# Курс по дисциплине «Программная инженерия»

**Максимов Константин Викторович**

**Аспирант, специализация 08.00.13 «Математические и инструментальные методы в экономике»**

**Научный руководитель Денисов Д.В.**

**kosmosbest@gmail.com**

# СФЕРА НАУЧНЫХ ИНТЕРЕСОВ

1. **Автоматизация управления предприятием (ИС автоматизации отдельных задач, интегрированные ИС, корпоративные ИС, системная интеграция)**
2. **Управление проектами разработки и внедрения программного обеспечения**
3. **Организация и проведение экономического анализа**
4. **Облачные технологии и Big Data**

# ПРОГРАММА КУРСА

- **4 лекции**
- **3 лабораторных практикума**

- **2: 0-50 баллов**
- **3: 51-69 баллов**
- **4: 70-89 баллов**
- **5: 90-100 баллов**

- **Аттестация**

- 3 практических задания: 30 баллов (после дедлайна, задание сдать можно в случае уважительного пропуска)
- Экзамен в виде теста: 50 баллов
- Тесты на занятиях: 20 баллов

# ОСНОВНЫЕ ИСТОЧНИКИ КУРСА

**Курс базируется на:**

1. Стандарт ISO/IEC 12207:2007 System and software engineering. Software life cycle processes Информационная технология СИСТЕМНАЯ И ПРОГРАММНАЯ ИНЖЕНЕРИЯ Процессы жизненного цикла программных средств.
2. Software Engineering Body of Knowledge (SWEBOK)

# ЛЕКЦИЯ 1

**Основные темы:**

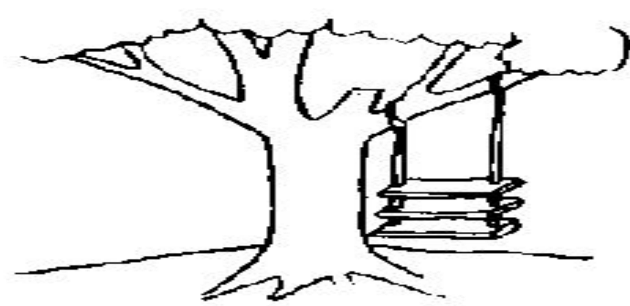
- Проблемы разработки сложных программных систем
- Жизненный цикл и процессы разработки ПО
- Модели жизненного цикла
- Унифицированный процесс разработки и экстремальное программирование

# ПРЕДМЕТ И ОБЪЕКТ

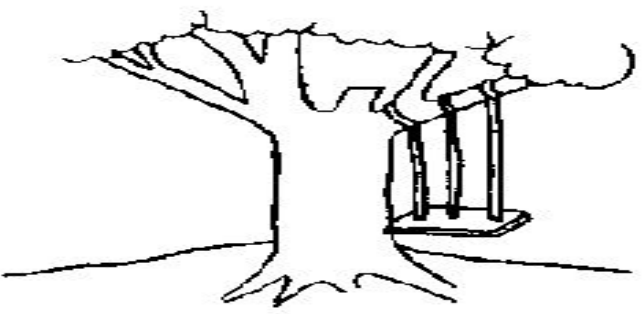
**Предметом** изучения является технология, методы и средства разработки, тестирования и отладки программного продукта.

**Объектом** изучения выступает жизненный цикл программного обеспечения.

# КАК ПРОЕКТИРУЮТСЯ ПРОГРАММЫ



1. Как было предложено организатором разработки

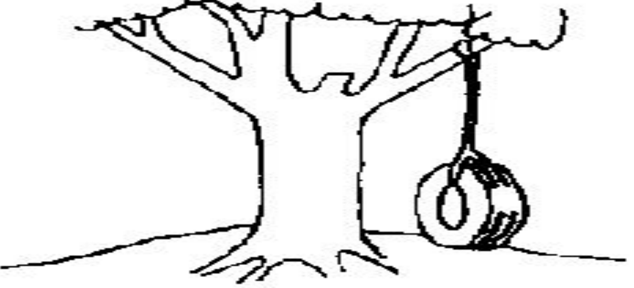


2. Как было описано в техническом задании

3. Как было спроектировано ведущим системным специалистом

4. Как было реализовано программистами

5. Как было внедрено



6. Чего хотел пользователь

# ПРОБЛЕМЫ РАЗРАБОТКИ СЛОЖНЫХ ПРОГРАММНЫХ СИСТЕМ

***Сложные*** или ***«большие» программы,*** называемые также ***программными системами***, ***программными комплексами***, ***программными продуктами,*** отличаются от «небольших» не столько по размерам (хотя обычно они значительно больше), сколько по наличию дополнительных факторов, связанных с их востребованностью и готовностью пользователей платить деньги как за приобретение самой программы, так и за ее сопровождение и даже за специальное обучение работе с ней.

# ОСОБЕННОСТИ СЛОЖНОЙ ПРОГРАММЫ

- Решает одну или несколько связанных задач
- Ее низкая производительность на реальных данных приводит к значимым потерям для пользователей.
- Существенно, чтобы она была удобной в использовании.
- Ее неправильная работа наносит ощутимый ущерб пользователям и другим организациям и лицам, даже если сбои происходят не слишком часто.
- В ее разработку вовлечено значительное количество людей (более 5-ти человек).

# ПРОГРАММНАЯ ИНЖЕНЕРИЯ

Изучением организационных, инженерных и технических аспектов создания ПО, включая методы разработки, занимается дисциплина, называемая **программной инженерией.**

Часто программное обеспечение (ПО) нельзя рассматривать отдельно от программно-аппаратной системы, куда оно входит в качестве составной части.

# ПРИНЦИПА РАБОТЫ СО СЛОЖНЫМИ ПРОГРАММАМИ

- **Абстракция (abstraction) и уточнение (refinement).**
- **Модульность (modularity).**
- **Переиспользование.**

# ЖИЗНЕННЫЙ ЦИКЛ И ПРОЦЕССЫ РАЗРАБОТКИ ПО

Весь период существования ПО, связанный с подготовкой к его разработке, разработкой, использованием и модификациями, начиная с того момента, когда принмается решение разработать/приобрести/собрать из имеющихся компонентов новую систему или приходит сама идея о необходимости программы определенного рода, до того момента, когда полностью прекращается всякое ее использование, называют **жизненным циклом ПО.**

# Проект

- Одним из ключевых понятий технологии разработки программного обеспечения, как и многих других областей деятельности, является понятие *проекта.*
- *Проект* есть уникальное временное предприятие, направленное на создание определенного, уникального продукта и услуги.
- Технология *управления проектом* есть совокупность знаний, навыков, инструментов и методов для планирования и реализации действий, направленных на достижение поставленной в рамках проекта цели.
- *Процесс* разработки программного обеспечения является плохо определенным и динамичным.

# Четыре «П» разработки ПО

- Персонал

  (кто это делает)
- Процесс

  (способ, которым это делается)
- Проект

  (выполнение необходимых действий)
- Продукт

  (артефакты)

# Продукт

Артефакт – любой вид информации, создаваемый, изменяемый и используемый сотрудниками при создании системы

Артефакты:

- Само приложение
- Спецификация требований
- Проектная модель
- Исходный и объектный код
- Тестовые процедуры
- …

# Проект

Совокупность действий, необходимых для создания артефакта:

- контакт с заказчиком
- написание документации
- проектирование
- программирование
- тестирование
- …

# Процесс

- Процесс создания ПО – определение полного набора видов деятельности, необходимых для преобразования требований пользователя в продукт.
- Процесс служит шаблоном для создания проекта.
- Процесс определяет:
  - кто делает
  - что делает
  - когда делает
  - как достичь цели
- Процессы делятся на тяжеловесные и легковесные (гибкие)

# Стандарты жизненного цикла

- IEEE — читается «ай-трипл-и», Institute of Electrical and Electronic Engineers, Институт инженеров по электротехнике и электронике;
- ISO — International Standards Organization, Международная организация по стандартизации;
- EIA — Electronic Industry Association, Ассоциация электронной промышленности;
- IEC — International Electrotechnical Commission, Международная комиссия по электротехнике;
- • ANSI — American National Standards Institute, Американский национальный институт стандартов;
- • SEI — Software Engineering Institute, Институт программной инженерии;
- • ECMA — European Computer Manufactures Association, Европейская ассоциация производителей компьютерного оборудования.

# Группа стандартов ISO

- **ISO/IEC 12207 Standard for Information Technology — Software Life Cycle Processes** [1] (процессы жизненного цикла ПО, есть его российский аналог **ГОСТ Р-1999** [2]).
- **ISO/IEC 15288 Standard for Systems Engineering — System Life Cycle Processes** [3] (процессы жизненного цикла систем).
- **ISO/IEC 15504 (SPICE) Standard for Information Technology — Software Process Assessment** [4] (оценка процессов разработки и поддержки ПО).

