Программная инженерия

UML. UP. Определение требований.

Вопросы

1. В список требований к разрабатываемой системе вкралась ошибочная информация. Стоимость ее исправления на этапе определения требований составляет порядка 200$ (с учетом времени работы экспертов). Сумма какого порядка может потребоваться на исправление этой ошибки на этапе тестирования? Внедрения?
2. Процесс тестирования состоит из 3-х основных видов тестирования. Назовите их. Какие ошибки выявляются на каждом из этапов тестирования?

Вопросы

3. Почему список требований к программному продукту и его общая спецификация так важны на финальном этапе разработки и внедрения?
4. Модель зрелости программного обеспечения определяет 5 уровней зрелости. Какой ключевой процесс необходимо обеспечить для перехода на 4-й уровень (Управляемый). Как это связано с метриками ПО?
5. Назовите основные недостатки применения размерно-ориентированных метрик ПО (SLOC, LSI и др.)

Унифицированный язык моделирования (Unified Modeling Language, UML)

Что такое UML?

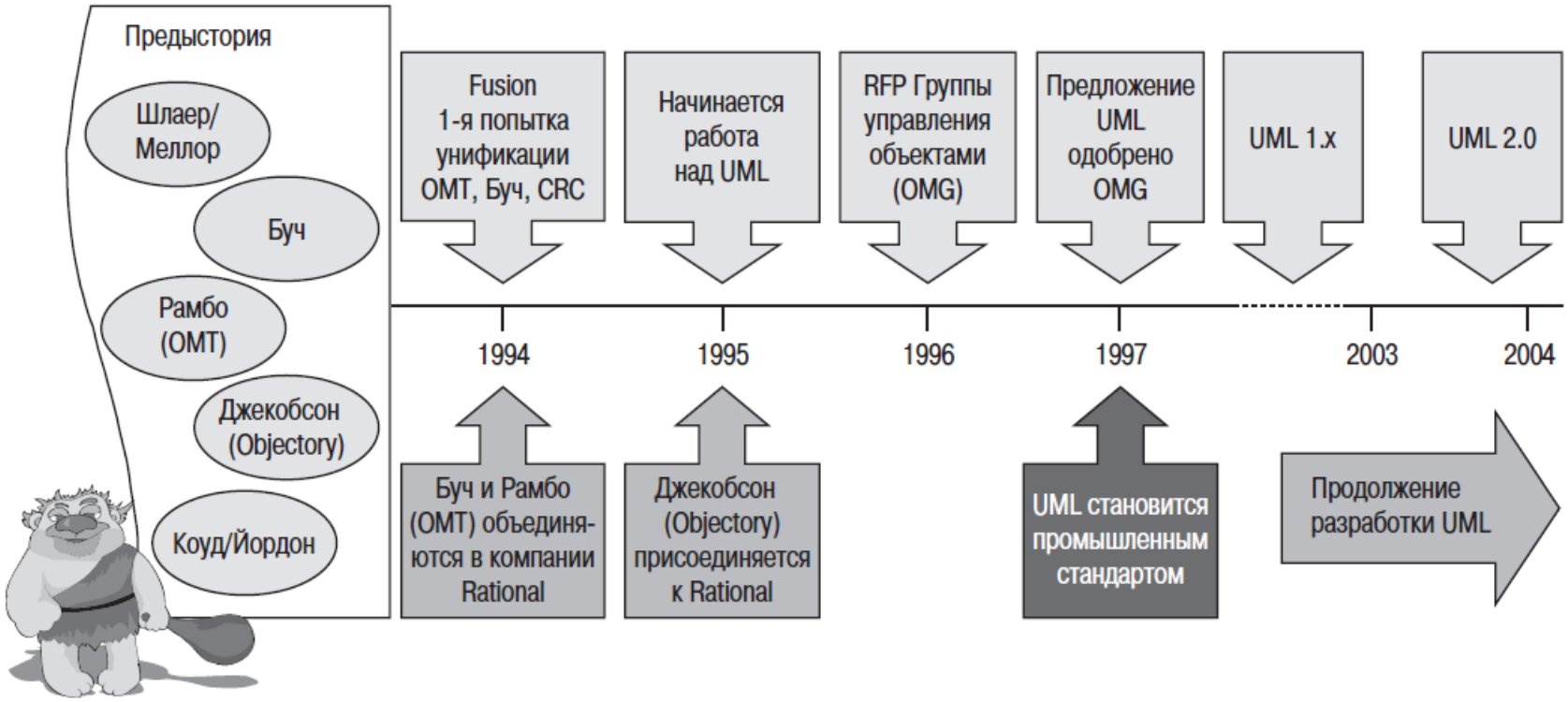
- **UML (Unified Modeling Language)** - это универсальный язык визуального моделирования систем

- **UML это не методология**
  - это унифицированный язык визуального моделирования.

Унифицированный процесс

- **Унифицированный процесс (Unified Process, UP)** – это методология моделирования программных систем.
- Она указывает на исполнителей, действия и артефакты, которые необходимо использовать, осуществить или создать для моделирования программной системы.

