**Лекция 1**

**Аннотация**: UML - унифицированный язык моделирования. В предыдущей лекции было рассказано о том, что же такое UML, о его истории, назначении, способах использования языка, структуре его определения, терминологии и нотации. Было отмечено, что модель UML — это набор диаграмм. В этой лекции мы рассмотрим такие вопросы: почему нужно несколько видов диаграмм; виды диаграмм; ООП и последовательность построения диаграмм

Прежде чем перейти к обсуждению основного материала этой лекции, давайте поговорим о том, зачем вообще строить какие-то диаграммы. Разработка модели любой системы (не только программной) всегда предшествует ее созданию или обновлению. Это необходимо хотя бы для того, чтобы яснее представить себе решаемую задачу. Продуманные модели очень важны и для взаимодействия внутри команды разработчиков, и для взаимопонимания с заказчиком. В конце концов, это позволяет убедиться в "архитектурной согласованности" проекта до того, как он будет реализован в коде.

Мы строим модели сложных систем, потому что не можем описать их полностью, "окинуть одним взглядом". Поэтому мы выделяем лишь существенные для конкретной задачи свойства системы и строим ее модель, отображающую эти свойства. Метод объектно-ориентированного анализа позволяет описывать реальные сложные системы наиболее адекватным образом. Но с увеличением сложности систем возникает потребность в хорошей технологии моделирования. Как мы уже говорили в предыдущей лекции, в качестве такой "стандартной" технологии используется унифицированный язык моделирования (Unified Modeling Language, UML), который является графическим языком для спецификации, визуализации, проектирования и документирования систем. С помощью UML можно разработать подробную модель создаваемой системы, отображающую не только ее концепцию, но и конкретные особенности реализации. В рамках UML-модели все представления о системе фиксируются в виде специальных графических конструкций, получивших название диаграмм.

**Примечание**. Мы рассмотрим не все, а лишь некоторые из видов диаграмм. Например, диаграмма компонентов не рассматривается в этой лекции, которая является лишь кратким обзором видов диаграмм. Количество типов диаграмм для конкретной модели приложения никак не ограничивается. Для простых приложений нет необходимости строить диаграммы всех без исключения типов. Некоторые из них могут просто отсутствовать, и этот факт не будет считаться ошибкой. Важно понимать, что наличие диаграмм определенного вида зависит от специфики конкретного проекта. Информацию о других (не рассмотренных здесь) видах диаграмм можно найти в стандарте UML.

**Почему нужно несколько видов диаграмм**

Для начала определимся с терминологией. В предисловии к этой лекции мы неоднократно использовали понятия системы, модели и диаграммы. Автор уверен, что каждый из нас интуитивно понимает смысл этих понятий, но, чтобы внести полную ясность, снова заглянем в глоссарий и прочтем следующее:

**Система** - совокупность взаимосвязанных управляемых подсистем, объединенных общей целью функционирования.

Да, не слишком информативно. А что же такое тогда подсистема? Чтобы прояснить ситуацию, обратимся к классикам:

**Системой** называют набор подсистем, организованных для достижения определенной цели и описываемых с помощью совокупности моделей, возможно, с различных точек зрения.

Что ж, ничего не попишешь, придется искать определение подсистемы. Там же сказано, что **подсистема** — это совокупность элементов, часть из которых задает спецификацию поведения других элементов. Ян Соммервилл объясняет это понятие таким образом:

**Подсистема** — это система, функционирование которой не зависит от сервисов других подсистем. Программная система структурируется в виде совокупности относительно независимых подсистем. Также определяются взаимодействия между подсистемами.

Тоже не слишком понятно, но уже лучше. Говоря "человеческим" языком, система представляется в виде набора более простых сущностей, которые относительно самодостаточны. Это можно сравнить с тем, как в процессе разработки программы мы строим графический интерфейс из стандартных "кубиков" - визуальных компонентов, или как сам текст программы тоже разбивается на модули, которые содержат подпрограммы, объединенные по функциональному признаку, и их можно использовать повторно, в следующих программах.

С понятием системы разобрались. В процессе проектирования система рассматривается **с разных точек зрения** с помощью моделей, различные представления которых предстают в форме диаграмм. Опять-таки у читателя могут возникнуть вопросы о смысле понятий *модели* и *диаграммы*. Думаем, красивое, но не слишком понятное определение **модели как семантически замкнутой абстракции системы** вряд ли прояснит ситуацию, поэтому попробуем объяснить "своими словами".

**Модель** — это некий (материальный или нет) объект, отображающий лишь наиболее значимые для данной задачи характеристики системы. Модели бывают разные - материальные и нематериальные, искусственные и естественные, декоративные и математические...