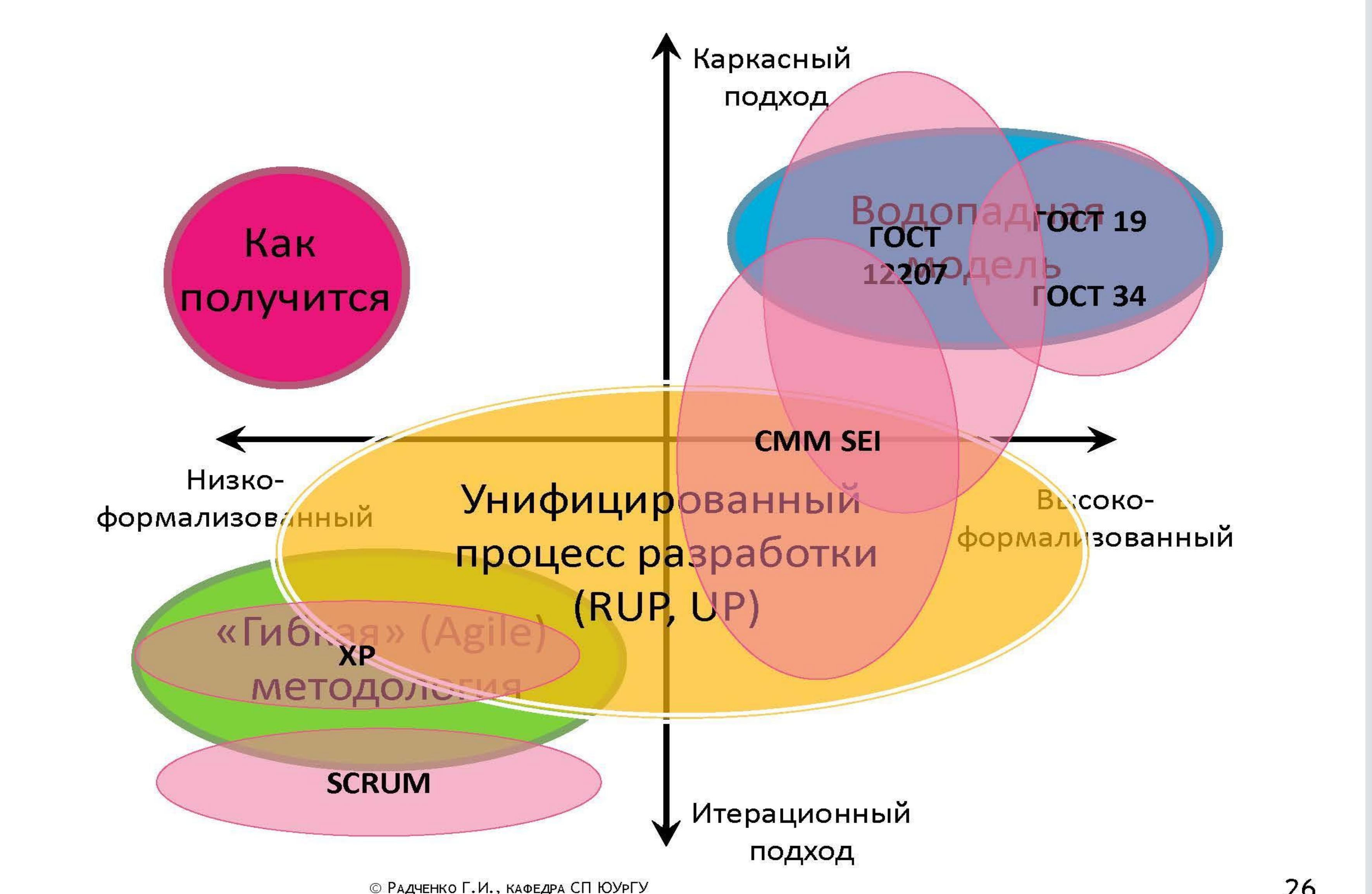
# Группа стандартов IEEE и CMM, разработанных SEI

- **IEEE 1074-1997 — IEEE Standard for Developing Software Life Cycle Processes** [5] (стандарт на создание процессов жизненного цикла ПО).
- **IEEE/EIA 12207-1997 — IEEE/EIA Standard: Industry Implementation of International Standard ISO/IEC 12207:1995 Software Life Cycle Processes** [6-8] (промышленное использование стандарта ISO/IEC 12207 на процессы жизненного цикла ПО).
- **Модель зрелости возможностей CMM (Capability Maturity Model)** [9,10] предлагает унифицированный подход к оценке возможностей организации выполнять задачи различного уровня.
- **Интегрированная модель зрелости возможностей CMMI (Capability Maturity Model Integration)** [11,12]. Эта модель представляет собой результат интеграции моделей CMM для продуктов и процессов, а также для разработки ПО и разработки ПАК

# Модели жизненного цикла

- ***каскадная*** или ***водопадная (waterfall)*** модель жизненного цикла
- ***Итеративные*** или ***инкрементальные модели*** (известно несколько таких моделей)
- ***спиральная*** модель жизненного цикла ПО

# Сводный график стандартов



# Семейства процессов разработки ПО

- тяжеловесные (heavyweight)
  - применяются при фиксированных требованиях и многочисленной группе разработчиков разной квалификации
- облегченные (lightweight, agile)
  - применяются при малочисленной группе квалифицированных разработчиков и грамотном заказчике, который имеет возможность участвовать в процессе

Начнем с гибких технологий - наиболее актуальных.

# Стратегии создания ПО

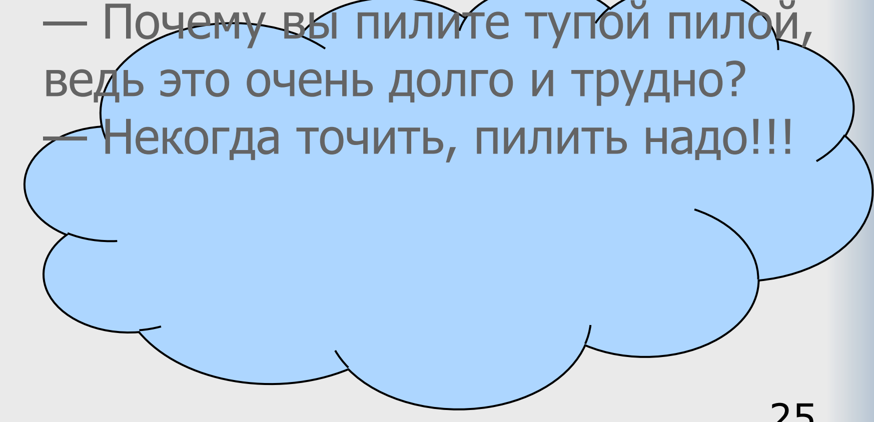
| | *Водопад-ная* | *Итеративные* | |
|---|---|---|---|
| | | *Инкремен-тная* | *Эволюци-онная* |
| В начале определены все требования? | + | + | - |
| Циклов конструирования | 1 | >1 | >1 |
| Промежуточное ПО распространяется? | - | ± | + |

# Технологии программирования

Технология программирования (технология разработки ПО) — способ организации **процесса** создания программы, совокупность приемов и способов выполнения определенных видов деятельности.

На разных уровнях и по разным критериям выделяют пересекающиеся модели:

- Водопадная (каскадная) модель, нисходящее (структурное) программирование
- Макетирование
- Спиральная (итерационная) модель разработки ПО
- Объектно-ориентированное программирование
- Гибкие (agile) технологии: экстремальное программирование (XP), Scrum, TDD, FDD…
- RUP
- Компонентный подход (COM, CORBA)
- CASE-технологии
- RAD
- …



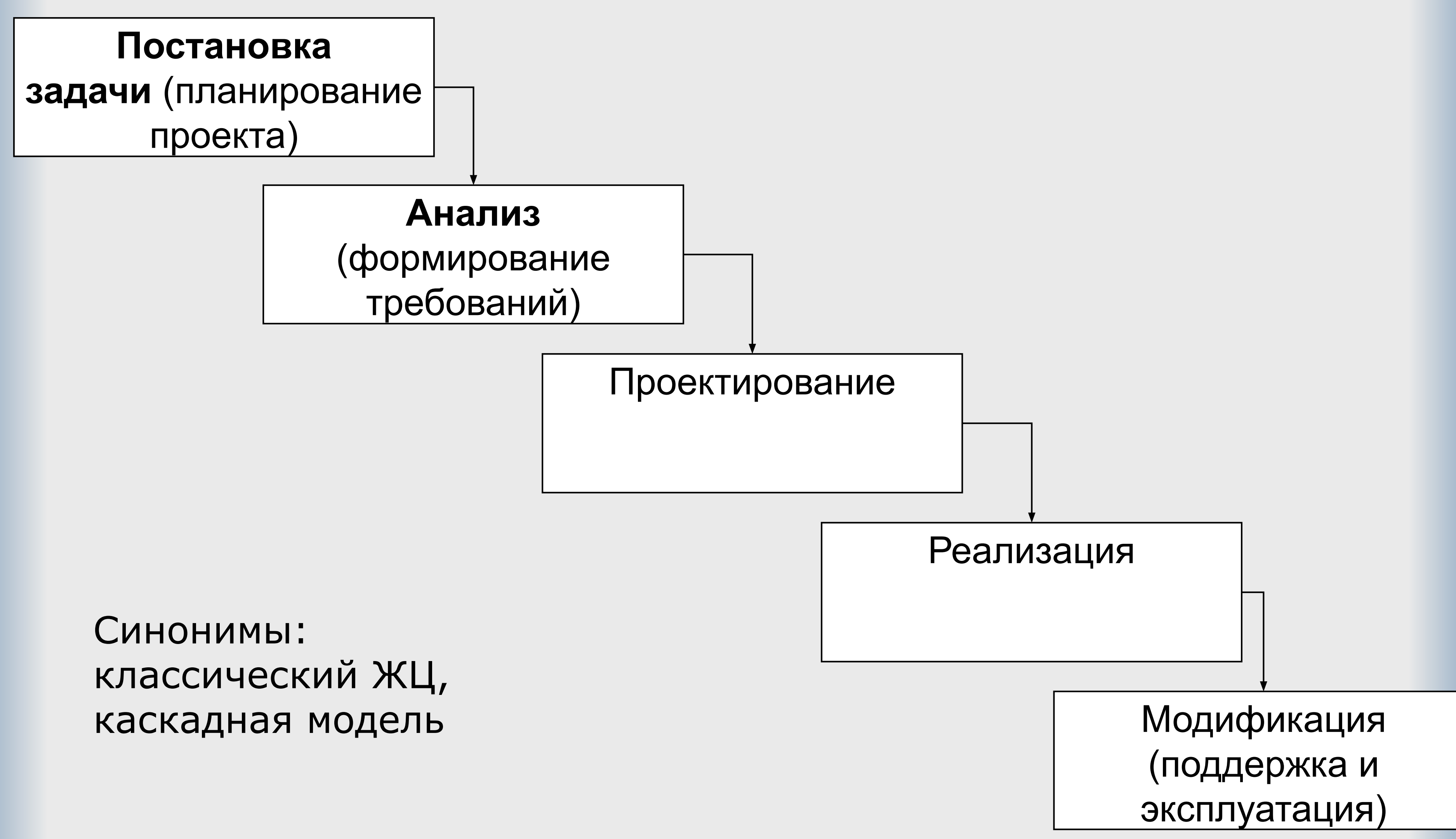
# Источники сложности проекта

- Наличие высококвалифицированных специалистов на рынке труда.
- Стабильность используемой технологической платформы, стабильность и функциональность инструментов разработки.
- Эффективность используемых методов разработки, включая методы моделирования, проектирования, тестирования и управления версиями.
- Доступность специалистов, обладающих экспертизой в прикладной области.
- Используемая методология и ее соответствие данному проекту.
- Сроки и финансирование проекта.
- Множество других организационных и технических переменных.

