узлы представляют физическое размещение артефактов.

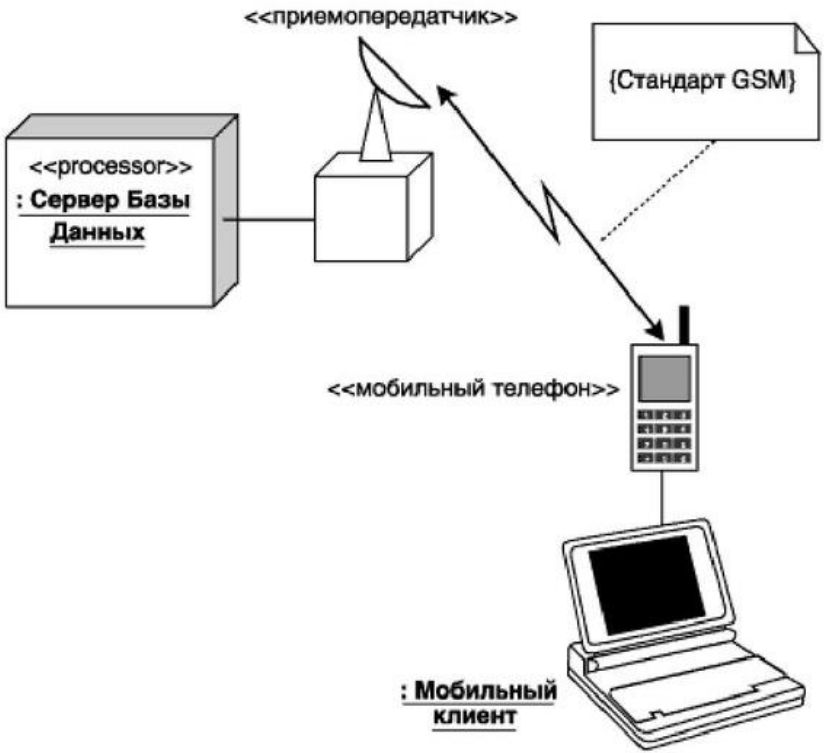
Узлы исполняют артефакты, артефакты работают на узлах.

Артефакт – это материализация множества логических элементов, таких как классы и кооперации, а узел – место, на котором размещены артефакты. Класс может быть реализован одним или несколькими артефактами, а артефакт, в свою очередь, размещен в одном или нескольких узлах.