История UML


Предыстория
Шлаер/
Меллор
Буч
Рамбо
(OMT)
Джекобсон
(Objectory)
Коуд/Йордон
Fusion
1-я попытка
унификации
OMT, Буч, CRC
Начинается
работа
над UML
RFP Группы
управления
объектами
(OMG)
Предложение
UML
одобрено
OMG
UML 1.x
UML 2.0
1994
1995
1996
1997
2003
2004
Буч и Рамбо
(OMT) объединя-
ются в компании
Rational
Джекобсон
(Objectory)
присоединяется
к Rational
UML становится
промышленным
стандартом
Продолжение
разработки UML

Унификация UML

- **Жизненный цикл разработки:** UML предоставляет визуальный синтаксис для моделирования на протяжении всего жизненного цикла разработки программного обеспечения – от постановки требований до реализации.
- **Области приложений:** UML используется для моделирования всех аспектов – от аппаратных встроенных систем реального времени до систем поддержки принятия решений.
- **Языки реализации и платформы:** UML является независимым от языков и платформ. Естественно, он прекрасно поддерживает чистые ОО языки (Smalltalk, Java, C# и др.), но также эффективен и для гибридных ОО языков, таких как C++, и основанных на концепции объектов, таких как Visual Basic. UML также используется для создания моделей, реализуемых на неОО языках программирования, таких как C.

Объекты и UML

- UML моделирует мир как **системы взаимодействующих объектов**.
- **Объект** – это цельный блок, состоящий из данных и функциональности.
- В UML-модели есть два аспекта:
  - **Статическая структура** – описывает, какие типы объектов важны для моделирования системы и как они взаимосвязаны.
  - **Динамическое поведение** – описывает жизненные циклы этих объектов и то, как они взаимодействуют друг с другом для обеспечения требуемой функциональности системы.

Строительные блоки UML

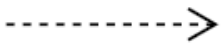
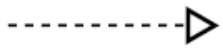
- **Сущности** – это сами элементы модели.
- **Отношения** связывают сущности. Отношения определяют, как семантически связаны две или более сущностей.
- **Диаграммы** – это представления моделей UML. Они показывают наборы сущностей, которые «рассказывают» о программной системе и являются нашим способом визуализации того
  - **что будет делать система** (аналитические диаграммы) или
  - **как она будет делать это** (проектные диаграммы).

Сущности

Все UML-сущности можно разделить на:

- **структурные сущности** – существительные UML-модели, такие как класс, интерфейс, кооперация, прецедент, активный класс, компонент, узел;
- **поведенческие сущности** – глаголы UML-модели, такие как взаимодействия, деятельности, автоматы;
- **группирующая сущность** – пакет, используемый для группировки семантически связанных элементов модели в образующие единое целое модули;
- **аннотационная сущность** – примечание, которое может быть добавлено к модели для записи специальной информации, очень похожее на стикер.

Отношения

Тип отношения	UML-синтаксис (источник / цель)	Краткая семантика
Зависимость		Исходный элемент зависит от целевого элемента и изменение последнего может повлиять на первый.
Ассоциация		Описание набора связей между объектами.
Агрегация		Целевой элемент является частью исходного элемента.
Композиция		Строгая (более ограниченная) форма агрегирования.
Включение		Исходный элемент содержит целевой элемент.
Обобщение		Исходный элемент является специализацией более обобщенного целевого элемента и может замещать его.
Реализация		Исходный элемент гарантированно выполняет контракт, определенный целевым элементом.