# Проблемы управления проектами

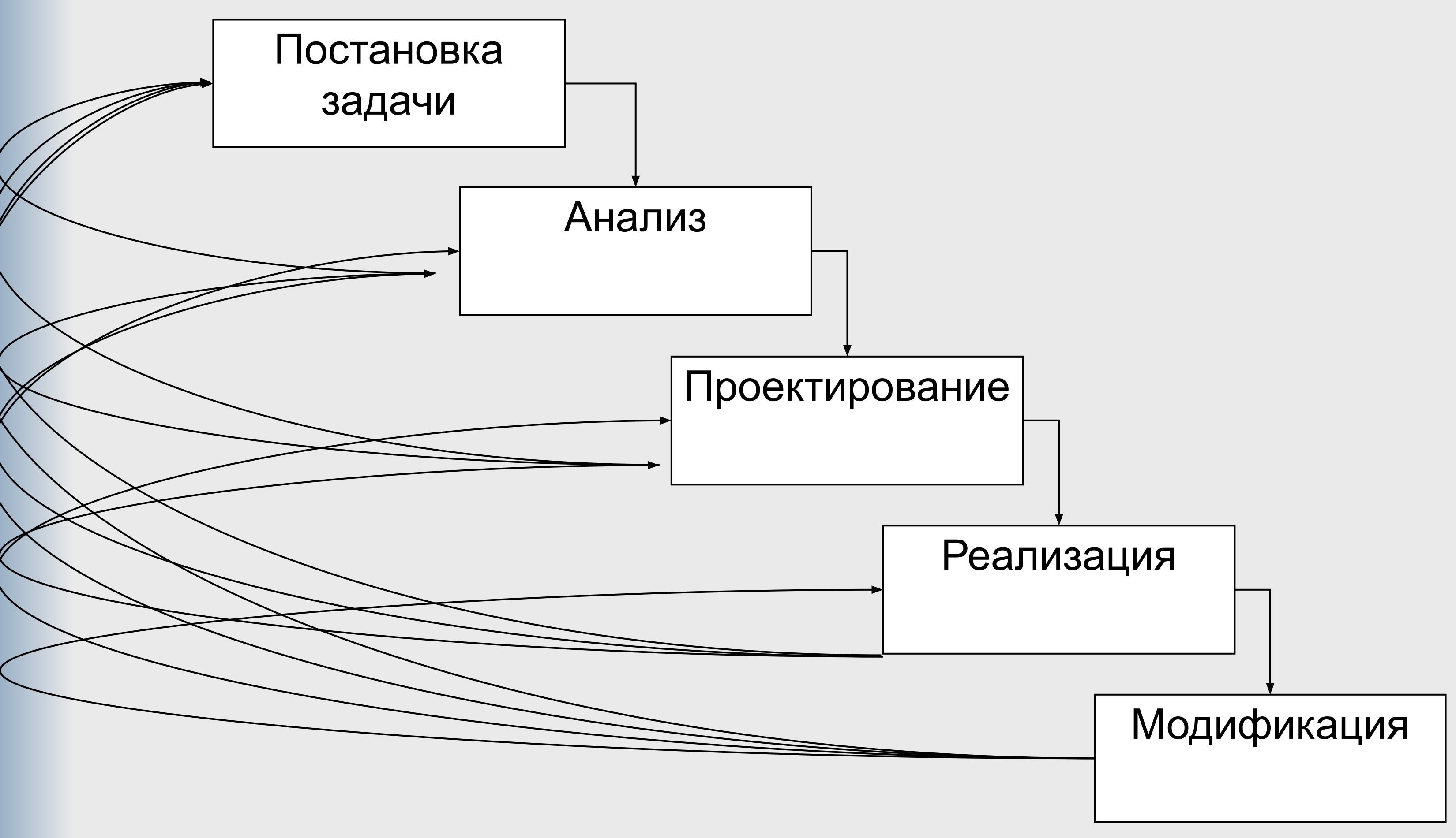
- Многие процессы разработки неуправляемы. Их исходные данные и желаемый результат неизвестны или определены очень нечетко.
- Процесс достижения желаемого результата не поддается формализации (например, разработка архитектуры и исчерпывающее тестирование продукта).
- Идентифицированные процессы разработки сопровождаются неизвестным количеством неидентифицированных.
- Требования к продукту часто меняются в течение жизненного цикла проекта, что требует сложной процедуры изменения и согласования требований.
- Попытки предложить формальную, детализованную методологию разработки ПО оказываются безуспешны, потому что сам процесс разработки не поддается детализации и формализации.
- Слепое следование методологиям, предполагающим управляемость и предсказуемость процессов разработки, приводит к непредсказуемым результатам проекта.

# Водопадная модель жизненного цикла ПО:

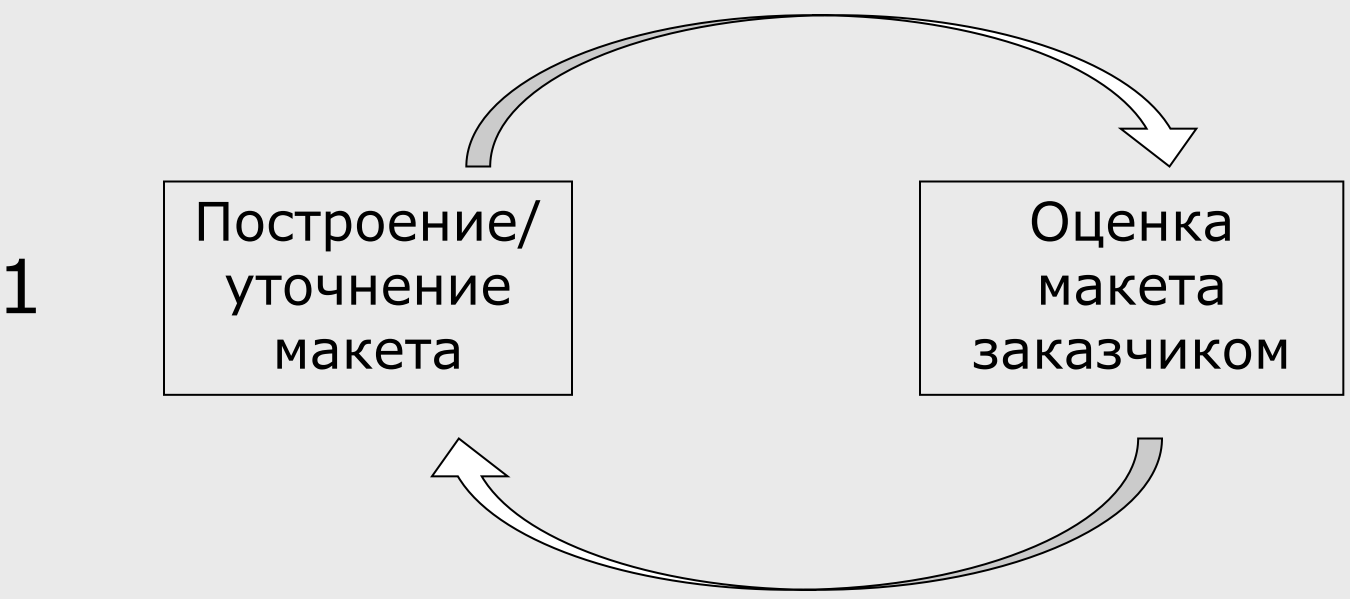



Синонимы:
классический ЖЦ,
каскадная модель

# Модель с промежуточным контролем:



# Макетирование (прототипирование)




2 Проектирование продукта

# Инкрементная модель

# Технология RAD

Rapid Application Development — Быстрая разработка приложений. Ориентирована на максимально быстрое получение первых версий разрабатываемого ПО. Она предусматривает:

- ведение разработки небольшими группами (3-7 человек), каждая из которых проектирует и реализует отдельные подсистемы, позволяет улучшить управляемость проекта;
- использование готовых компонентов способствует уменьшению времени получения работоспособного прототипа;
- наличие четко проработанного графика цикла, рассчитанного не более чем на три месяца, существенно увеличивает эффективность работы.

- Технология RAD хорошо зарекомендовала себя для относительно небольших стандартных проектов, разрабатываемых для конкретного заказчика.

