Rational Unified Process

(курс лекций)

Разработчик: ассистент каф. ПМИ
Бондаренко Иван Юрьевич

Жизненный цикл ПО

Одним из базовых понятий методологии проектирования ИС является понятие **жизненного цикла** ее программного обеспечения (**ЖЦ ПО**).

ЖЦ ПО - это непрерывный процесс, который начинается с момента принятия решения о необходимости его создания и заканчивается в момент его полного изъятия из эксплуатации.

Стандартизация жизненного цикла

Основным нормативным документом, регламентирующим ЖЦ ПО, является **международный стандарт ISO/IEC 12207** (ISO - International Organization of Standardization - Международная организация по стандартизации, IEC - International Electrotechnical Commission - Международная комиссия по электротехнике). Он определяет структуру ЖЦ, содержащую процессы, действия и задачи, которые должны быть выполнены во время создания ПО.

Структура жизненного цикла

Структура ЖЦ ПО по стандарту ISO/IEC 12207 базируется на трех группах процессов:

- **основные процессы** (приобретение, поставка, разработка, эксплуатация, сопровождение);
- **вспомогательные процессы**, которые обеспечивают выполнение основных процессов (документирование, управление конфигурацией, обеспечение качества, верификация, аттестация, оценка, аудит, решение проблем);
- **организационные процессы** (управление проектами, создание инфраструктуры проекта, определение, оценка и улучшение самого ЖЦ, обучение).

Основной процесс - разработка ПО

Все работы по созданию ПО и его компонент в соответствии с заданными требованиями, включая оформление проектной и эксплуатационной документации, подготовку материалов, необходимых для проверки работоспособности и соответствующего качества программных продуктов, материалов, необходимых для организации обучения персонала и т.д.

Разработка ПО включает в себя, как правило, анализ, проектирование и реализацию (программирование).

Основной процесс - эксплуатация

Работы по внедрению компонентов ПО в эксплуатацию, в том числе конфигуриро-вание базы данных и рабочих мест пользователей, обеспечение эксплуатаци-онной документацией, проведение обучения персонала и т.д., и непосред-ственно эксплуатацию, в том числе локализацию проблем и устранение причин их возникновения, модификацию ПО в рамках установленного регламента, подготовку предложений по совершенствованию, развитию и модернизации системы.

Вспомогательный процесс – контроль качества

Проблемы верификации, проверки и тестирования ПО.

Верификация - это процесс определения того, отвечает ли текущее состояние разработки, достигнутое на данном этапе, требованиям этого этапа.

Проверка позволяет оценить соответствие параметров разработки с исходными требованиями. Проверка частично совпадает с тестированием.

Тестирование связано с идентификацией различий между действительными и ожидаемыми результатами и оценкой соответствия характеристик ПО исходным требованиям.

Организационный процесс – управление проектом

Вопросы планирования и организации работ, создания коллективов разработчиков и контроля за сроками и качеством выполняемых работ. Техническое и организационное обеспечение проекта включает выбор методов и инструментальных средств для реализации проекта, определение методов описания промежуточных состояний разработки, разработку методов и средств испытаний ПО, обучение персонала и т.п.

Организационный процесс – управление конфигурацией

Поддерживает основные процессы жизненного цикла ПО, прежде всего процессы разработки и сопровождения ПО.