Приведем несколько примеров. Знакомые всем нам пластмассовые игрушечные автомобильчики, которыми мы с таким азартом играли в детстве, это не что иное, как *материальная искусственная декоративная* модель реального автомобиля. Конечно, в таком "авто" нет двигателя, мы не заполняем его бак бензином, в нем не работает (более того, вообще отсутствует) коробка передач, но как модель эта игрушка свои функции вполне выполняет: она дает ребенку представление об автомобиле, поскольку отображает его характерные черты - наличие четырех колес, кузова, дверей, окон, способность ехать и т. д.

В ходе медицинских исследований опыты на животных часто предшествуют клиническим испытаниям медицинских препаратов на людях. В таком случае животное выступает в роли *материальной естественной* модели человека.



Уравнение, изображенное выше - тоже модель, но это модель математическая, и описывает она движение материальной точки под действием силы тяжести.

Осталось лишь сказать, что такое диаграмма. **Диаграмма** — это графическое представление множества элементов. Обычно изображается в виде графа с вершинами (сущностями) и ребрами (отношениями). Примеров диаграмм можно привести множество. Это и знакомая нам всем со школьных лет блок-схема, и схемы монтажа различного оборудования, которые мы можем видеть в руководствах пользователя, и дерево файлов и каталогов на диске, которое мы можем увидеть, выполнив в консоли Windows команду tree, и многое-многое другое. В повседневной жизни диаграммы окружают нас со всех сторон, ведь рисунок воспринимается нами легче, чем текст...

Но вернемся к проектированию ПО (и не только). В этой отрасли с **помощью диаграмм можно визуализировать систему с различных точек зрения**. Одна из диаграмм, например, может описывать взаимодействие пользователя с системой, другая - изменение состояний системы в процессе ее работы, третья - взаимодействие между собой элементов системы и т. д. Сложную систему можно и нужно представить в виде набора небольших и почти независимых моделей-диаграмм, причем ни одна из них не является достаточной для описания системы и получения полного представления о ней, поскольку каждая из них фокусируется на каком-то определенном аспекте функционирования системы и выражает разный уровень абстракции. Другими словами, каждая модель соответствует некоторой определенной, частной точке зрения на проектируемую систему.

Несмотря на то, что в предыдущем абзаце мы весьма вольготно обошлись с понятием модели, следует понимать, что в контексте приведенных выше определений **ни одна отдельная диаграмма не является моделью**. Диаграммы - лишь средство визуализации модели, и эти два понятия следует различать. Лишь **набор диаграмм составляет модель системы** и наиболее полно ее описывает, но не одна диаграмма, вырванная из контекста.

**Виды диаграмм**

UML 1.5 определял **двенадцать типов диаграмм**, разделенных на три группы:

* четыре типа диаграмм представляют статическую структуру приложения;
* пять представляют поведенческие аспекты системы;
* три представляют физические аспекты функционирования системы (диаграммы реализации).

Текущая версия UML 2.1 внесла не слишком много изменений. Диаграммы слегка изменились внешне (появились фреймы и другие визуальные улучшения), немного усовершенствовалась нотация, некоторые диаграммы получили новые наименования.

Впрочем, точное число канонических диаграмм для нас абсолютно неважно, так как мы рассмотрим не все из них, а лишь некоторые - по той причине, что количество типов диаграмм для конкретной модели конкретного приложения не является строго фиксированным. Для простых приложений нет необходимости строить все без исключения диаграммы. Например, для локального приложения не обязательно строить диаграмму развертывания. Важно понимать, что перечень диаграмм зависит от специфики разрабатываемого проекта и определяется самим разработчиком. Если же любопытный читатель все-таки пожелает узнать обо всех диаграммах UML, мы отошлем его к стандарту UML (<http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML>). Напомним, что цель этого курса - не описать абсолютно все возможности UML, а лишь познакомить с этим языком, дать первоначальное представление об этой технологии.

Итак, мы кратко рассмотрим такие виды диаграмм, как:

* диаграмма прецедентов;
* диаграмма классов;
* диаграмма объектов;
* диаграмма последовательностей;
* диаграмма взаимодействия;
* диаграмма состояний;
* диаграмма активности;
* диаграмма развертывания.

О некоторых из этих диаграмм мы будем говорить подробнее в следующих лекциях. Пока же мы не станем заострять внимание на подробностях, а зададимся целью научить читателя хотя бы визуально различать виды диаграмм, дать начальное представление о назначении основных видов диаграмм. Итак, начнем.