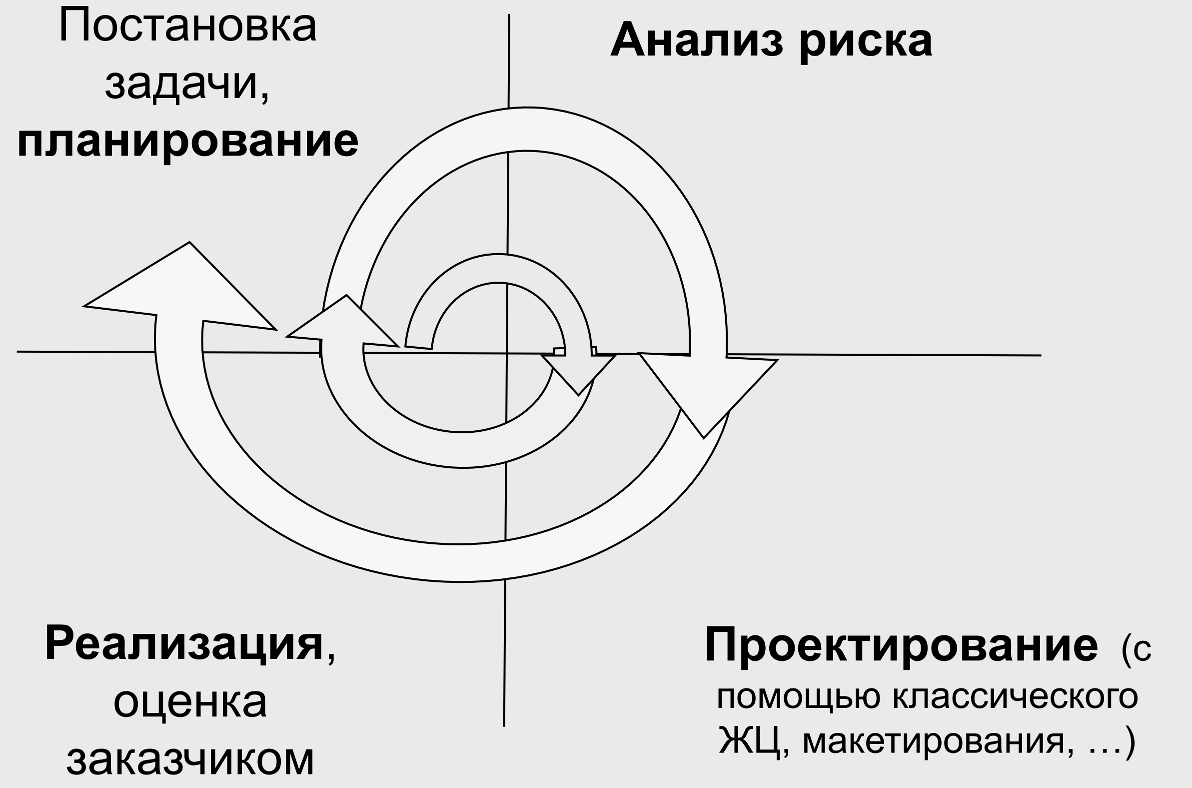
# Этапы RAD

- Бизнес-моделирование (моделируются информационные потоки между бизнес-функциями)
- Моделирование данных (набор объектов, которые требуются для поддержки бизнес-процессов)
- Моделирование обработки (определяются преобразования объектов, обеспечивающие реализацию бизнес-функций. Описание обработки для добавления, изменения, удаления и поиска данных)
- Создание приложения (используются готовые компоненты и утилиты автоматизации)
- Объединение и тестирование (компоненты тестировать не надо).

# Спиральная модель разработки ПО

- Программное обеспечение создается итерационно с использованием метода прототипирования.
- Прототипом обычно называют действующий программный продукт, реализующий отдельные функции и внешние интерфейсы разрабатываемого программного обеспечения.

На 1-й итерации может использоваться макет, который оценивается заказчиком.



# Особенности спиральной модели

Основным *достоинством* спиральной схемы является то, что, начиная с некоторой итерации, продукт можно предоставлять пользователю, что позволяет:

- сократить время до появления первых версий программного продукта;
- заинтересовать большое количество пользователей, обеспечивая быстрое продвижение следующих версий продукта на рынке;
- ускорить формирование и уточнение спецификаций за счет появления практики использования продукта;
- уменьшить вероятность морального устаревания системы за время разработки.

Основной *проблемой* использования спиральной схемы является определение моментов перехода на следующие стадии. Для ее решения обычно ограничивают сроки прохождения каждой стадии, основываясь на экспертных оценках.

Спиральную модель применяют для программ, основанных как на процедурной, так и на объектно-ориентированной парадигме.

# Гибкие технологии разработки ПО

- Минимизируют риски благодаря разделению процесса разработки на маленькие промежутки времени (*итерации*), обычно 1-4 недели.
- Каждая итерация может рассматриваться как полноценный проект (может включать в себя планирование, анализ требований, проектирование, реализацию, тестирование и документирование).
- Обычно результатом итерации не является продукт, готовый к выходу на рынок. Но целью каждой итерации является получение стабильной версии продукта.
- В конце каждой итерации происходит переоценка приоритетов проекта, что значительно сокращает риски.

Все гибкие методологии имеют общие характеристики:

- итеративная разработка;
- фокус на взаимодействии и коммуникации;
- полный или частичный отказ от создания дорогостоящих промежуточных артефактов проекта.

# Основные идеи agile

- Личности и их взаимодействие важнее, чем процессы и инструменты.
- Работающее программное обеспечение важнее, чем полная документация.
- Сотрудничество с заказчиком важнее, чем переговоры по контракту.
- Реакция на изменения важнее, чем следование плану.

Краеугольным камнем гибких технологий программирования является **разработка через тестирование**:

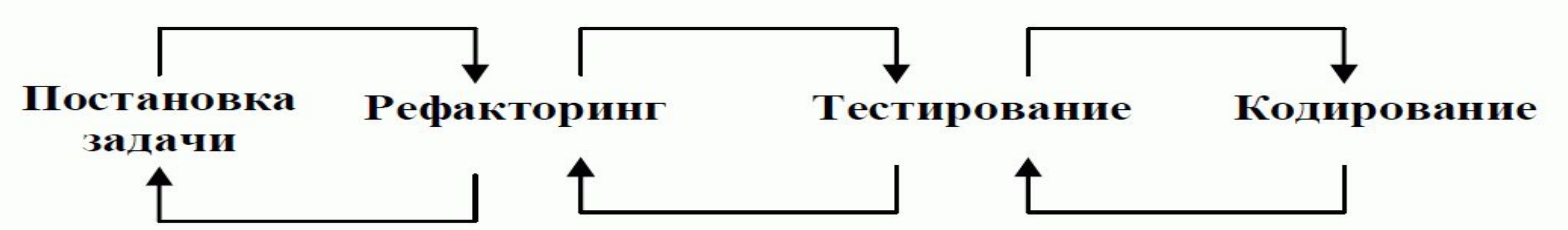
- автоматические тесты пишутся для любой части реализации, которая гипотетически «может сломаться»;
- тесты пишутся непосредственно *перед* написанием соответствующего кода;
- существующий код никогда не меняется без написания соответствующих тестов;
- выполняется регулярный запуск всех автоматических тестов.

# Основы манифеста гибких технологий

- Главное – удовлетворение требований заказчика путем скорой и непрерывной поставки ценного и работоспособного ПО.
- Приветствуются изменяющиеся требования: их используют для повышения конкурентоспособности продукта.
- Работоспособное ПО поставляется как можно чаще, периодами от пары недель до пары месяцев.
- Бизнесмены и разработчики ежедневно работают сообща.
- Проекты строятся вокруг мотивированных личностей, которым оказывается доверие и создаются все условия для работы.
- Наиболее эффективным способом передачи информации (как внутри команды разработчиков, так и вовне) является личный разговор.
- Основной мерой прогресса является работоспособное ПО.
- Устанавливается удобный режим ведения разработки.
- Непрерывное внимание к техническому совершенству и хорошему дизайну повышает гибкость.
- Простота — искусство НЕ делать лишней работы.
- Лучшие архитектурные решения, наборы требований и дизайны создаются самоорганизующимися командами.
- Команда регулярно рассматривает и внедряет любые методы повышения своей эффективности.

# Проектирование в гибких технологиях

- Отказ от длительного проектирования перед началом работы и выполнение проектирования на протяжении всего выполнения проекта.
- В начале проекта выполняется лишь *формирование общего представления*. Для этого используются *системные метафоры*, на основе которых формируется высокоуровневая схема проекта.
- Процесс разработки состоит из большого количества очень коротких циклов. Конечный результат этапа планирования – список задач, подлежащих реализации на следующей итерации.




- Разработчики получают задачу, берут соответствующий фрагмент разрабатываемого кода, выполняют рефакторинг, необходимый для упрощения написанного кода, составляют тесты, а только затем создают сам код, который должен пройти тесты.
- Поскольку циклы «*дизайн–тест–код*» непродолжительны, а заказчик часто получает работающие версии программного продукта, обратная связь осуществляется непрерывно и служит для контроля, что проектирование и кодирование продвигаются в нужном направлении.
- Так как изменения на каждом цикле малы, решения, от которых приходится отказываться, невелики, в результате чего можно быстро реагировать на изменения с наименьшими затратами.

# Экстремальное программирование

- Основная идея экстремального программирования (XP) — устранить высокую стоимость изменений, вносимых в ПО в процессе как разработки, так и эксплуатации.
- Цикл разработки в XP состоит из очень коротких итераций. Четырьмя базовыми действиями в цикле являются:
  - выслушивание заказчика
  - проектирование
  - кодирование
  - тестирование.
- Заказчик постоянно присутствует в группе разработчиков.
- При принятии решений всегда стремятся выбрать самое простое, тесты пишутся еще до написания кода.
- Сборка системы выполняется ежедневно.

Идеолог XP - Кент Бек

# Основные принципы XP

- Планирование
- Частая смена версий
- Метафора
- Простой проект
- Тесты
- Переработка системы
- Программирование в паре
- Непрерывная интеграция

- Коллективное владение
- Заказчик с постоянным у частием
- 40-часовая неделя
- Открытое рабочее пространство
- Стандарты кодирования
- Не более чем правила

Область применимости XP: небольшие и средние проекты.