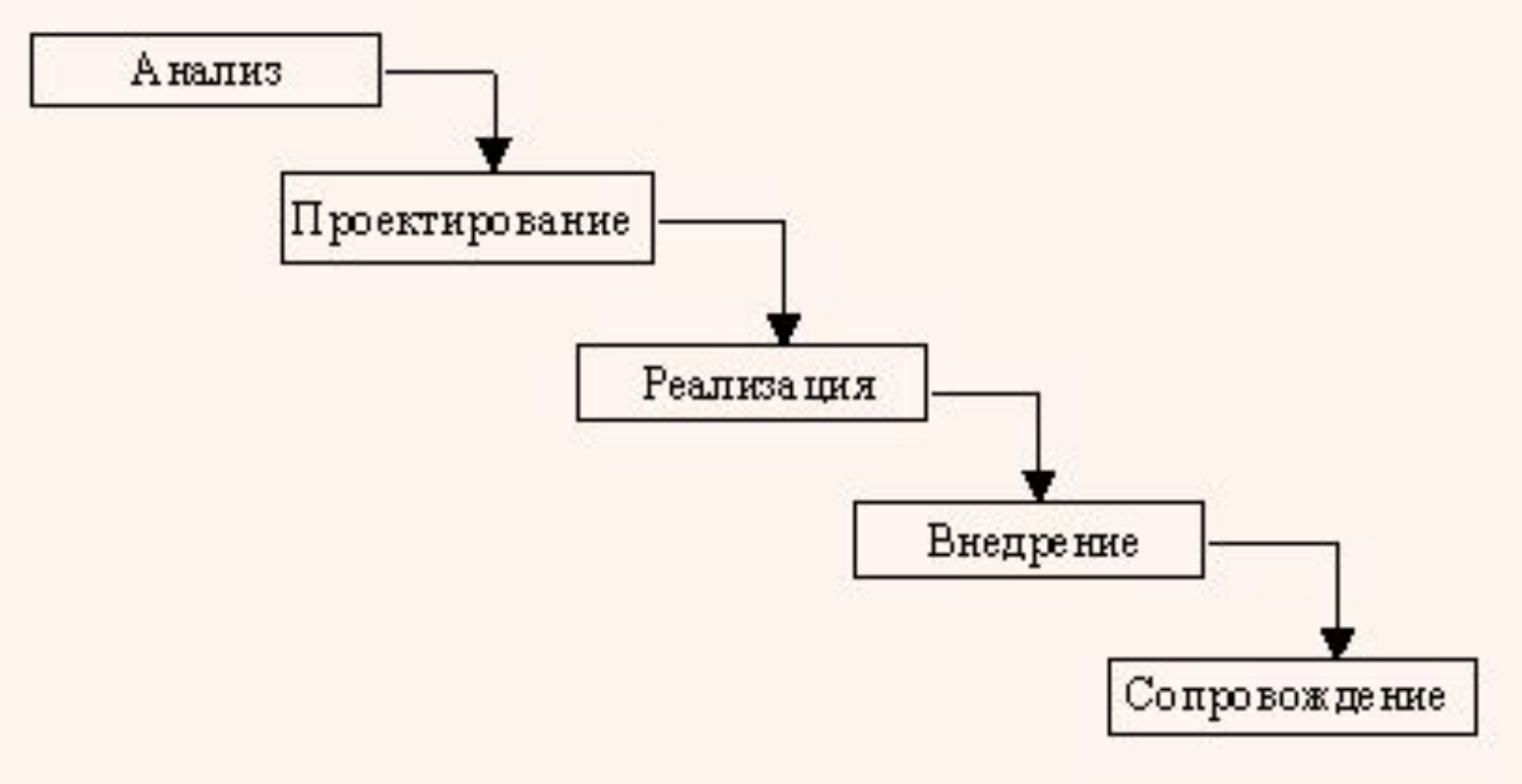
При создании проектов сложных ИС, состоящих из многих компонентов, каждый из которых может иметь разновидности или версии, возникает проблема учета их связей и функций, создания унифицированной структуры и обеспечения развития всей системы.

Управление конфигурацией позволяет организовать, систематически учитывать и контролировать внесение изменений в ПО на всех стадиях ЖЦ.

Каскадная модель жизненного цикла

Разбиение всей разработки на этапы, причем переход с одного этапа на следующий происходит только после того, как будет полностью завершена работа на текущем. Каждый этап завершается выпуском полного комплекта документации, достаточной для того, чтобы разработка могла быть продолжена другой командой разработчиков.