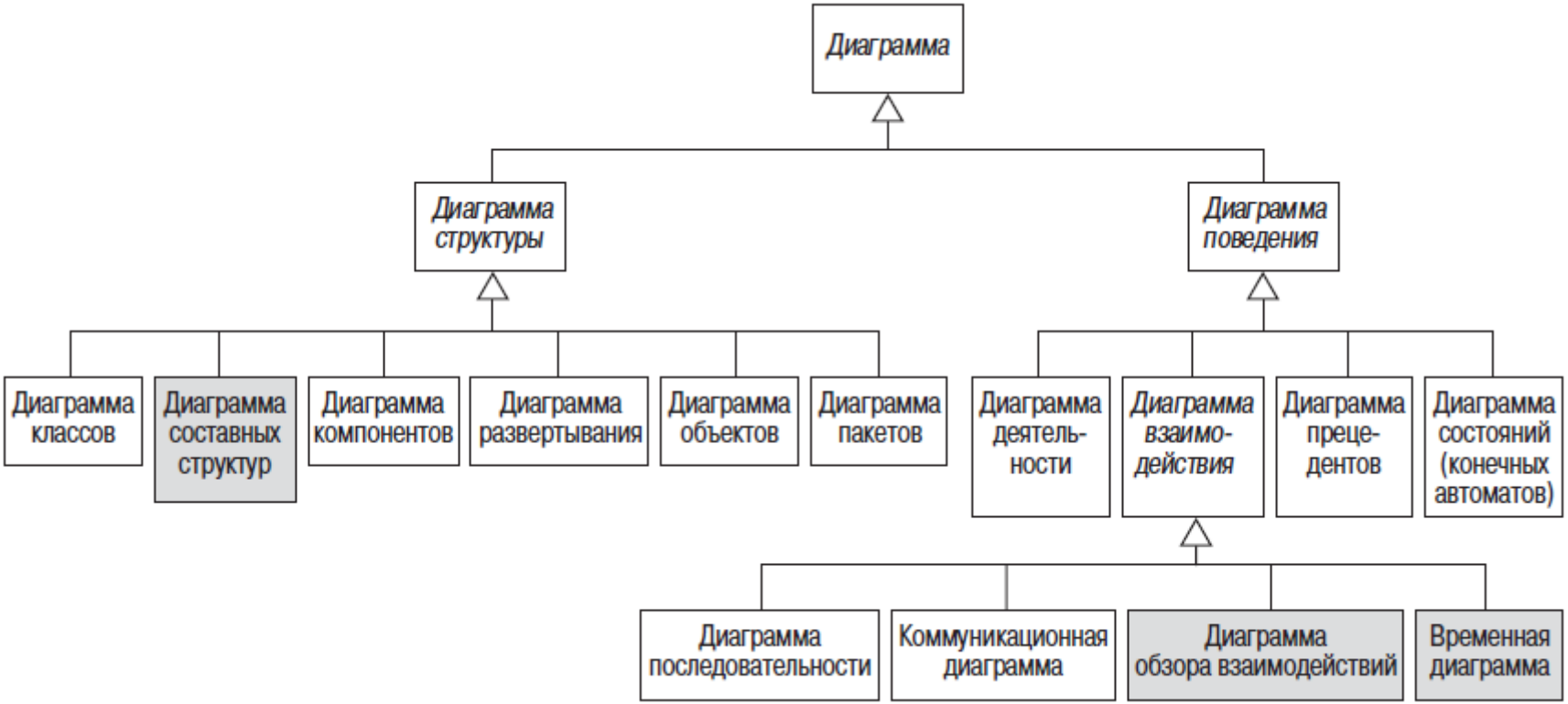
Диаграмма

- **Диаграммы** – это своего рода *картины*, или *представления* модели.

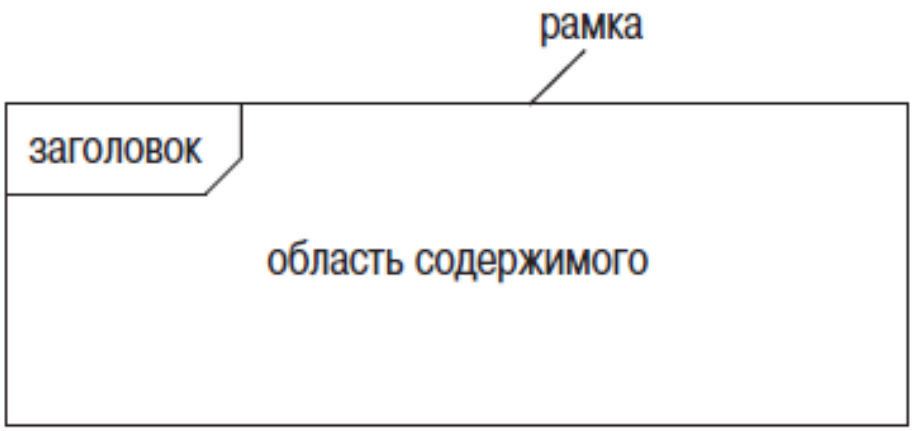
Диаграмма это *не модель!*

- Сущность или отношение могут быть **удалены с диаграммы**, или даже со всех диаграмм, но по-прежнему они **продолжают существовать в модели**.