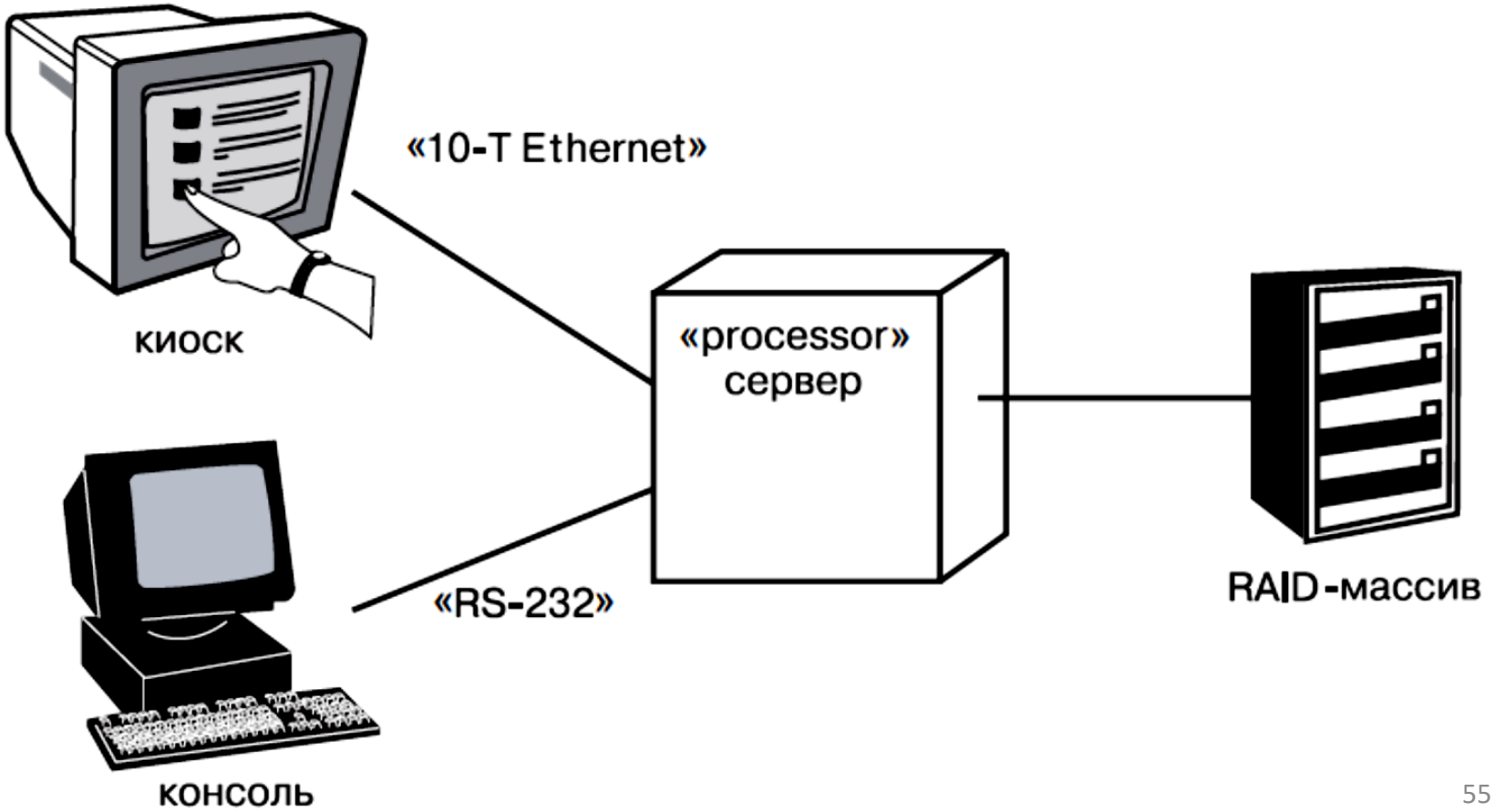


ПРИМЕР ДИАГРАММЫ РАЗМЕЩЕНИЯ

Мобильный доступ к корпоративной БД



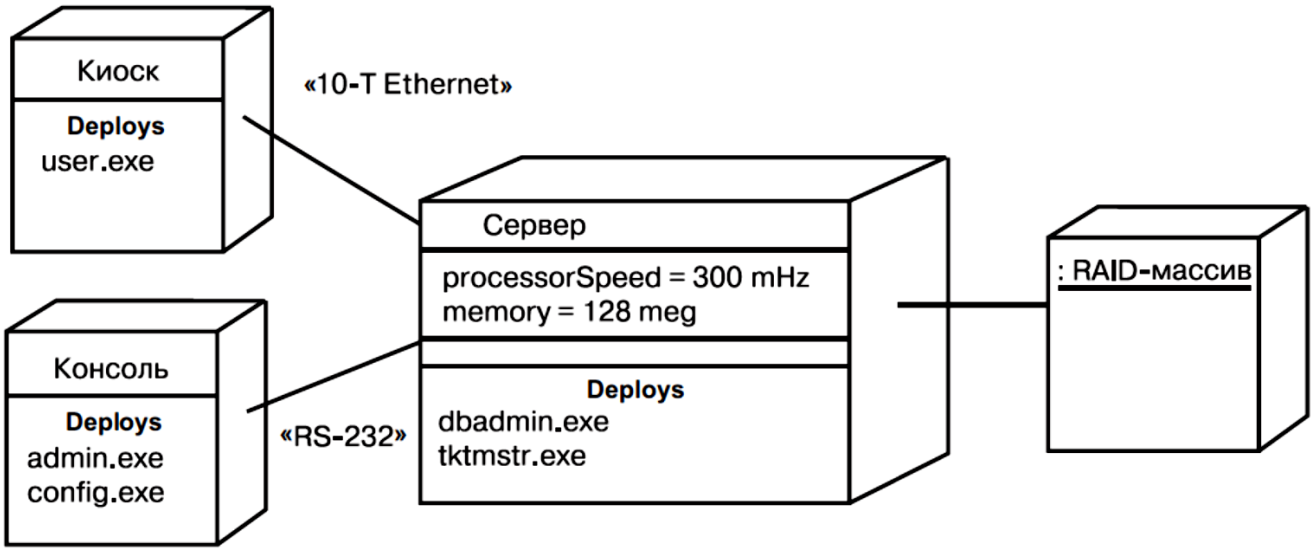
ИСПОЛЬЗОВАНИЕ ДИАГРАММ РАЗМЕЩЕНИЯ. МОДЕЛИРОВАНИЕ ПРОЦЕССОВ И УСТРОЙСТВ



ИСПОЛЬЗОВАНИЕ ДИАГРАММ РАЗМЕЩЕНИЯ. МОДЕЛИРОВАНИЕ РАСПРЕДЕЛЕНИЯ АРТЕФАКТОВ

При моделировании топологии системы бывает полезно визуализировать или специфицировать физическое распределение ее артефактов по процессорам и устройствам, входящим в состав системы. Моделирование распределения артефактов состоит из следующих шагов:

1. Припишите каждый значимый компонент системы к определенному узлу.
2. Рассмотрите возможности дублирования размещения артефактов. Часто одни и те же артефакты (например, некоторые исполняемые программы и библиотеки) размещаются одновременно в нескольких узлах.
3. Изобразите распределение артефактов по узлам.



РЕКОМЕНДАЦИИ ПО МОДЕЛИРОВАНИЮ

Хорошо структурированный узел обладает следующими свойствами:

- представляет абстракцию некоторой сущности из словаря аппаратных средств области решения;
- декомпозирован лишь до такого уровня, который необходим, чтобы выразить замысел разработчика;
- раскрывает только те атрибуты и операции, которые относятся к моделируемой области;
- явно показывает набор артефактов, которые на нем расположены;
- соединен с другими узлами способом, отражающим топологию реальной системы.

При изображении узла на UML-диаграмме необходимо :

- определить набор стереотипов с подходящими пиктограммами, которые несут очевидную для всех смысловую нагрузку;
- показывать только те атрибуты и операции (если таковые существуют), которые необходимы для понимания назначения узла в данном контексте.

РЕКОМЕНДАЦИИ ПО ИСПОЛЬЗОВАНИЮ UML-ДИАГРАММ

При выборе диаграмм необходимо ответить на вопросы:

- Какие именно виды диаграмм лучше всего отражают архитектуру системы и возможные технические риски, связанные с проектом?
- Какие из диаграмм удобнее всего превратить в инструмент контроля над процессом (и прогрессом) разработки системы?

Модели предметной области задачи можно строить в виде диаграммы классов. Это хороший способ понять, как визуализировать множества взаимосвязанных абстракций для построения модели статической части задач. Для моделирования динамической части задачи можно использовать простые диаграммы последовательностей и кооперации.

Моделирование целесообразно начать с анализа взаимодействия пользователя с системой - так возможно выделить наиболее важные прецеденты.

Можно предложить такую последовательность построения диаграмм при разработке объектно-ориентированных систем:

1. диаграмма прецедентов,
2. диаграмма классов,
3. диаграмма объектов,
4. диаграмма последовательностей,
5. диаграмма кооперации,
6. диаграмма состояний,
7. диаграмма активности,
8. диаграмма размещения.

РЕКОМЕНДАЦИИ ПО ИСПОЛЬЗОВАНИЮ UML-ДИАГРАММ. ПРОДОЛЖЕНИЕ

- Полезно строить модели классов и отношений между ними для уже написанного кода на C++ или Java.
- Целесообразно применять UML для того, чтобы прояснить неявные детали реализации существующей системы или использованные в ней "хитрые механизмы программирования".
- Рекомендуется строить UML-модели, прежде чем начать новый проект. После получения удовлетворительных для всех разработчиков результатов моделирования, можно использовать их как основу для кодирования.
- Необходимо обратить внимание на средства UML для моделирования компонентов, параллельности, распределенности, паттернов проектирования.
- Необходимо использовать средства расширения UML для предметной области решаемой задачи. Число таких средств надо стараться ограничивать.
- Целесообразно выбрать пакет UML-моделирования (CASE-средства) с учетом индивидуального стиля проектирования.

ОСОБЕННОСТИ МОДЕЛИРОВАНИЯ АРХИТЕКТУРЫ СИСТЕМЫ

Системная архитектура является наиболее важным продуктом, который может быть использован для **управления разнообразными точками зрения** на систему и тем самым управляет итеративным и пошаговым процессом ее разработки на протяжении всего ее жизненного цикла.