Каскадная модель жизненного цикла



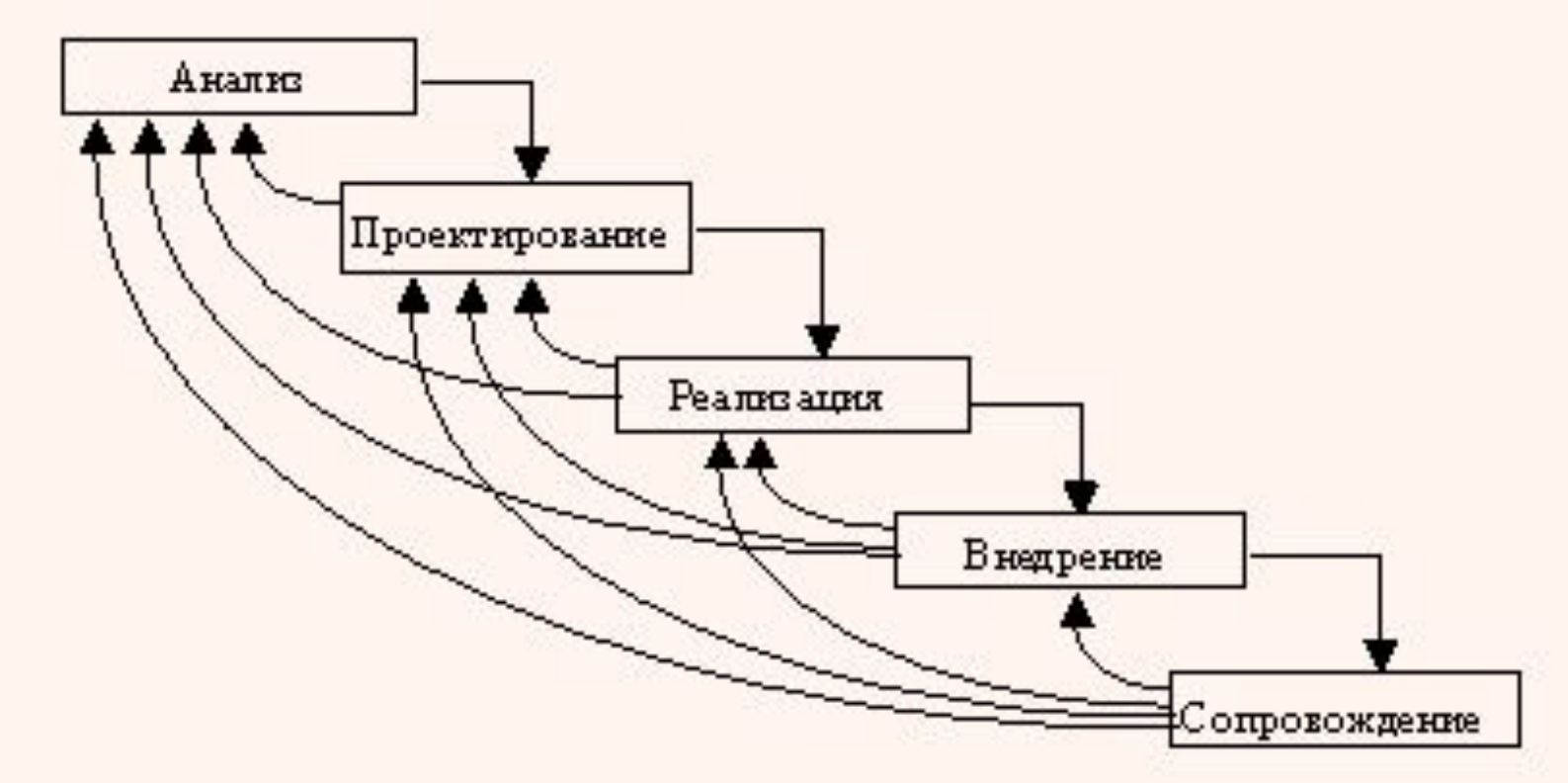
Каскадная модель - преимущества

- на каждом этапе формируется законченный набор проектной документации, отвечающий критериям полноты и согласованности;
- выполняемые в логичной последовательности этапы работ позволяют планировать сроки завершения всех работ и соответствующие затраты.

Каскадная модель - недостатки

- пригодна только для таких проектов, для которых в самом начале разработки можно достаточно точно и полно сформулировать все требования ;
- существенное запаздывание с получением результатов (согласование результатов с пользователями производится только в точках, планируемых после завершения каждого этапа работ, требования к ИС "заморожены" в виде технического задания на все время ее создания).