Типы UML-диаграмм



Синтаксис UML-диаграммы

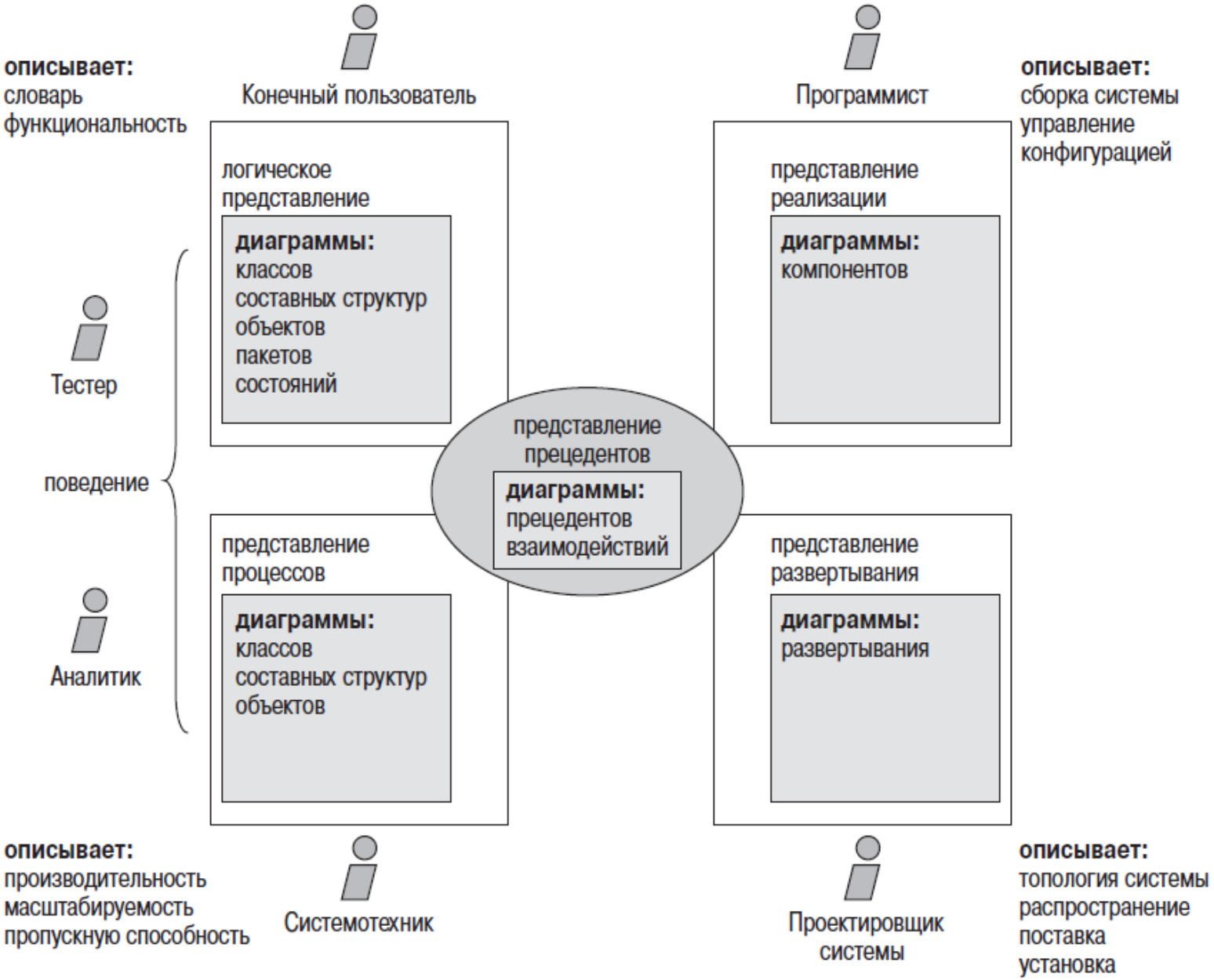


синтаксис заголовка: <тип> <имя> <параметры>
особое внимание: <тип> и <параметры> необязательны

Описание архитектуры посредством UML

- Архитектура системы – это организационная структура системы, включая ее разбиение на части, их связность, взаимодействия, механизмы и направляющие принципы, передающие конструкцию системы.
- Стратегические аспекты системы можно описать «4+1 представлениями» архитектуры:
  1. логическое представление,
  2. представление процессов,
  3. представление реализации,
  4. представление развертывания,
  5. представление прецедентов.