Архитектура – это набор существенных решений относительно:

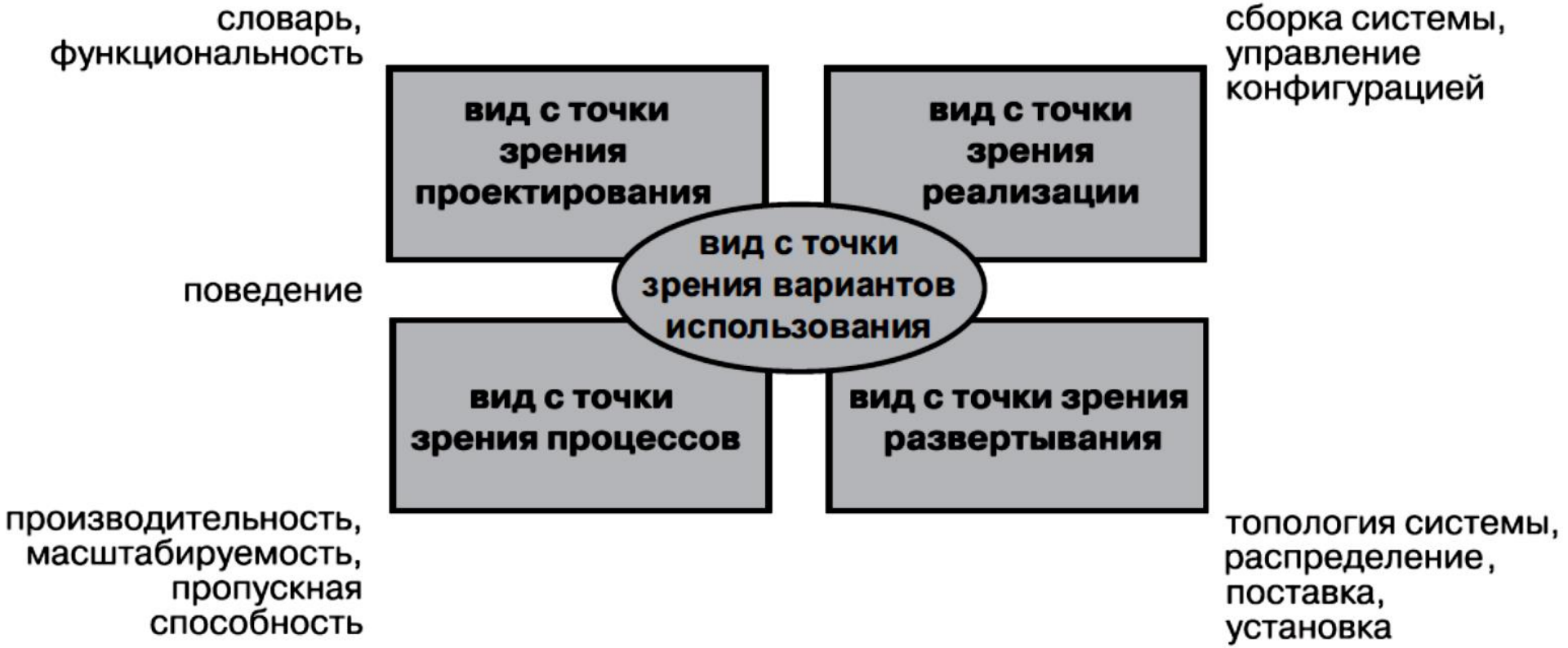
- организации программной системы;
- выбора структурных элементов, составляющих систему, и их интерфейсов;
- поведения этих элементов, определенного в их кооперациях;
- объединения этих структурных и поведенческих элементов в более крупные подсистемы;
- архитектурного стиля, определяющего организацию системы: статические и динамические элементы и их интерфейсы, кооперацию и композицию.

Архитектура программного обеспечения касается не только его структуры и поведения, но также пользовательских свойств, функциональности, производительности, гибкости, возможности повторного использования, понятности, экономических и технологических ограничений и компромиссов, а также эстетических вопросов.

Для моделирования архитектуры системы необходимо:

- Идентифицировать представления, которые вы будете использовать для моделирования архитектуры. Чаще всего это представления с точки зрения вариантов использования, проектирования, взаимодействия, реализации и размещения.
- Специфицировать контекст системы, включая окружающие ее действующие лица.
- При необходимости декомпозировать систему на элементарные подсистемы.