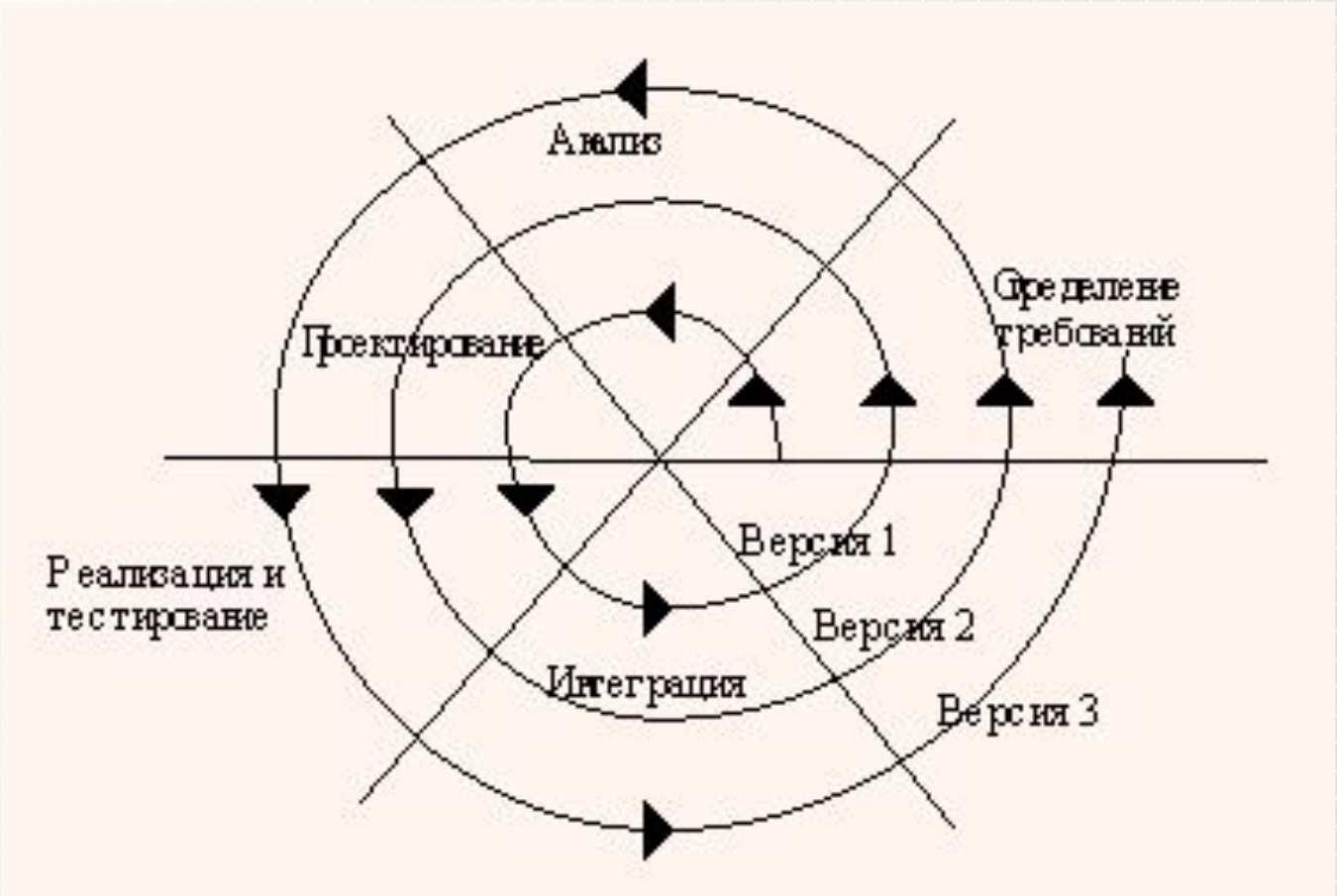
Во что превращается каскадная модель в реальности



Спиральная модель жизненного цикла

Делает упор на начальные этапы ЖЦ: анализ и проектирование. На этих этапах реализуемость технических решений проверяется путем создания прототипов. Каждый виток спирали соответствует созданию фрагмента или версии ПО, на нем уточняются цели и характеристики проекта, определяется его качество и планируются работы следующего витка спирали. Таким образом углубляются и последовательно конкретизируются детали проекта и в результате выбирается обоснованный вариант, который доводится до реализации.

Спиральная модель жизненного цикла



Спиральная модель - достоинства

- Разработка итерациями отражает объективно существующий спиральный цикл создания системы.
- Неполное завершение работ на каждом этапе позволяет переходить на следующий этап, не дожидаясь полного завершения работы на текущем.
- Недостающую работу можно будет выполнить на следующей итерации. Главная же задача - как можно быстрее показать пользователям системы работоспособный продукт, тем самым активизируя процесс уточнения и дополнения требований.