# Тестирование в XP

**Тестирование модулей (unit testing):**

- позволяет разработчикам убедиться, что код работает корректно, и без опасений выполнять рефакторинг (refactoring).
- помогает не авторам кода понять, зачем нужен тот или иной фрагмент кода и как он функционирует

**Приемочное тестирование (acceptance testing):**

- позволяет убедиться в том, что система действительно обладает заявленными возможностями и функционирует корректно.

**TDD (Test Driven Development):**

- пишется тест (не проходит)
- пишется код, чтобы тест прошел
- выполняется рефакторинг кода.

# Scrum

- Основой *Scrum* является итеративная разработка. *Scrum* определяет итеративные правила управления проектом, которые призваны обеспечивать достижение максимального эффекта от реализованной функциональности.
- В *Scrum* определяются основные правила взаимодействия участников команды, которые призваны обеспечивать максимально быструю реакцию на существующую ситуацию.
- Каждая итерация в *Scrum* может быть описана так: планируем – фиксируем – реализуем – анализируем.
- За счет фиксирования требований на время одной итерации и изменения длины итерации методология *Scrum* позволяет управлять балансом между гибкостью и предсказуемостью разработки.

# Общие положения

**3 роли:**

- *владелец продукта* (*Product Owner*) - отвечает за определение требований к продукту
- *команда* (*Team*) - группа самостоятельных и инициативных разработчиков, ответственных за реализацию проекта
- *скрам-мастер* (*ScrumMaster*) отвечает за решение всех организационных проблем и соблюдение методологии *Scrum*.

**3 фазы проекта:**

- *Подготовка* (*Pregame*)*:* общий план проекта, список основных требований к продукту, высокоуровневая архитектура продукта.
- *Реализация* (*Game*)*:* итеративное развитие продукта.
- *Завершение* (*Postgame*)*:* действия, необходимые для подготовки продукта к выходу на рынок.

# Реализация проекта в Scrum

- Фаза реализации разбита на последовательность итераций - *спринтов* (*Sprint*).
- В результате каждого спринта в продукте реализуется новый, заметный для владельца продукта, объем функциональности.
- В конце каждого спринта продукт остается в работоспособном состоянии.
- Спринт начинается с сессии планирования (*Sprint Planning Meeting*) - определяется объем функциональности, которая будет реализована в течение спринта.
- Ежедневно проводится собрание участников проекта - *скрам-сессия* (*Daily Scrum Meeting*).
- По завершению спринта проводится демонстрационная сессия (*Sprint Review Meeting*).

# Документация в Scrum

Всего 3 документа:

- *журнал продукта* (*Product Backlog*)
  - высокоуровневый список функциональных и технических требований, необходимых для реализации продукта
- *журнал спринта* (*Sprint Backlog*)
  - детализированный список функциональных и технических требований, необходимых для успешного завершения итерации
- *график спринта* (*Burndown Chart*).
  - показывает ежедневное изменение общего объема работ, оставшегося до завершения итерации.

# Унифицированный процесс (RUP)

- Разработчики: Г. Буч, А. Якобсон, Д. Рамбо (Rational, 1998)
- Обобщенный каркас процесса разработки ПО
- Компонентно-ориентирован

УП управляет действиями всех его участников:

- разработчиков
- руководства
- пользователей
- заказчиков

Процесс должен постоянно адаптироваться к реальному положению дел, которое определяется:

- доступными технологиями
- утилитами
- персоналом
- организационными шаблонами.

# Характеристики УП

- управляемый вариантами использования
- архитектурно-ориентированный
- итеративный и инкрементный
- использует UML
- основан на компонентном подходе, использует стандарт визуального моделирования

Варианты использования

Направляет

Управляет

Архитектура

- Архитектура - представление всего проекта с выделением важных характеристик. Архитектура описывается различными представлениями и охватывает наиболее важные статические и динамические аспекты системы.
- Разработка делится на мини-проекты (итерации), в ходе которых реализуется группа вариантов использования. Итерации не обязательно аддитивны.

# Преимущества управляемого УП

- Ограничивает финансовые риски затратами на одну итерацию
- Снижает риск непоставки продукта
- Ускоряет темпы процесса разработки в целом
- Облегчает адаптацию к неизбежным изменениям требований

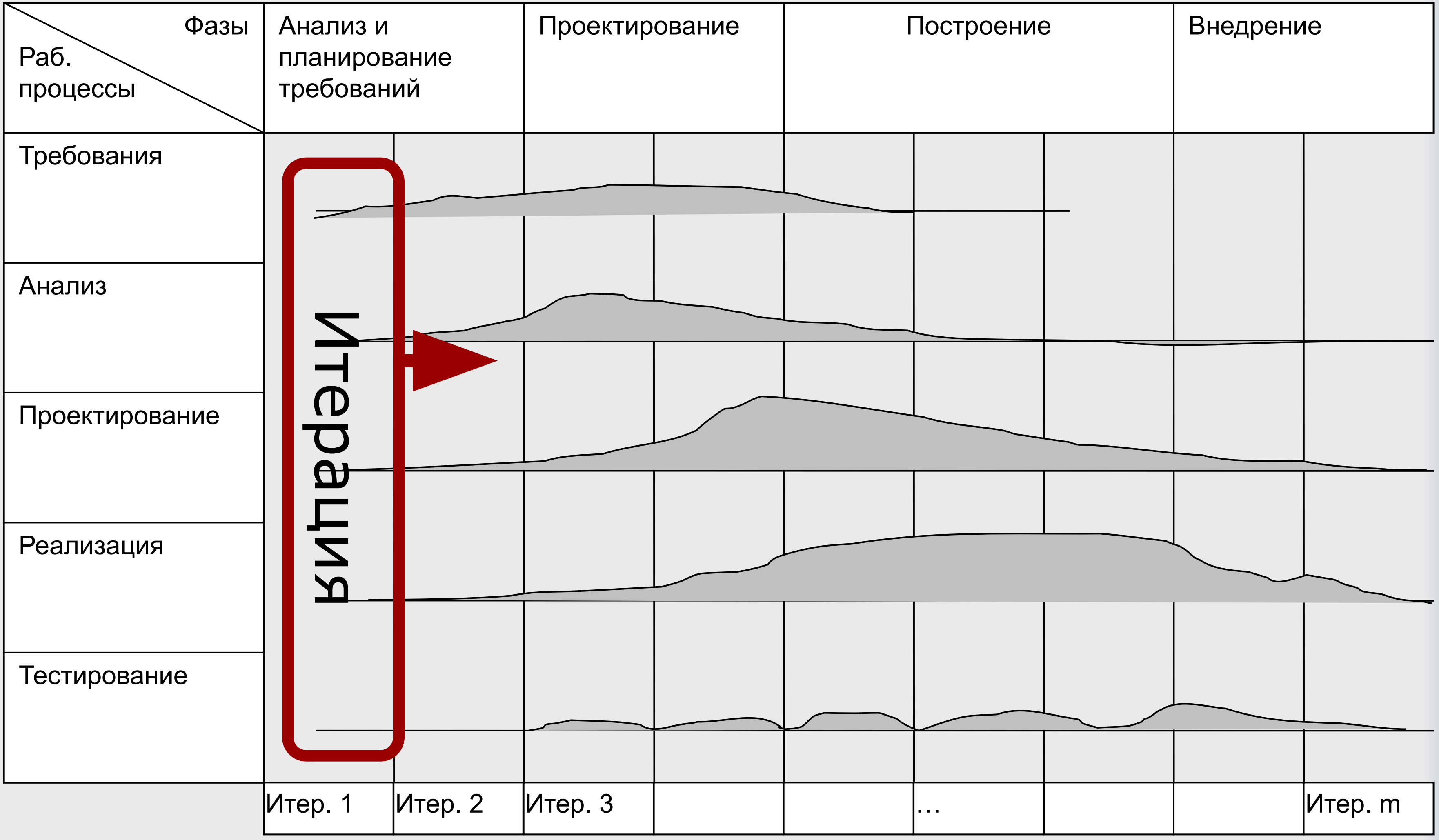
# Жизненный цикл УП

- Каждый цикл состоит из 4х фаз, каждая фаза разделяется на итерации
- Результатом каждого цикла является новый выпуск системы
- Каждая фаза заканчивается вехой
- Веха определяется по наличию определенного набора артефактов
- Артефакт – любой вид информации, создаваемый, изменяемый и используемый сотрудниками при создании системы