МОДЕЛИРОВАНИЕ СИСТЕМНОЙ АРХИТЕКТУРЫ



ЗАДАЧИ МОДЕЛИРОВАНИЯ АРХИТЕКТУРЫ СИСТЕМЫ

- Специфицировать представление системы **с точки зрения вариантов использования**, которые описывают поведение системы, каким оно представляется конечным пользователям, аналитикам и тестировщикам. Для моделирования статических аспектов применить **диаграммы вариантов использования**, а для моделирования динамических – **диаграммы взаимодействия, состояния и деятельности**.
- Специфицировать представление **системы с точки зрения проектирования**, куда входят классы, интерфейсы и кооперации, формирующие словарь предметной области и предлагаемого решения. Для моделирования статических аспектов применить **диаграммы классов и объектов**, а для моделирования динамических – **диаграммы взаимодействия, состояний и деятельности**.
- Специфицировать представление системы **с точки зрения взаимодействия**, куда входят потоки, процессы и сообщения, формирующие механизмы параллелизма и синхронизации в системе. Использовать те же диаграммы, что и для представления проектирования, но основное внимание уделить активным классам и объектам, которыми представлены процессы и потоки, а также сообщениям и потоку управления.
- Специфицировать представление системы **с точки зрения реализации**, куда входят артефакты, используемые для сборки и выпуска физической системы. Для моделирования статических аспектов использовать **диаграммы артефактов**, для моделирования динамических – **диаграммы взаимодействия**, **состояний и деятельности**.
- Специфицировать представление системы **с точки зрения размещения**, куда входят узлы, формирующие топологию аппаратных средств, на которых работает система. Для моделирования статических аспектов применить **диаграммы размещения**, а для моделирования динамических – **диаграммы взаимодействия**, **состояний и деятельности**.
- Смоделировать **архитектурные образцы** и **образцы проектирования**, формирующие перечисленные модели, с помощью кооперации.

ОСОБЕННОСТИ МОДЕЛИРОВАНИЯ СИСТЕМЫ СИСТЕМ

По мере возрастания сложности системы возникает необходимость ее декомпозиции на более простые, каждую из которых можно разрабатывать отдельно, а затем постепенно объединять. Разработка подсистемы и системы подчиняется одним и тем же принципам.

Для моделирования системы систем необходимо:

- Идентифицировать основные функциональные составляющие системы, которые можно разрабатывать, выпускать и размещать до некоторой степени независимо. На результаты этого разбиения системы часто влияют технические, политические и юридические факторы.
- Для каждой подсистемы специфицировать ее контекст так же, как это делается для системы в целом (в число действующих лиц, окружающих подсистему, включаются все соседние подсистемы, поэтому необходимо проектировать их совместную работу).
- Смоделировать архитектуру каждой подсистемы так же, как это делается для всей системы.

Важно выбрать **правильное множество моделей** для визуализации, специфицирования, конструирования и документирования системы.

ПРИЗНАКИ ХОРОШО СТРУКТУРИРОВАННОЙ МОДЕЛИ

- Дает упрощенное представление реальности с одной точки зрения.
- Самодостаточна, то есть не требует для понимания ее семантики никакой дополнительной информации.
- Слабо связана с другими моделями посредством связей трассировки.
- Совместно с другими смежными моделями дает полное представление об артефактах системы.

Хорошо структурированная модель системы систем:

- функционально, логически и физически согласована;
- может быть декомпозирована на почти независимые подсистемы, которые сами являются системами на более низком уровне абстракции;
- может быть визуализирована, специфицирована, сконструирована и документирована в виде набора взаимосвязанных, неперекрывающихся моделей.

ОСОБЕННОСТИ ИСПОЛЬЗОВАНИЯ НЕКОТОРЫХ ДИАГРАММ UML НА РАЗНЫХ ЭТАПАХ ЖЦ ПРОЕКТИРУЕМОЙ СИСТЕМЫ

Фаулер М., Скотт К. UML в кратком изложении. Применение стандартного языка объектного моделирования. – М.: Мир, 1999. – 191 с.

Стадии и этапы ЖЦ	Рекомендации по применению UML-диаграмм
Формирование требований к ИС, Техническое задание	**Диаграммы вариантов использования**. Их совокупность образует внешнее представление системы, обеспечивает понимание потребностей пользователей, определяет **базовую архитектуру системы**, может использоваться для планирования и управления проектом, а также для оценки рисков, связанных с требованиями. **Диаграммы классов** для моделирования понятий языка пользователей на концептуальном уровне. **Диаграммы деятельностей** для моделирования основных аспектов поведения системы и поиска параллельных процессов.
Эскизный проект, Технический проект	**Диаграммы классов**, **Диаграммы объектов**, **Диаграммы взаимодействий, Диаграммы размещений** и др. виды диаграмм для проектирования системы. **Диаграммы вариантов использования** для разработки тестов и ПМИ.
Разработка и адаптация программ, Проведение испытаний	Использование обратного проектирования некоторых диаграмм для верификации разработанных решений требованиям ТЗ (например, **Диаграмм вариантов использования**).