Спиральная модель - недостатки

Основной недостаток спирального цикла – проблема определение момента перехода на следующий этап.

Для ее решения необходимо ввести временные ограничения на каждый из этапов жизненного цикла. Переход осуществляется в соответствии с планом, даже если не вся запланированная работа закончена. План составляется на основе статистических данных, полученных в предыдущих проектах, и личного опыта разработчиков.

Rational Unified Process

Rational Unified Process (RUP, рациональный унифицированный процесс) – это один из вариантов итеративной модели жизненного цикла ПО.

RUP – это "тяжелый" процесс, детально описанный и предполагающий поддержку собственно разработки исходного кода ПО большим количеством вспомогательных дей-ствий. Это позволяет отделить успешные прак-тики разработки и сопровождения ПО от кон-кретных людей, умеющих их применять, и сде-лать возможным решение задач по конструи-рованию и поддержке сложных систем с помо-щью имеющихся работников, не обязательно являющихся суперпрофессионалами.

История RUP

Исторически RUP является развитием модели процесса разработки, принятой в компании Ericsson в 70–80-х годах XX века. Эта модель была создана Джекобсоном (Ivar Jacobson), впоследствии, в 1987 году, основавшим компанию Objectory AB именно для развития технологического процесса разработки ПО как отдельного продукта, который можно было бы переносить в другие организации. После вливания Objectory в Rational в 1995 году разработки Джекобсона были интегрированы с работами Ройса (Walker Royce, сын автора "классической" каскадной модели), Крухтена (Philippe Kruchten) и Буча (Grady Booch), а также с развивавшимся параллельно **универсальным языком моделирования (Unified Modeling Language, UML)**.

«Три кита» RUP

1. Весь ход работ направляется итоговыми целями проекта, выраженными в виде **вариантов использования (use cases)** — сценариев взаимодействия результирующей программной системы с пользователями или другими системами, при выполнении которых пользователи получают значимые для них результаты и услуги. Разработка начинается с выделения вариантов использования и на каждом шаге контролируется степенью приближения к их реализации.

«Три кита» RUP

2. Основным решением, принимаемым в ходе проекта, является архитектура результирующей программной системы. Архитектура устанавливает набор компонентов, из которых будет построено ПО, ответственность каждого из компонентов (т.е. решаемые им подзадачи в рамках общих задач системы), четко определяет интерфейсы, через которые они могут взаимодействовать, а также способы взаимодействия компонентов друг с другом.

«Три кита» RUP

3. Основой процесса разработки являются **планируемые** и **управляемые итерации**, объем которых (реализуемая в рамках итерации функциональность и набор компонентов) определяется на основе архитектуры.

Описание требований к программной системе – диаграмма вариантов использования

Варианты использования – это методика формирования требований, основанная на сценариях. Этот вид диаграмм позволяет создать список операций, которые выполняет система. Часто этот вид диаграмм называют диаграммой функций, потому что на основе набора таких диаграмм создается список требований к системе и определяется множество выполняемых системой функций.

В самой простой форме в вариантах использования определены действующие лица – актеры и возможные взаимодействия – прецеденты.

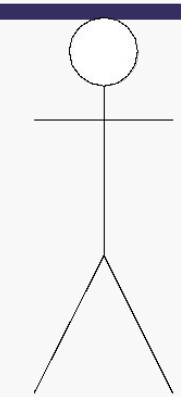
24

Диаграмма вариантов использования

- Диаграммы вариантов использования описывают взаимоотношения и зависимости между группами *вариантов использования (прецедентами)* и *действующих лиц (актерами)*, участвующими в процессе.