Описание архитектуры системы

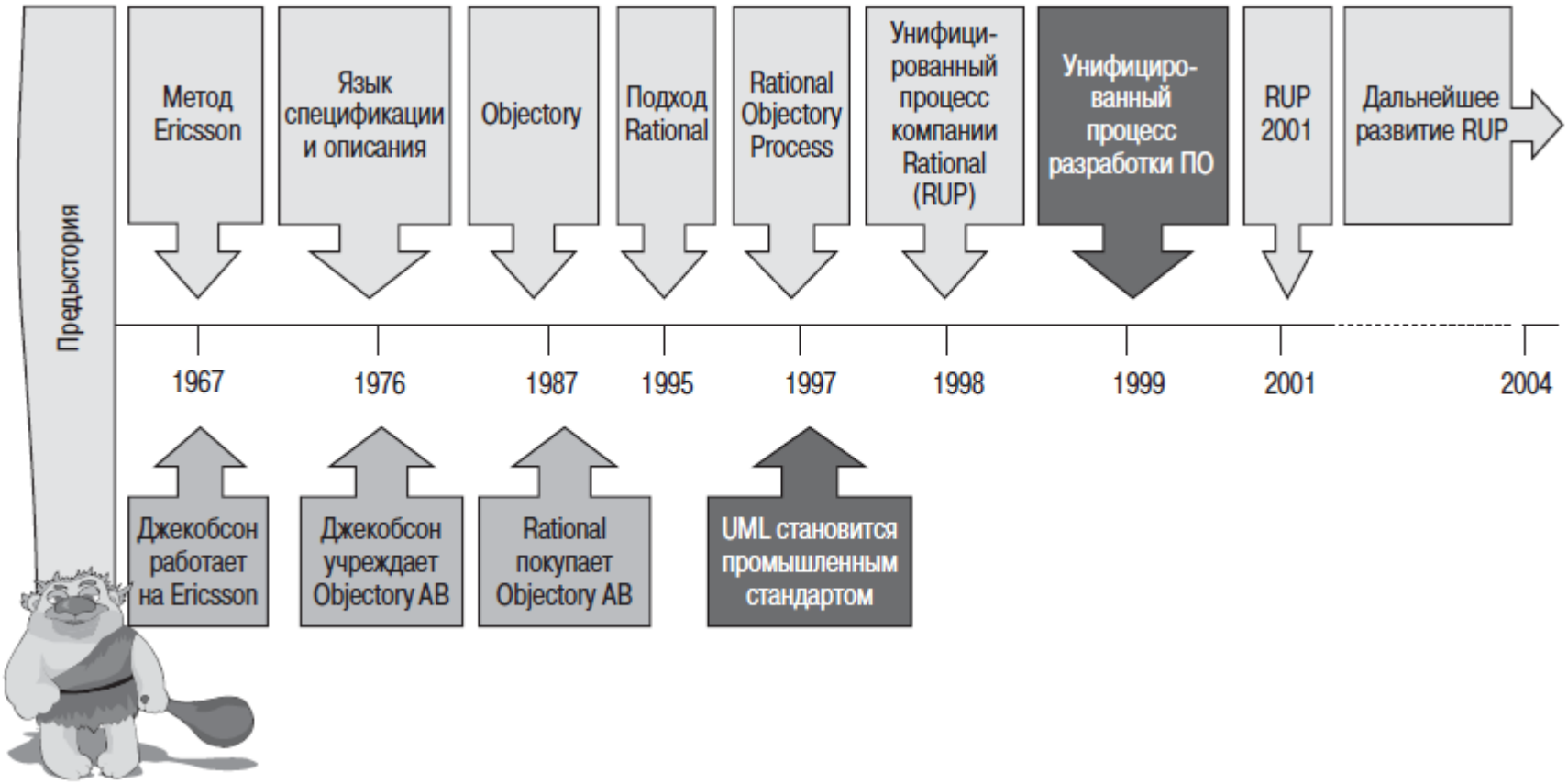


Унифицированный процесс (Unified Process, UP)

Унифицированный процесс

- Процесс производства программного обеспечения (Software Engineering Process, **SEP**), также известный как процесс разработки программного обеспечения (Software Development Process), определяет *кто, что, когда и как* в разработке ПО.
- **Унифицированный процесс разработки программного обеспечения** (Unified Software Development Process, USDP) – это SEP от авторов UML.

История UP


Предыстория
Метод Ericsson
Язык спецификации и описания
Objectory
Подход Rational
Rational Objectory Process
Унифици-рованный процесс компании Rational (RUP)
Унифициро-ванный процесс разработки ПО
RUP 2001
Дальнейшее развитие RUP
1967
1976
1987
1995
1997
1998
1999
2001
2004
Джекобсон работает на Ericsson
Джекобсон учреждает Objectory AB
Rational покупает Objectory AB
UML становится промышленным стандартом

UP и RUP

- Компания Rational развила подход Якобсона в «Унифицированный процесс компании Rational» (**Rational Unified Process, RUP**).
- UP и RUP очень тесно связаны. RUP – это коммерческий продукт, расширяющий UP
- Но RUP – это пропиретарный процесс, являющийся продуктом компании Rational, а UP – это открытый SEP от авторов UML.

Аксиомы UP

Унифицированный процесс является:

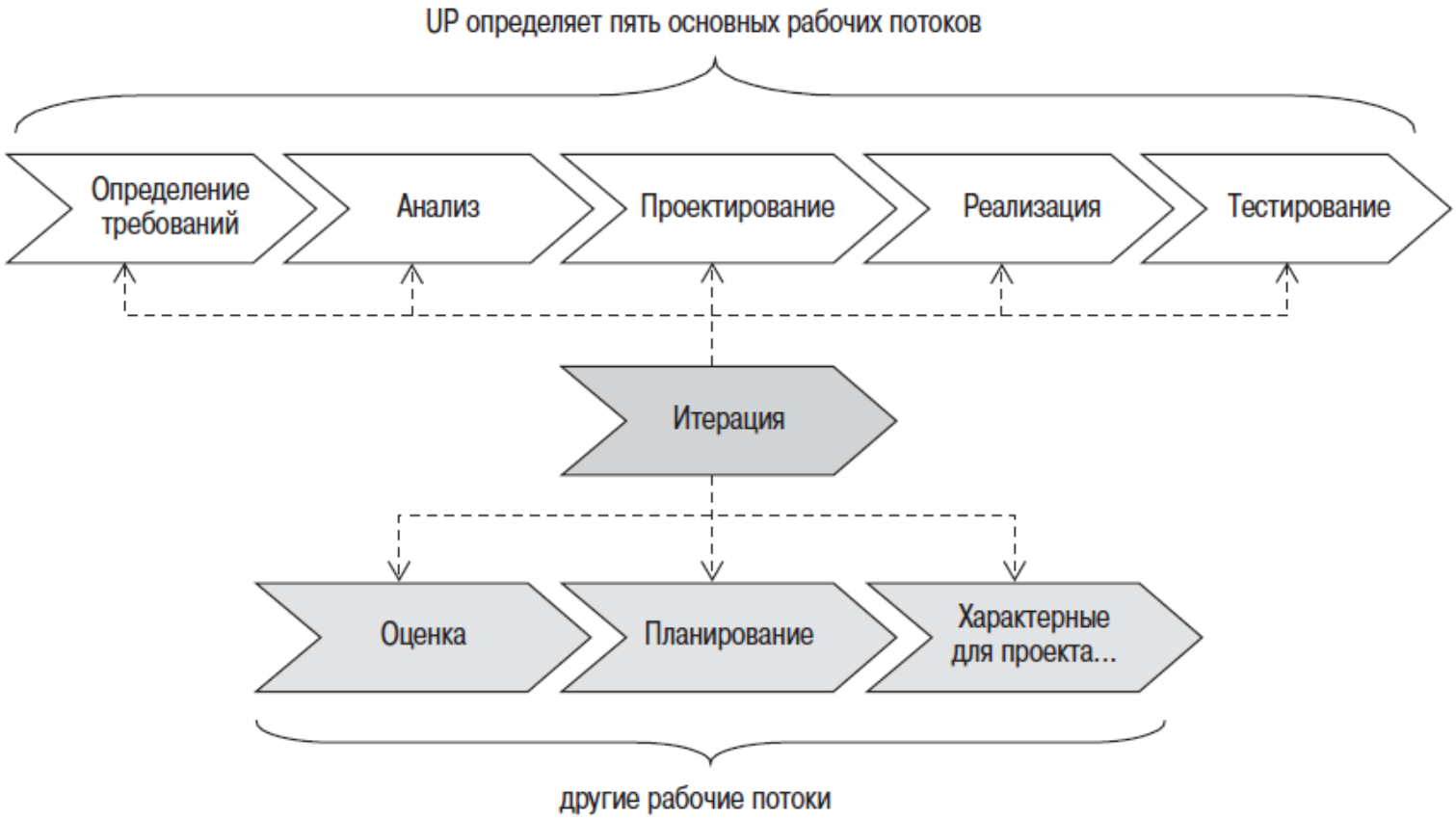
- **управляемым требованиями и риском**
  - требования и анализ возможных рисков закладывается в основу создания ПО
- **архитектуроцентричным**
  - подход UP заключается в создании надежной архитектуры системы
- **итеративным и инкрементным**
  - проект разбивается на небольшие подпроекты (итерации), которые обеспечивают функциональность системы по частям

Итерации UP

Каждая итерация включает ***все*** элементы обычного проекта по разработке ПО:

- **Планирование**
- **Анализ и проектирование**
- **Построение**
- **Интеграция и тестирование**
- **Версия для внутреннего или внешнего использования**