ОБЗОР МЕТОДОЛОГИИ RUP

Язык UML не зависит от организации процесса разработки, не привязан к какому-либо конкретному жизненному циклу проектирования и его можно использовать в различных процессах разработки программного обеспечения.

Унифицированный процесс разработки программного обеспечения (**Rational Unified Process, RUP**) – это один из подходов к организации жизненного цикла программного обеспечения, который особенно хорошо сочетается с UML.

Цель RUP – обеспечить создание программного продукта высочайшего качества, соответствующего потребностям пользователя, в заданные сроки и в пределах запланированного бюджета.

С **точки зрения управления проектом** RUP предлагает упорядоченный подход к способам распределения заданий и обязанностей в организации, занимающейся разработкой программного обеспечения.

ХАРАКТЕРИСТИКИ МЕТОДОЛОГИИ RUP

RUP – это итерационный процесс. Итерационный подход предполагает постепенное проникновение в суть проблемы путем последовательных уточнений и пошагового наращивания решения на протяжении нескольких циклов. Присущая итерационному процессу внутренняя гибкость позволяет включать в бизнес-цели новые требования и тактические изменения. Его применение обеспечивает возможность выявлять и устранять риски, связанные с проектом, на возможно более ранних стадиях разработки.

Суть деятельности в рамках RUP – создание и сопровождение моделей, а не бумажных документов. Модели, особенно выраженные на UML, обеспечивают семантически богатое представление программной системы в процессе разработки.

Разработка в рамках RUP сконцентрирована на архитектуре. Основное внимание уделяется ранней разработке и стабилизации архитектуры программного обеспечения.

Разработка в рамках RUP управляется вариантами использования. Варианты использования и сценарии применяются на всех стадиях процесса – от формулирования требований до тестирования.

RUP поддерживает объектно-ориентированные методы.

RUP – конфигурируемый процесс. RUP поддается настройке и масштабируется для использования как в совсем небольших коллективах разработчиков, так и в больших компаниях.

RUP обеспечивает постоянный контроль качества и управление риском. Контроль качества встроен в сам процесс распространяется на все виды деятельности в рамках разработки и всех ее участников; при этом используются объективные критерии и методы оценки рисков.

ФАЗЫ И ИТЕРАЦИИ RUP

Фаза (phase) – это промежуток времени между двумя важными опорными точками процесса, в которых должны быть достигнуты четко определенные цели, подготовлены те или иные рабочие продукты и приняты решения о том, можно ли переходить к следующей фазе.

RUP состоит из следующих четырех фаз:

1. Начальная фаза (inception) – определение границ проекта, разработка концепции и начального плана проекта;
2. Разработка (elaboration) – проектирование, реализация и тестирование стабильной архитектуры, завершение разработки плана проекта;
3. Конструирование (construction) – построение первой эксплуатационной версии системы;
4. Внедрение (transition) – поставка системы конечным пользователям.