25

Диаграмма вариантов использования. Условные обозначения



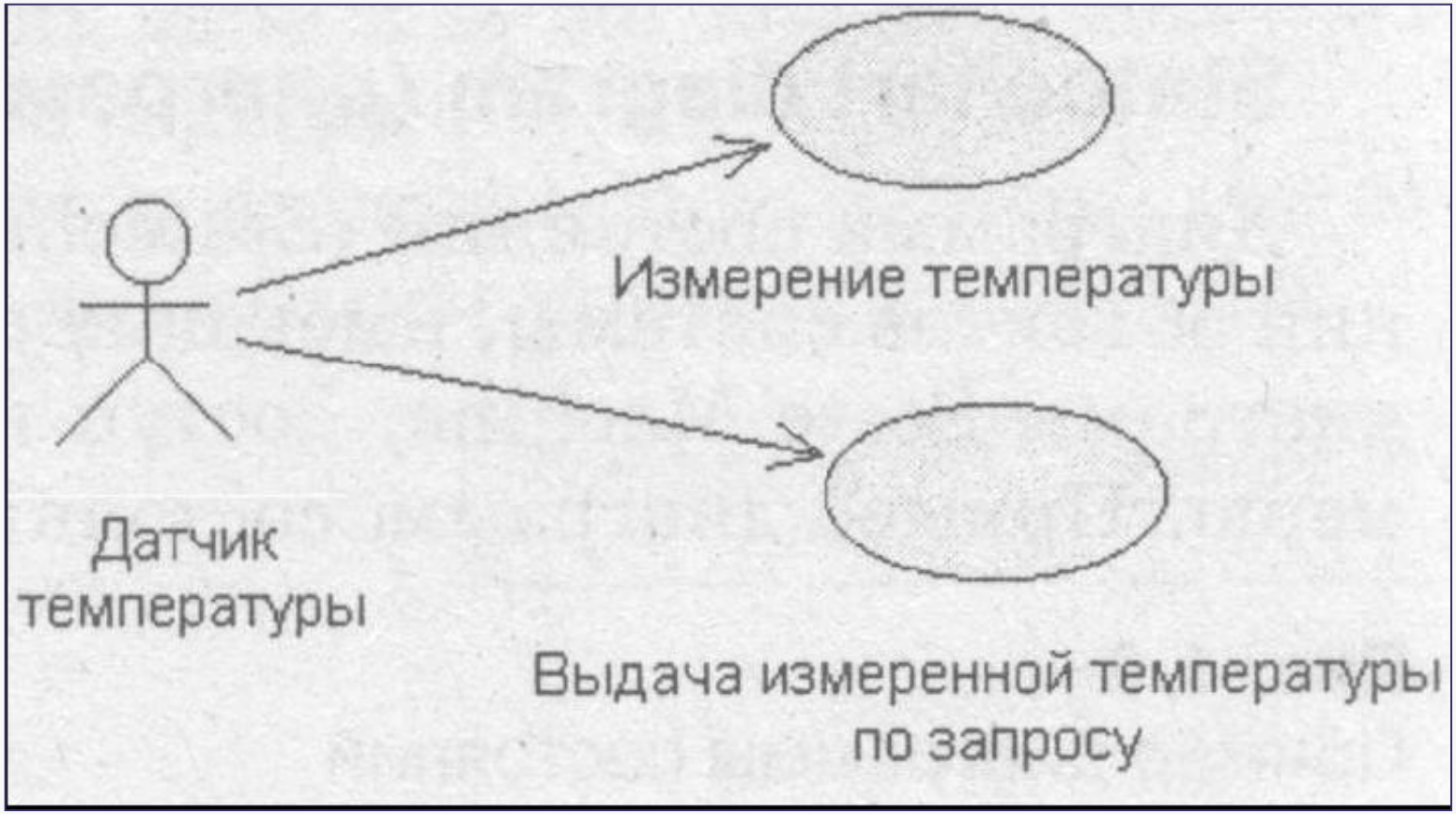
Актер

Пользователи программной системы и внешняя среда, которая взаимодействует с программной системой

Вариант сценария работы программной системы

26

Диаграмма вариантов использования. Пример



27

Правила построения диаграммы

При работе с вариантами использования важно помнить несколько простых правил:

- каждый вариант использования относится как минимум к одному действующему лицу;
- каждый вариант использования имеет инициатора;
- каждый вариант использования приводит к соответствующему результату.