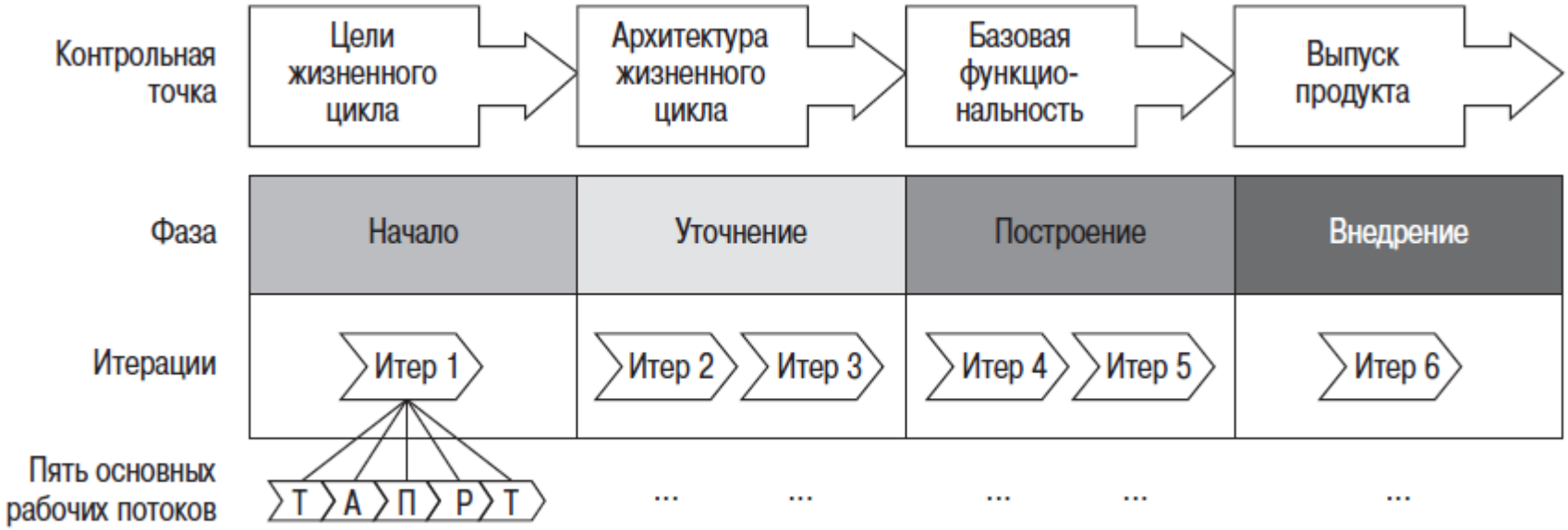
Основные рабочие потоки итерации



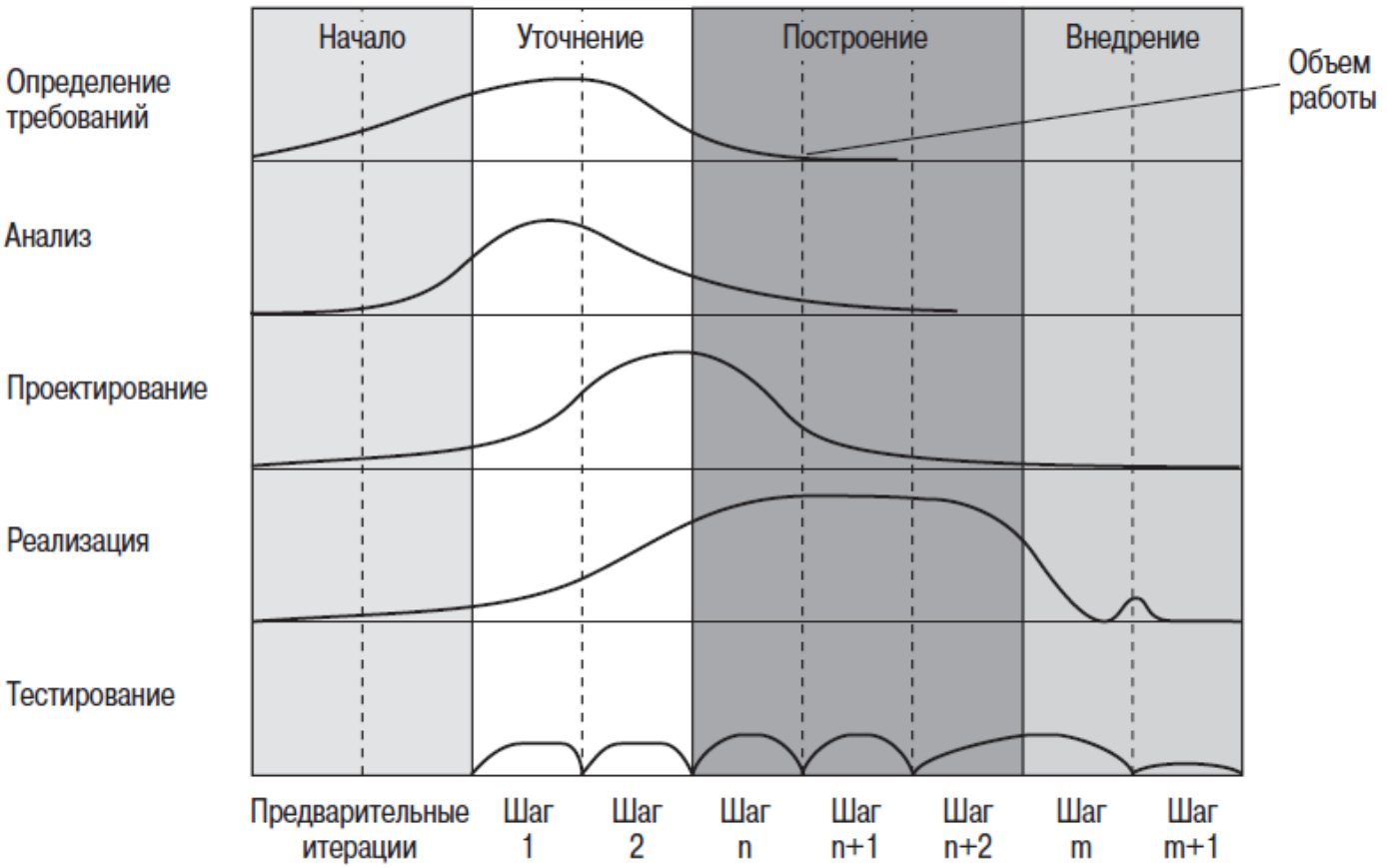
Фазы UP

- В UP четыре фазы, каждая из которых имеет свои контрольные точки. Каждая фаза может состоять из 1 или более итераций.
- **Начало (Inception)** – цели жизненного цикла;
- **Уточнение (Elaboration)** – архитектура жизненного цикла;
- **Построение (Construction)** – базовая функциональность;
- **Внедрение (Transition)** – выпуск продукта.

Структура UP



Распределение работ по фазам UP



Определение требований

Определение требований – основа проектирования системы

- Большая часть работы с требованиями проводится в начале проекта, в фазах Начало и Уточнение.
- До начала работы над ОО анализом и проектированием уже необходимо иметь представление о том, что должно получиться в результате.
- Выработка требований – это переговорный процесс, поскольку часто выдвигаются противоречивые требования, которые должны быть согласованы.