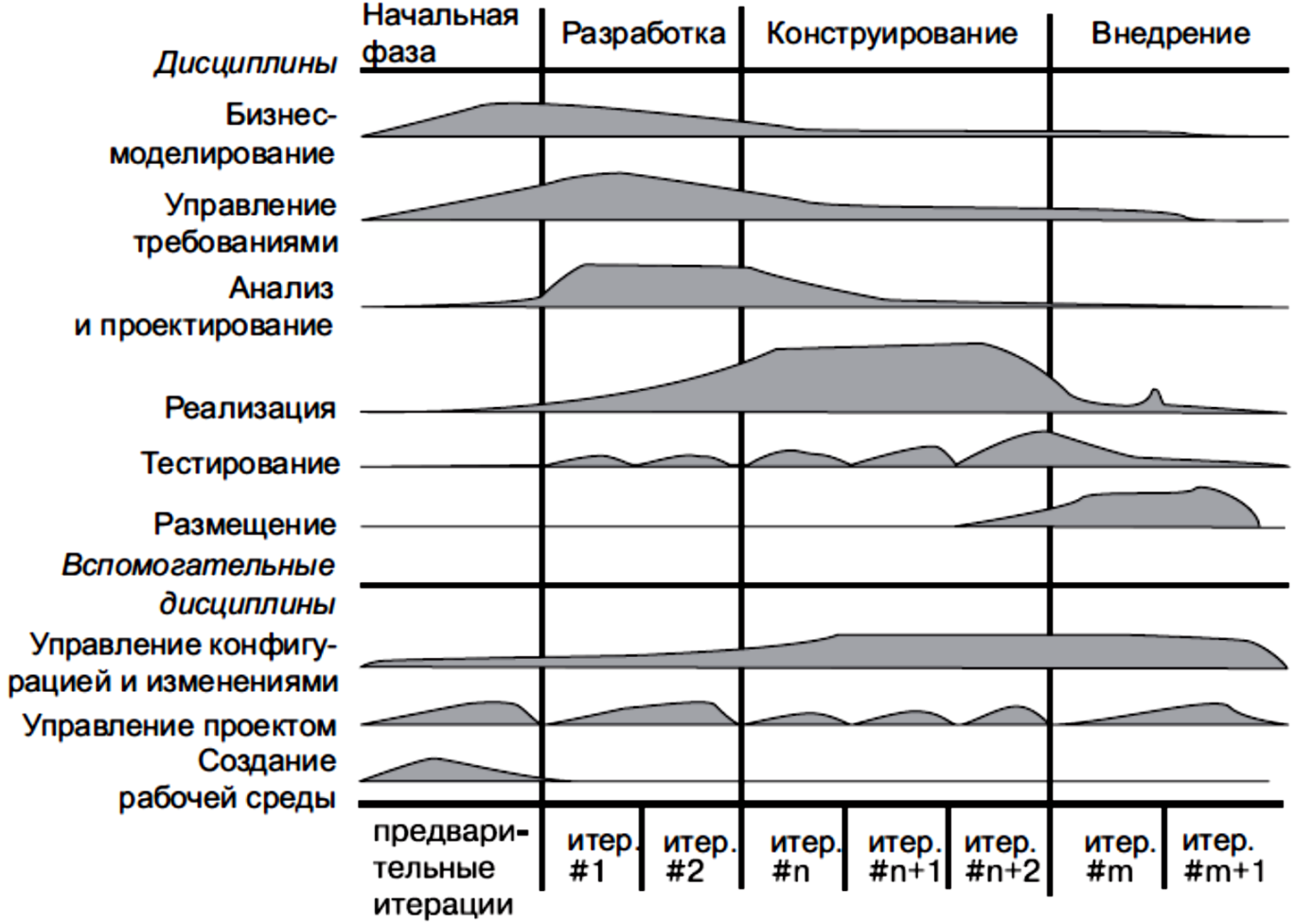
В пределах каждой фазы может происходить множество итераций.

Итерация представляет собой полный цикл разработки, в результате которого появляется работающая версия, – от анализа и формулирования требований до реализации и тестирования. **Назначение работающий версии** – обеспечить прочную базу для последующей оценки и тестирования, подвести промежуточный итог, необходимый для начала следующего цикла разработки.

Каждая фаза и итерация предполагает определенные трудозатраты, направленные на снижение риска.

В конце каждой фазы **определяется опорная точка** (milestone), позволяющая оценить, в какой мере достигнуты намеченные цели и следует ли внести в процесс какие-либо изменения, прежде чем двигаться дальше.

ЖЦ ПРОЦЕССА РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ



ДИСЦИПЛИНЫ RUP

RUP состоит из девяти дисциплин:

1. Бизнес-моделирование (business modeling) – описывает структуру и динамику организации-заказчика;
2. Управление требованиями (requirements) – выявляет требования на основе множества подходов;
3. Анализ и проектирование (analysis & design) – описывает множество архитектурных представлений системы;
4. Реализация (implementation) – собственно, разработка программного обеспечения, модульное тестирование и интеграция;
5. Тестирование (test) – описывает тестовые сценарии, процедуры и метрики для оценки дефектов;
6. Размещение (deployment) – включает спецификацию материалов, описание версии, обучение и другие аспекты, связанные с поставкой системы;
7. Управление конфигурацией и изменениями (configuration management) – сфокусировано на изменениях, поддержке целостности рабочих продуктов проекта и управленческой деятельности;
8. Управление проектом (project management) – описывает различные стратегии работы в условиях итерационного процесса;
9. Создание рабочей среды (environment) – рассматривает необходимую инфраструктуру для разработки системы.