28

Связи между прецедентами в диаграмме вариантов использования

Варианты использования также могут взаимодействовать с другими вариантами использования. Три наиболее часто встречающихся типа взаимодействия между вариантами использования приведены ниже:

- *<<включение>> указывает, что вариант использования встраивается в другой вариант использования;*

29

Связи между прецедентами в диаграмме вариантов использования

- <<добавление>> (<<расширение>>) указывает, что в определённых ситуациях или в некоторой точке (называемой точкой расширения) вариант использования будет расширен другим;

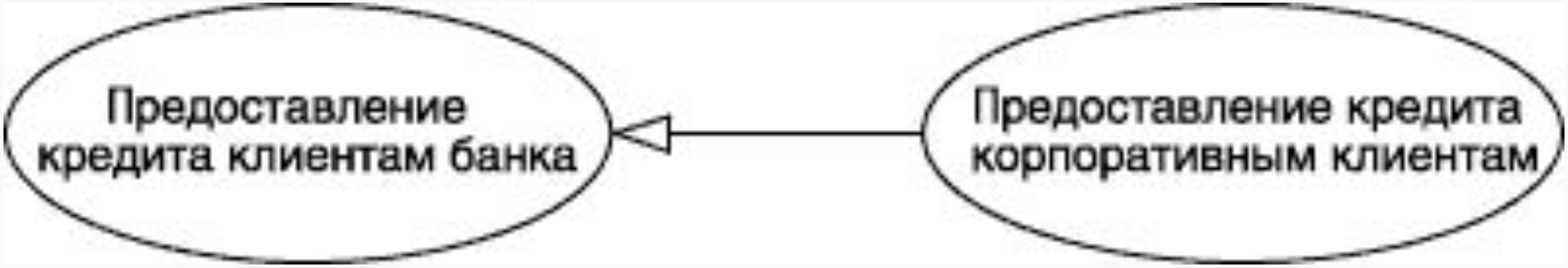
- <<обобщение>> (<<наследование>>) указывает, что вариант использования наследует характеристики «родительского» варианта использования и может переопределить некоторые из них или добавить новые, подобно наследованию в классах.

30

Связи между прецедентами в диаграмме вариантов использования

31

Связи между прецедентами в диаграмме вариантов использования



Следует подчеркнуть, что потомок наследует все свойства поведения своего родителя, а также может обладать дополнительными особенностями поведения.

32

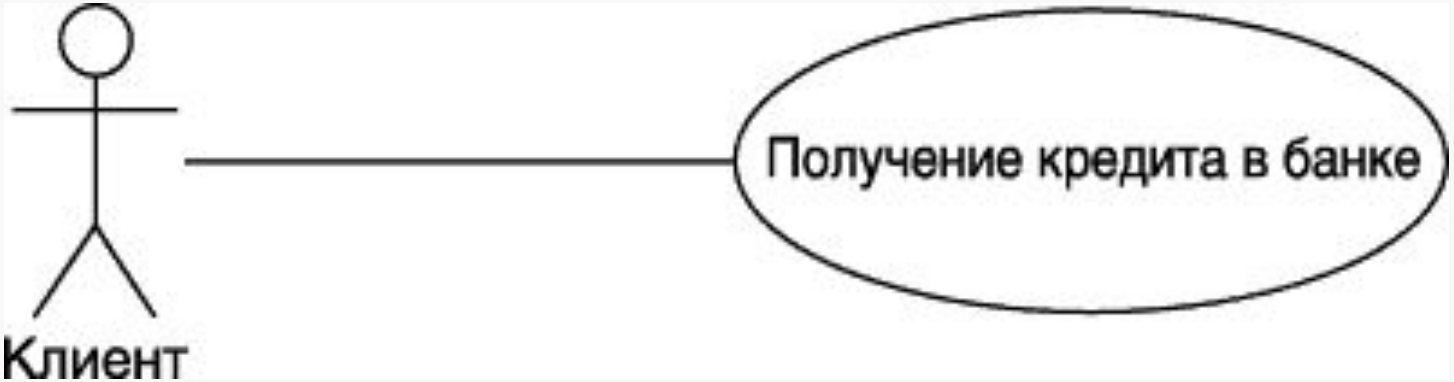
Связи между актером и прецедентом

Отношение <<ассоциации>> – одно из фундаментальных понятий в языке UML и в той или иной степени используется при построении всех графических моделей систем в форме канонических диаграмм.