Требования и диаграмма прецедентов

- Некоторые курсы UML утверждают, что прецеденты UML являются единственным способом фиксирования требований.
  - Прецеденты могут отражать только функциональные требования, которые являются описанием того, *что будет делать система.*
- Но существуют требования которые не относятся к функциональности, но являются ограничениями, накладываемыми на систему (производительность, надежность)

Требование

- Требование – это подробное описание того, что должно быть реализовано.
  - Функциональные требования – какое поведение должна предоставлять система;
  - Нефункциональные требования – особое свойство или ограничение, накладываемое на систему

Требования указывают что должно быть построено, но не говорят как это сделать.

Традиционное описание требований

Примеры функциональных требований

- 1. Система ATM должна проверять действительность вставленной в банкомат карточки.
- 2. Система ATM должна проверять достоверность PIN-кода, введенного пользователем.
- 3. Система ATM должна выдавать по одной ATM-карточке не более $250 в сутки.

Примеры нефункциональных требований

- 1. Система ATM должна быть написана на C++.
- 2. Система ATM должна обмениваться информацией с банком, используя 256-разрядную кодировку.
- 3. Система ATM должна проверять действительность карточки ATM в течение не более трех секунд.
- 4. Система ATM должна проверять достоверность PIN-кода в течение не более трех секунд.

Атрибуты требований

Набор критериев важности «MoSCoW»

Значения атрибута Priority	Семантика
Must have (обязан иметь)	Обязательные требования, являющиеся фундаментальными для системы.
Should have (должен иметь)	Важные требования, которые могут быть опущены.
Could have (мог бы иметь)	По-настоящему необязательные требования (реализуются, если есть на это время).
Want to have (хотел бы иметь)	Требования, которые могут подождать до следующих версий системы.

Поиск требований

Поиск требований

- непосредственные пользователи системы
- другие заинтересованные стороны (например, руководители, специалисты обслуживания, установщики)
- другие системы, с которыми взаимодействует данная система
- аппаратные устройства, с которыми взаимодействует данная система
- правовые и регулирующие ограничения
- технические ограничения
- коммерческие цели

Выявление требований

Естественный язык формируют 3 фильтра:

- Пропуск – информация отфильтровывается
- Искажение – информация изменяется взаимосвязанными механизмами вымысла и представления
- Обобщение – информация обобщается в правила, убеждения и понятия об истинности и ложности

Пример «Пропуска»

- **Пример**: «Они используют систему для получения книг на время» – ***пропуск***.
- **Вопрос**: Кто именно использует систему для получения книг на время?
- **Ответ**: читатели, другие библиотеки и библиотекари.

Пример «Искажения»

- **Пример**: «Тот, кто имеет книгу на руках, не может взять другую книгу до тех пор, пока не вернет предыдущую, срок возврата которой истек» – ***искажение***.
- **Вопрос**: Существуют ли такие обстоятельства, при которых кто-либо мог бы взять новую книгу до того, как будут возвращены все имеющиеся на руках книги, срок возврата которых истек?
- **Ответ**: Фактически существует два обстоятельства, при которых право читателя на получение книг может быть восстановлено. Во первых, все имеющиеся на руках книги, срок возврата которых истек, возвращены; во-вторых, за все невозвращенные книги, срок возврата которых истек, внесена плата.

Пример «Обобщения»

- **Пример**: «Для получения книг у всех должен быть формуляр» – ***обобщение***.
- **Вопрос**: Есть ли пользователи системы, которым не обязательно иметь формуляр?
- **Ответ**: Некоторые пользователи системы, например другие библиотеки, могут не иметь формуляра или имеют специальный формуляр с другими сроками и условиями возврата книг.

Квантор общности

- При встрече с квантором общности всегда можно найти пропуск, обобщение и искажение

- **все**
- **каждый**
- **всегда**

- **никогда**
- **никто**
- **нисколько**

Интервью

- Не заблуждайтесь, вам может ***казаться*** что вы все очень хорошо понимаете;
- Задавайте контекстно-свободные вопросы;
- Не занимайтесь телепатией. Телепатия – это **заблуждение** по поводу того, что вам известны чьи-то чувства.
- Лучший способ записи информации во время интервью – бумага и ручка!
- Анкеты не заменяют интервью, но могут быть полезным дополнением.

Резюме

- **UML (Unified Modeling Language)** - это универсальный язык визуального моделирования систем
- **Унифицированный процесс разработки программного обеспечения** (Unified Software Development Process, USDP) – это SEP от авторов UML.
- В UP четыре фазы: Начало, Уточнение, Построение, Внедрение. Каждая фаза может состоять из 1 или более итераций. Каждая итерация состоит из определения требований, анализа, проектирования, реализации, тестирования.

Резюме

- Определение требований – основа проектирования системы.
- Требование – это подробное описание того, что должно быть реализовано
- Бывают Функциональные и нефункциональные требования
- Для выявления требований используют интервью. При проведении интервью необходимо учитывать 3 **фильтра**: пропуск; искажение; обобщение.