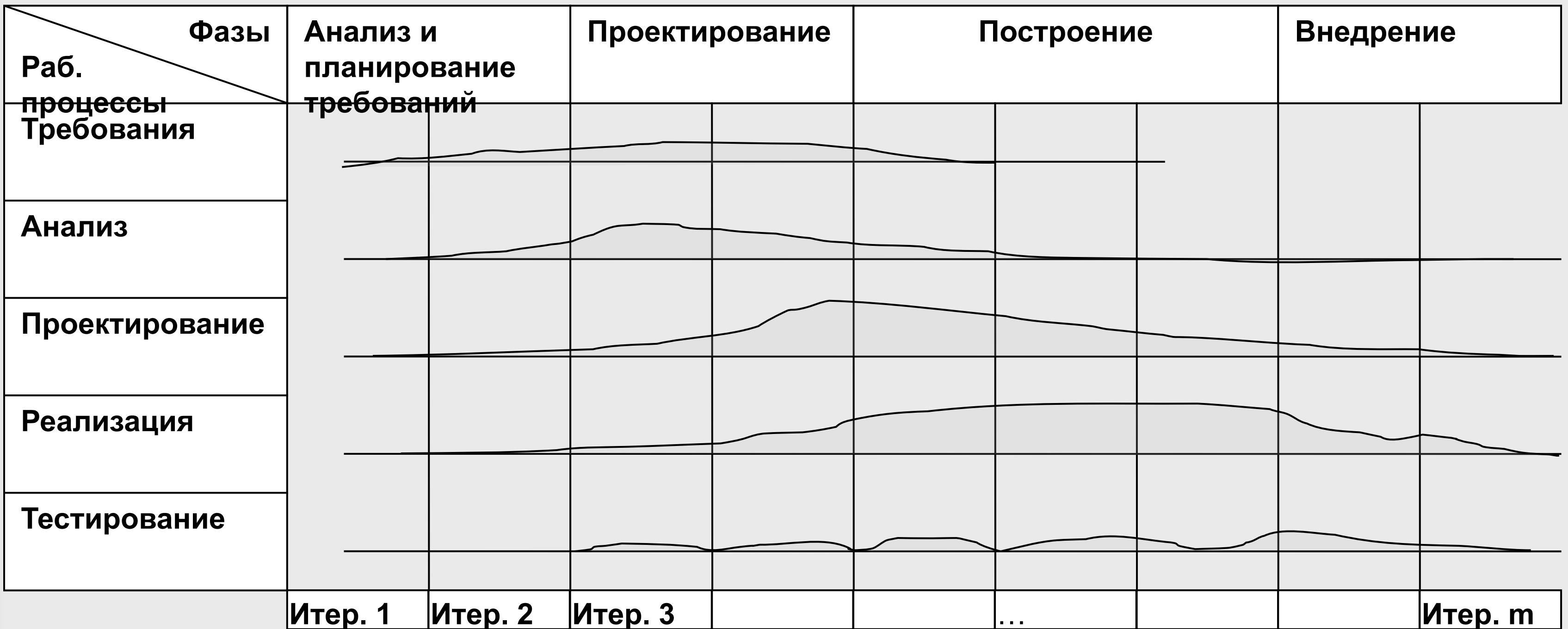
# Назначение вех

- По ним руководитель принимает решения перед тем, как перейти на следующую фазу
- Возможность отслеживать процесс
- Возможность прогнозирования оценок в других процессах

# Цикл разработки



# Содержание фаз




Анализ и планирование требований:

- идея превращается в концепцию готового продукта
- создается бизнес-план разработки
- упрощенная модель вариантов использования
- пробный вариант архитектуры
- выявление рисков и расстановка приоритетов
- грубая оценка проекта

Проектирование:

- детальное описание вариантов использования
- архитектура в виде представлений всех моделей
- план действий и оценка ресурсов

| Фазы / Раб. процессы | Анализ и планирование требований | | Проектирование | | Построение | | | Внедрение | |
|---|---|---|---|---|---|---|---|---|---|
| Требования | | | | | | | | | |
| Анализ | | | | | | | | | |
| Проектирование | | | | | | | | | |
| Реализация | | | | | | | | | |
| Тестирование | | | | | | | | | |
| | Итер. 1 | Итер. 2 | Итер. 3 | | | … | | | Итер. m |

- Построение
  - уточнение базового уровня архитектуры
  - реализация всех вариантов использования
- Внедрение
  - бета-версия
  - тренинги сотрудников заказчиков
  - исправление дефектов

# Модели УП

Модели – наиболее важный тип артефактов. Каждая модель описывает систему с определенной точки зрения на определенном уровне абстракции.

- Вариантов использования
- Анализа
- Проектирования
- Развертывания
- Реализации
- Тестирования

Все модели связаны, они полностью описывают систему.

Набор моделей дает варианты обозрения системы для всех сотрудников.

# UML

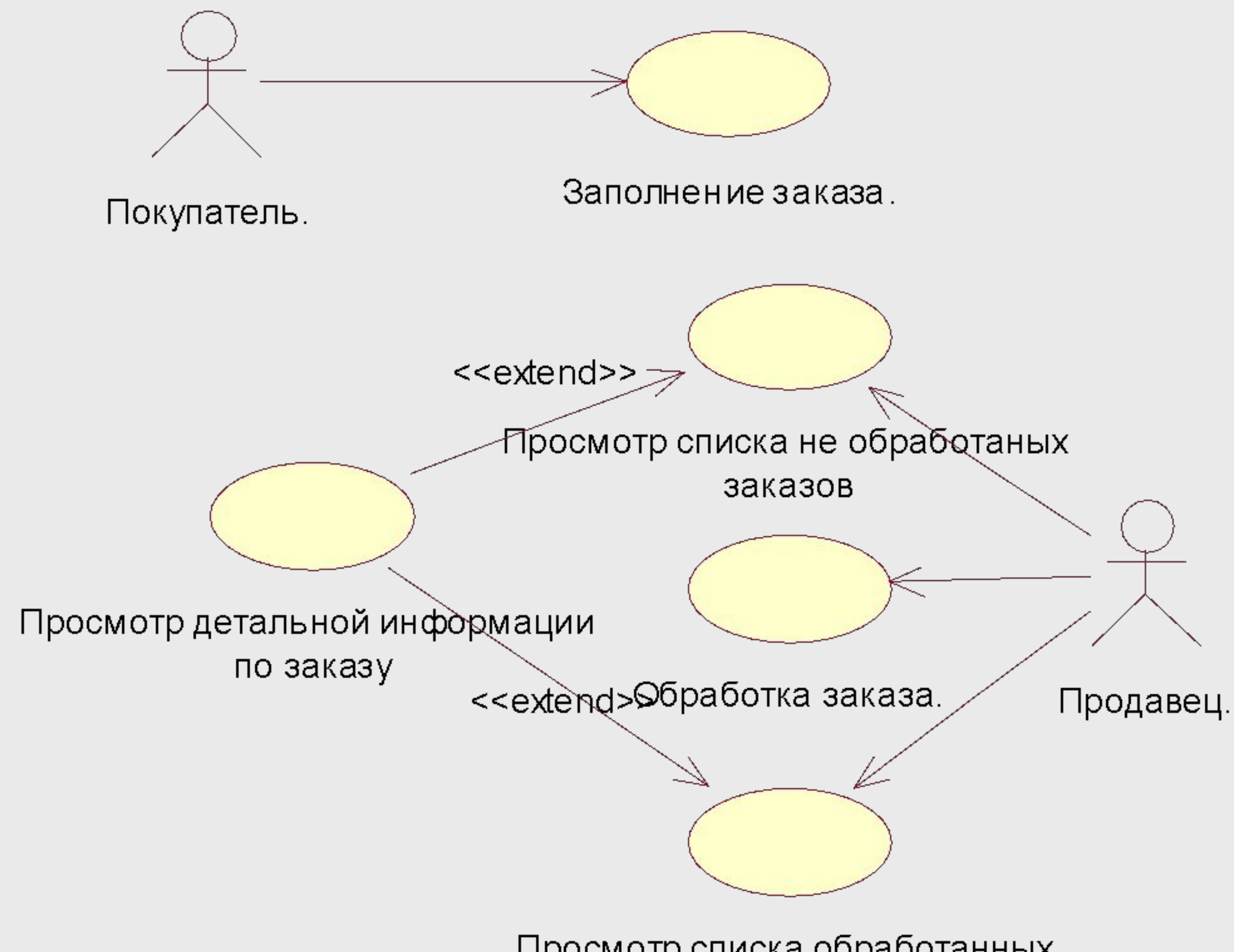
Язык для специфицирования, визуализации, конструирования и документирования программных продуктов.

Также используется в бизнес-моделировании и моделировании любых иных (не программных) систем.

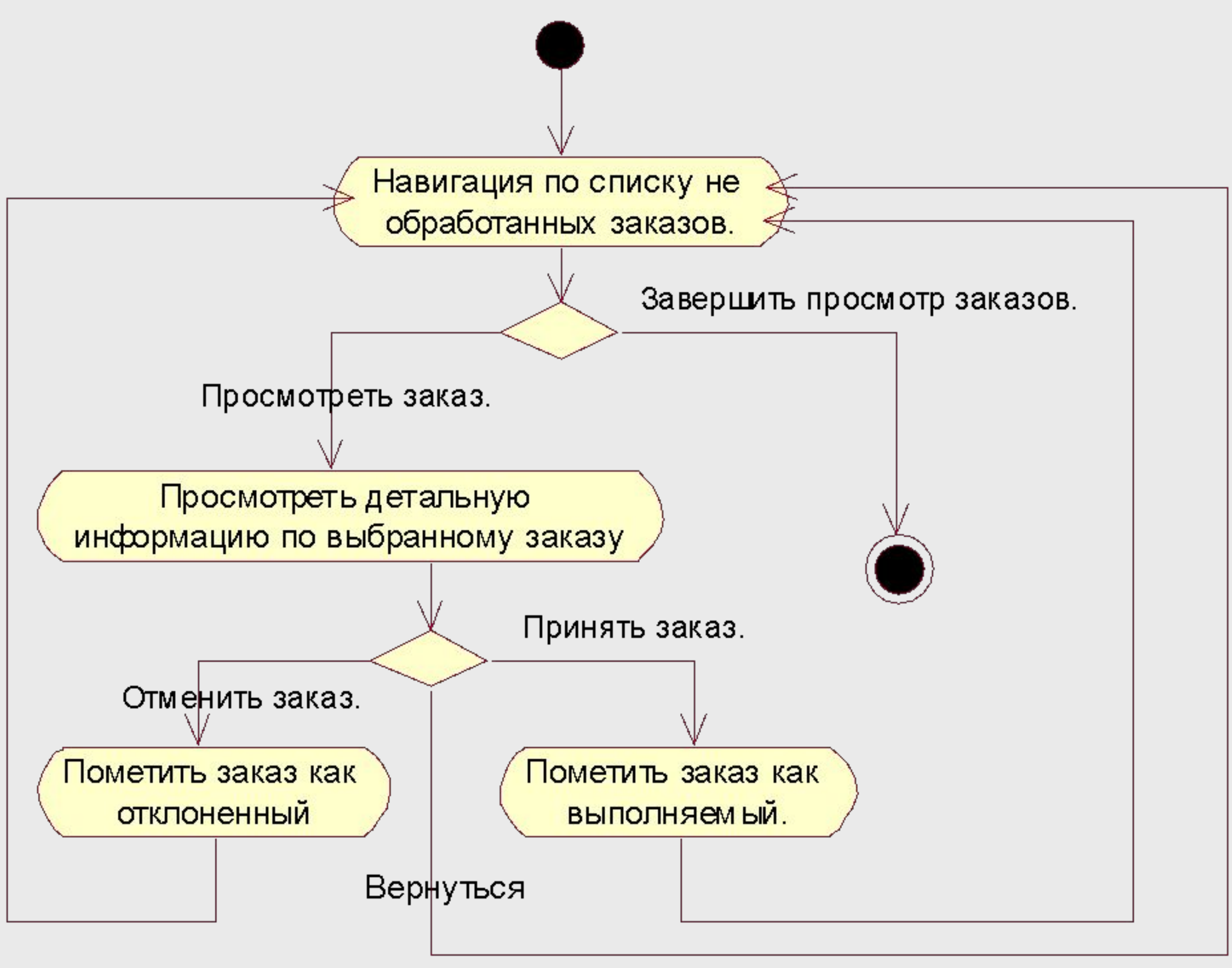
UML позволяет задавать следующие аспекты:

- Диаграммы вариантов использования (use case diagrams)
- Диаграммы классов (class diagrams)
- Диаграммы поведения
  - Диаграммы состояний (statechart diagrams)
  - Диаграммы действий (activity diagrams)
  - Диаграммы взаимодействия (interaction diagrams)
    - Диаграммы последовательностей(sequence diagrams)
    - Диаграммы взаимодействий(collaboration diagrams)
  - Диаграммы реализации (implementation diagrams)
    - Диаграммы компонент (component diagram)
    - Диаграммы развертывания (deployment diagram)

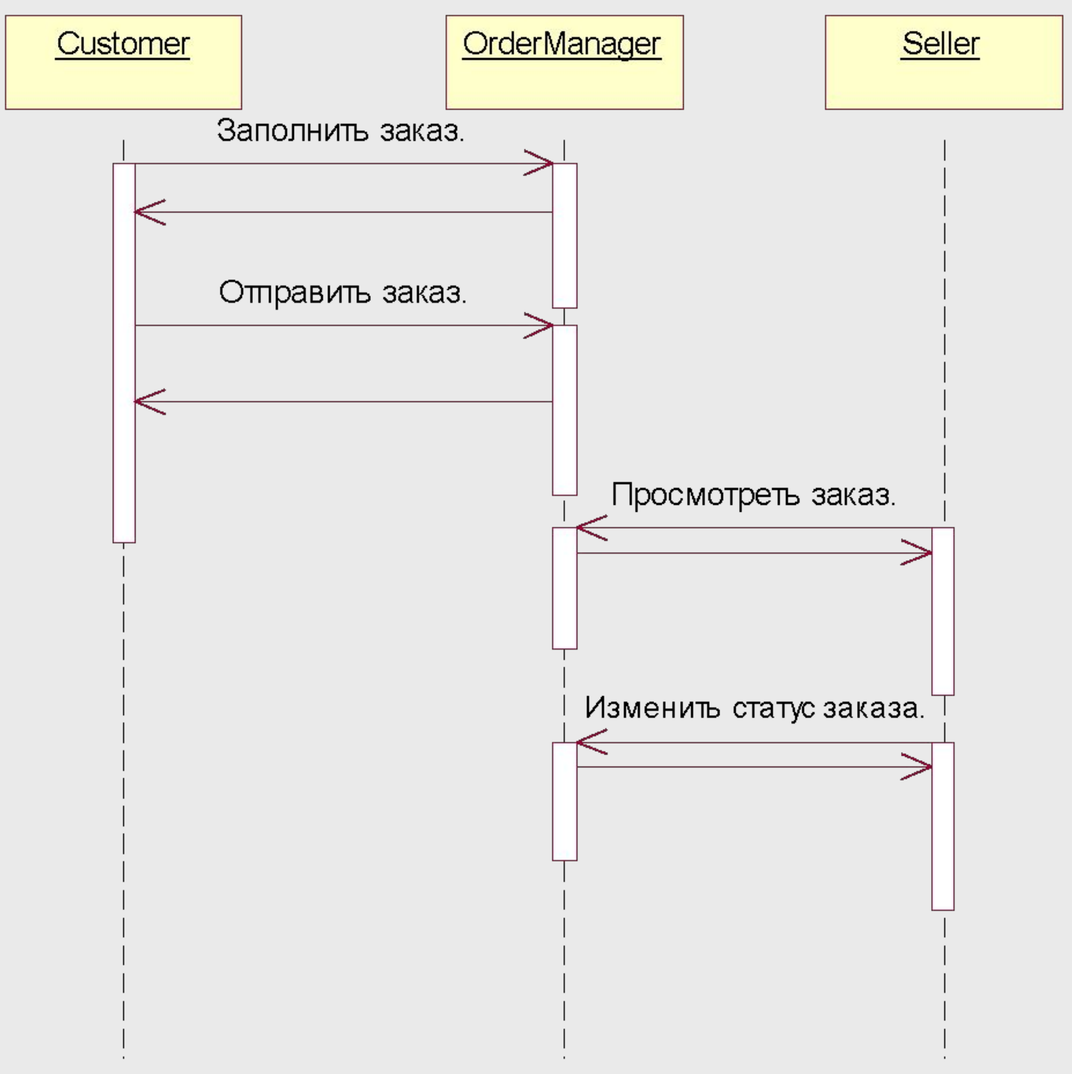
# Диаграммы вариантов использования (Use case diagrams)



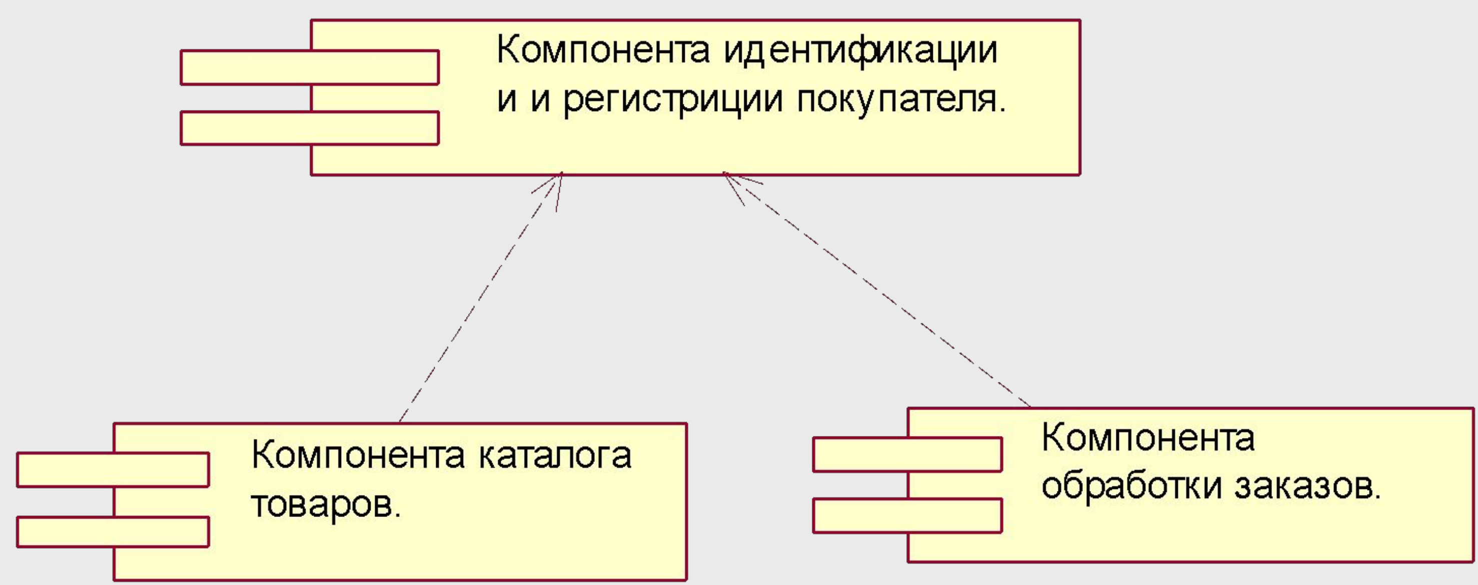
# Диаграммы деятельности (Activity diagrams)