ВИДЫ ДЕЯТЕЛЬНОСТИ И РАБОЧИЕ ПРОДУКТЫ

Каждая дисциплина охватывает свой набор связанных рабочих продуктов и видов деятельности.
Рабочий продукт – это некоторый документ, отчет или исполняемая программа, которые после их создания преобразуются или потребляются.
Вид деятельности описывает комплекс задач – этапы продумывания, выполнения, анализа, – выполняемых сотрудниками с целью создания или модификации рабочих продуктов, а также способы выполнения этих задач и соответствующие рекомендации. В число таких способов может входить применение инструментальных средств, позволяющих автоматизировать ряд процессов.
В некоторых из этих дисциплин установлены важные связи между рабочими продуктами.
Например, **модель вариантов использования**, созданная в ходе сбора требований, реализуется в виде проектной модели, выработанной в ходе анализа и проектирования, воплощается в модели дисциплины реализации и верифицируется моделью дисциплины тестирования.
С каждым видом деятельности в рамках RUP ассоциированы рабочие продукты (входные или выходные).
Модели – наиболее важная разновидность рабочих продуктов в RUP.
Представление (view) – это проекция модели.
В **RUP архитектура системы включает** пять тесно связанных друг с другом **представлений**: проектирования, взаимодействия, размещения, реализации и вариантов использования.

ОСНОВНЫЕ МОДЕЛИ RUP

1. **Модель бизнес-процессов** (business use case model) описывает деятельность организации;
2. **Модель бизнес-анализа** (business analysis model) определяет контекст системы;
3. **Модель вариантов использования** (use case model) выражает функциональные требования к системе;
4. **Модель анализа** (analysis model) – необязательная. Определяет проектные решения на концептуальном уровне;
5. **Проектная модель** (design model) охватывает словарь предметной области и области решения;
6. **Модель данных** (data model) – необязательная. Определяет представление данных в базах данных и других репозиториях;
7. **Модель размещения** (deployment model) специфицирует топологию аппаратных средств, на которых работает система, вместе с механизмами параллелизма и синхронизации;
8. **Модель реализации** (implementation model) определяет, какие части используются для сборки и реализации физической системы.

ГРУППЫ РАБОЧИХ ПРОДУКТОВ

Рабочие продукты RUP подразделяются на две группы: продукты управления и технические продукты. Последние, в свою очередь, делятся на пять основных подгрупп:

1. **Группа требований** (requirements set) описывает, что должна делать система. В составе этих рабочих продуктов могут быть **модель вариантов использования**, нефункциональные требования, модель предметной области, модель анализа и другие формы выражения потребностей пользователей, включая макеты, прототипы интерфейсов, юридические ограничения и т.д.;
2. **Группа анализа и проектирования** (analysis & design set) описывает, как система должна быть построена с учетом всех ограничений по времени, бюджету, унаследованным системам, повторному использованию, требованиям к качеству т.п.
3. **Группа тестирования** (test set) описывает подход, посредством которого система верифицируется и аттестуется. С этой целью используются скрипты, сценарии тестирования, способы измерения дефектов и критерии приемки;
4. **Группа реализации** (implementation set) дает информацию об элементах программного обеспечения, включая исходный код на разных языках программирования, файлы конфигурации, файлы данных и т.д., а также о сборке системы на основе разработанных программных компонентов;
5. **Группа размещения** (deployment set) представляет все данные, необходимые для конфигурирования поставляемой системы, – в частности, включает всю информацию о способах пакетирования, поставки, установки и запуска программного обеспечения на целевой платформе заказчика.

НЕДОСТАТКИ UML

- **Избыточность языка**. UML включает много избыточных или практически неиспользуемых диаграмм и конструкций.
- **Неточная семантика**. Так как UML определён комбинацией абстрактного синтаксиса, OCL (языком описания ограничений — формальной проверки правильности) и английского (подробная семантика), то возможны противоречия и неполнота описаний.
- **Проблемы при изучении и внедрении**.
- **Только код отражает код**. Существует потребность в лучшем способе написания ПО. UML не имеет свойств полноты по Тьюрингу и любой сгенерированный с его помощью код будет ограничен.
- **Кумулятивная нагрузка/Рассогласование нагрузки** (Cumulative Impedance/Impedance mismatch). UML может представить одни системы более кратко и эффективно, чем другие. Разработчик склоняется к решениям, которые более комфортно подходят к переплетению сильных сторон UML и языков программирования. Проблема возникает, если язык разработки не придерживается принципов ортодоксальной объектно-ориентированной доктрины.
- **Пытается быть всем для всех**. UML — это язык моделирования общего назначения, который пытается достигнуть совместимости со всеми возможными языками разработки. В контексте конкретного проекта, для достижения командой проектировщиков определённой цели, должны быть выбраны соответствующие возможности UML. Нет четких формальных ограничений на области применения UML в конкретной предметной области, существующие формализмы являются объектами критики.

КРИТИКА ООП

Представление «идеальной» разработки с использованием ООП критиками парадигмы