Применительно к диаграммам вариантов использования <<ассоциация>> служит для обозначения специфической роли актера при его взаимодействии с отдельным вариантом использования. Другими словами, <<ассоциация>> специфицирует семантические особенности взаимодействия актеров и вариантов использования в графической модели системы. **Других связей между актером и прецедентом быть не может!**

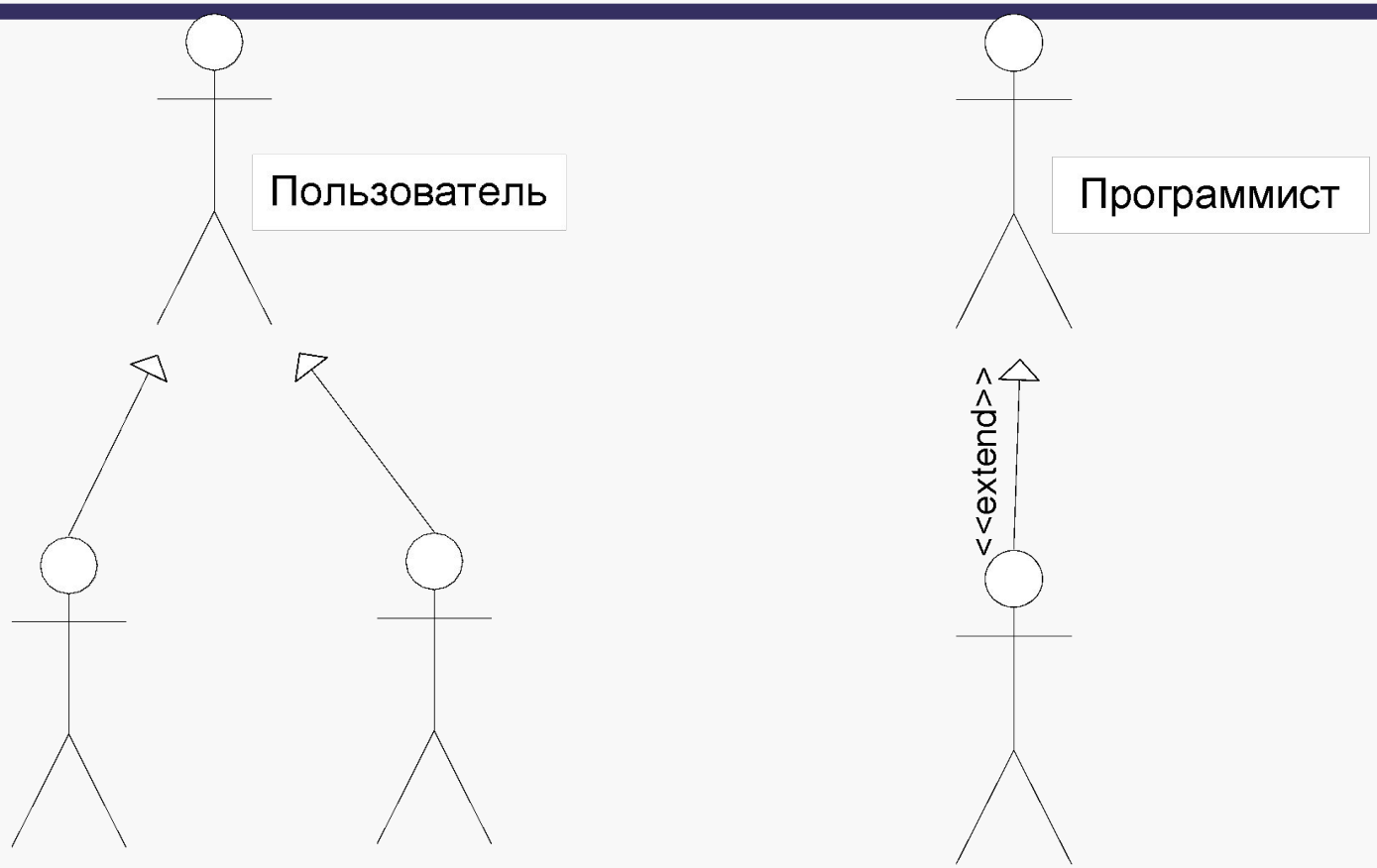
33

Связи между актером и прецедентом



34

Связи между актерами – расширение или наследование



35

Описание требований к системе дистанционного обучения

- Актеры: студент, преподаватель, администратор системы, база данных
- Прецеденты:
  - студент: аутентификация, просмотр лекций, выполнение заданий, тестирование;
  - преподаватель: аутентификация, внесение новой информации, составление тестов, просмотр результатов тестирования студентов;

36

Описание требований к системе дистанционного обучения

– администратор: поддержка системы.

37

Диаграмма вариантов использования