**Диаграмма прецедентов (use case diagram)**

Любые (в том числе и программные) системы проектируются с учетом того, что в процессе своей работы они будут использоваться людьми и/или взаимодействовать с другими системами. Сущности, с которыми взаимодействует система в процессе своей работы, называются **экторами**, причем каждый эктор ожидает, что система будет вести себя строго определенным, предсказуемым образом. Попробуем дать более строгое определение эктора. Для этого воспользуемся замечательным визуальным словарем по UML *Zicom Mentor*:

**Эктор (actor)** — это множество логически связанных ролей, исполняемых при взаимодействии с прецедентами или сущностями (система, подсистема или класс). Эктором может быть человек или другая система, подсистема или класс, которые представляют нечто вне сущности.

Графически эктор изображается либо " *человечком* ", подобным тем, которые мы рисовали в детстве, изображая членов своей семьи, либо *символом класса с соответствующим стереотипом*, как показано на рисунке. Обе формы представления имеют один и тот же смысл и могут использоваться в диаграммах. "Стереотипированная" форма чаще применяется для представления системных экторов или в случаях, когда эктор имеет свойства и их нужно отобразить ([рис. 2.1](https://intuit.ru/studies/courses/1007/229/lecture/5954?page=1#image.2.1)).

Внимательный читатель сразу же может задать вопрос: *а почему эктор, а не актер*? Согласны, слово "эктор" немного режет слух русского человека. Причина же, почему мы говорим именно так, проста - эктор образовано от слова **action**, что в переводе означает *действие*. Дословный же перевод слова "эктор" - *действующее лицо* - слишком длинный и неудобный для употребления. Поэтому мы будем и далее говорить именно так.



**Рис. 2.1.**

Тот же внимательный читатель мог заметить промелькнувшее в определении эктора слово "прецедент". Что же это такое? Этот вопрос заинтересует нас еще больше, если вспомнить, что сейчас мы говорим о *диаграмме прецедентов*. Итак,

**Прецедент (use-case)** - описание отдельного аспекта поведения системы с точки зрения пользователя (Буч).

Определение вполне понятное и исчерпывающее, но его можно еще немного уточнить, воспользовавшись тем же *Zicom Mentor* 'ом:

**Прецедент (use case)** - описание множества последовательных событий (включая варианты), выполняемых системой, которые приводят к наблюдаемому эктором результату. Прецедент представляет поведение сущности, описывая взаимодействие между экторами и системой. Прецедент не показывает, "как" достигается некоторый результат, а только "что" именно выполняется.

Прецеденты обозначаются очень простым образом - в виде эллипса, внутри которого указано его название. *Прецеденты и экторы соединяются с помощью линий*. Часто на одном из концов линии изображают *стрелку*, причем *направлена она к тому, у кого запрашивают сервис*, другими словами, чьими услугами пользуются. Это простое объяснение иллюстрирует *понимание прецедентов как сервисов*, пропагандируемое компанией IBM.



**Рис. 2.2.**

Прецеденты могут включать другие прецеденты, расширяться ими, наследоваться и т. д. *Все эти возможности мы здесь рассматривать не будем*. Как уже говорилось выше, цель этого обзора - просто научить читателя выделять диаграмму прецедентов, понимать ее назначение и смысл обозначений, которые на ней встречаются.

Кстати, к этому моменту мы уже потратили достаточно много времени на объяснение понятий и их условных обозначений. Наверное, пора уже, наконец, привести пример диаграммы прецедентов. Как вы думаете, что означает эта диаграмма ([рис. 2.3](https://intuit.ru/studies/courses/1007/229/lecture/5954?page=1#image.2.3))?



**Рис. 2.3.**

Полагаем, здесь все было бы понятно, если бы даже мы никогда не слышали о диаграммах прецедентов! Ведь так? Все мы в студенческие годы пользовались библиотеками (которые теперь для нас заменил Интернет), и потому все это для нас знакомо. Обратите также внимание на примечание, сопоставленное с одним из прецедентов. Следует заметить, что иногда на диаграммах прецедентов *границы системы обозначают прямоугольником*, в верхней части которого может быть указано *название системы*. Таким образом, прецеденты - действия, выполняемые системой в ответ на действия эктора, - помещаются внутри прямоугольника.

А вот еще один пример ([рис. 2.4](https://intuit.ru/studies/courses/1007/229/lecture/5954?page=1#image.2.4)). Думаем, вы сами, без нашей помощи, легко догадаетесь, о чем там идет речь.



**Рис. 2.4.**

Из всего сказанного выше становится понятно, что *диаграммы прецедентов* относятся к той группе диаграмм, которые представляют динамические или поведенческие аспекты системы. Это отличное *средство для достижения взаимопонимания между разработчиками, экспертами и конечными пользователями* продукта. Как мы уже могли убедиться, такие диаграммы очень просты для понимания и могут восприниматься и, что немаловажно, обсуждаться людьми, не являющимися специалистами в области разработки ПО.

Подводя итоги, можно выделить такие **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.