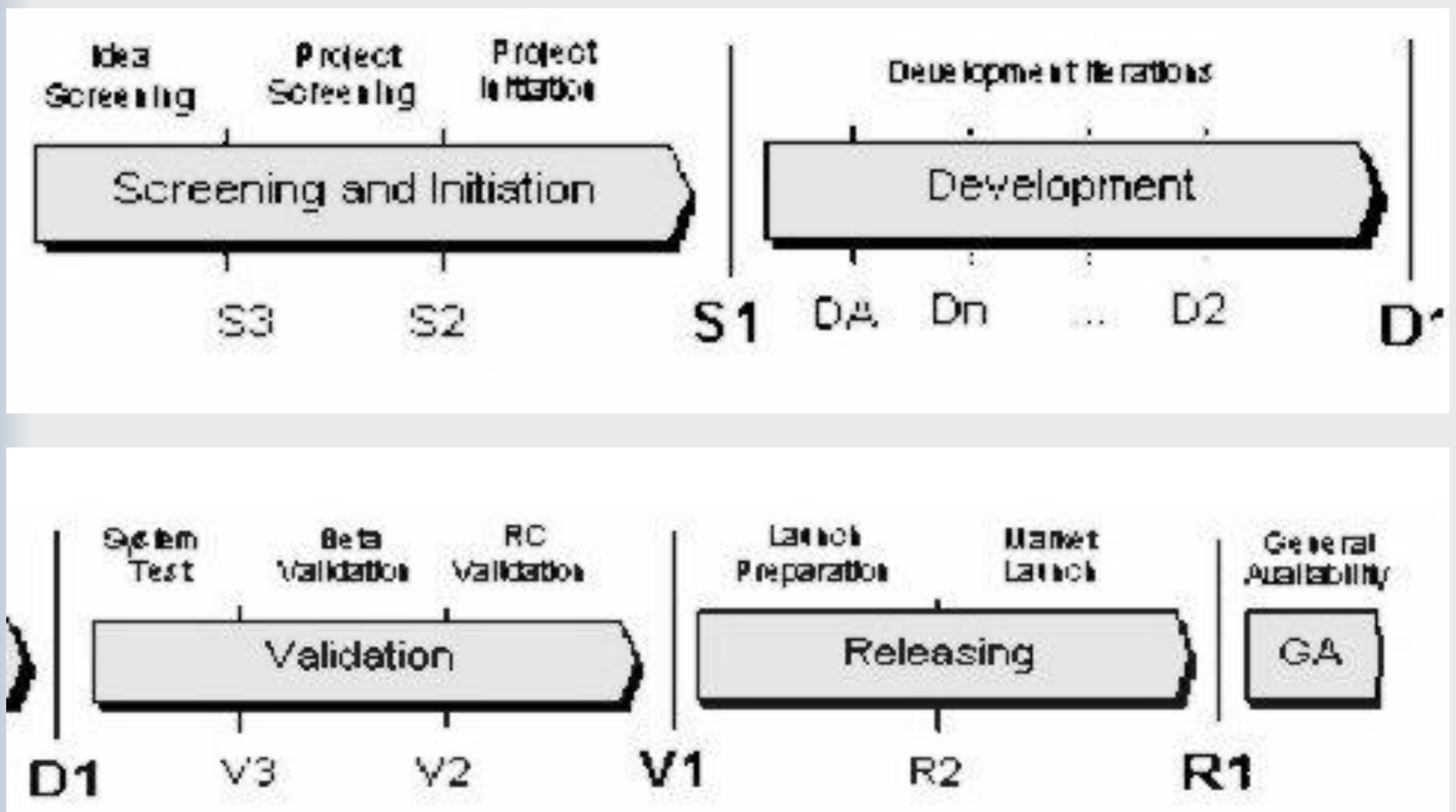
# Диаграммы последовательностей действий (Sequence diagrams)



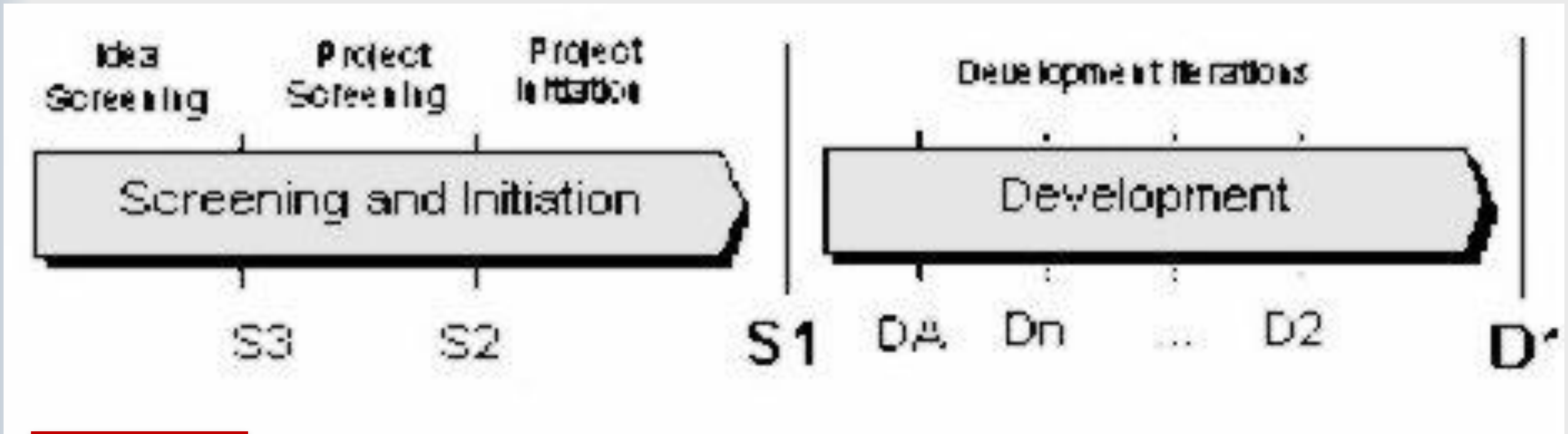
# Диаграммы компонент (Component diagrams)



# Пример реального процесса разработки ПО

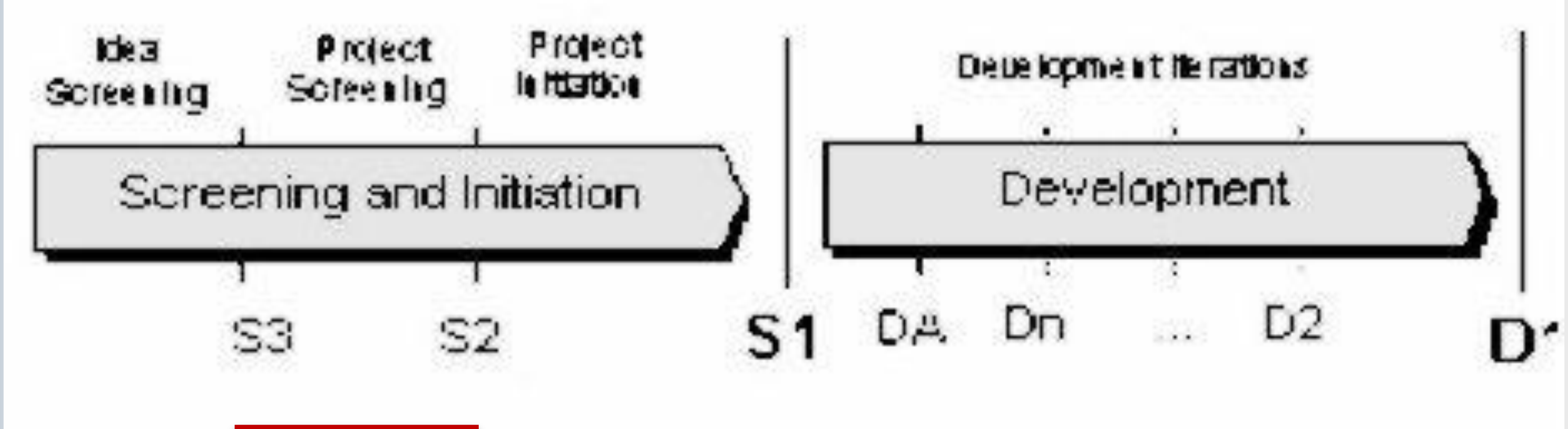


# Обзор идеи



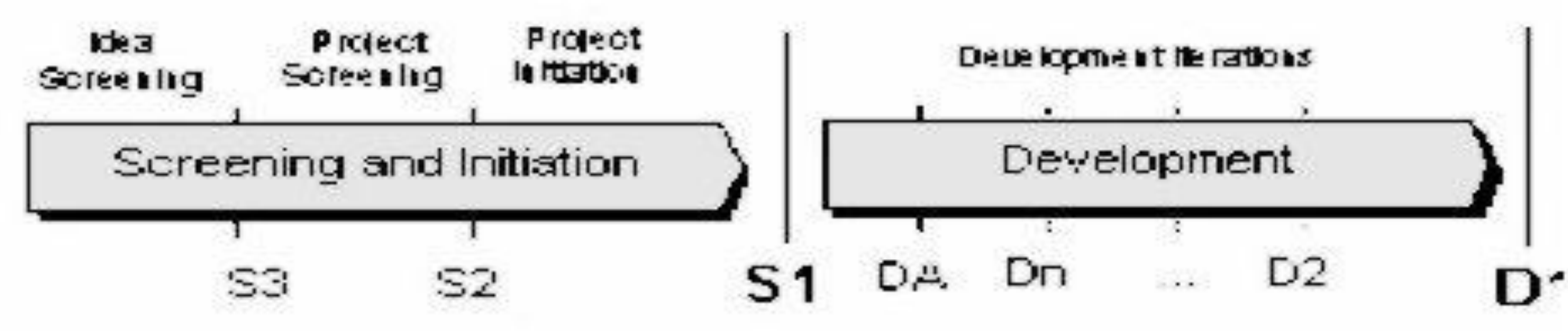

- Выдвигается идея нового продукта
- Назначается менеджер по продукту (PdM). Он оценивает идею и составляет ее краткий обзор, который направляет на утверждение HBU и HPdM.
- Назначается PjM
- Milestone **S3**: HBU или HPdM принимают решение о дальнейшем анализе бизнес-идеи

# Обзор проекта




- PjM назначает системного архитектора (SWA) и старшего тестера (CQA).
- PdM, PjM, представитель спонсора, SWA, CQA формируют руководящую группу (Steering Group), принимающую решения по проекту.
- SWA анализирует техническую возможность реализации.
- PjM составляет обзор по своему проекту.
- PjM составляет черновик плана проекта (Project Plan)
- PdM подготавливает отчет об анализе бизнес-идеи продукта.
- Milestone **S2**: HBU или HPdM дают добро на начало разработки проекта.

# Подготовка проекта




- PjM уточняет план проекта, назначает команду разработчиков, организует взаимодействие с другими отделами (документация, локализация, поддержка пользователей, технические тренинги и т.д.)
- PdM и SWA составляют список требований к программному продукту (Stakeholder Requirements):
  - Функциональность (Functionality), Удобство использования (Usability), Надежность (Reliability), Быстродействие (Performance), Безопасность (security), Обеспеченность поддержкой (Supportability)
- требования могут градуироваться по приоритетам: обязательно (must), желательно (should), возможно (may).
- SWA с SWE возможно создают прототип продукта
- StakeHolder Requirements – основной продукт по завершению фазы.
- Milestone **S1**: Product Council разрешает начать разработку продукта.

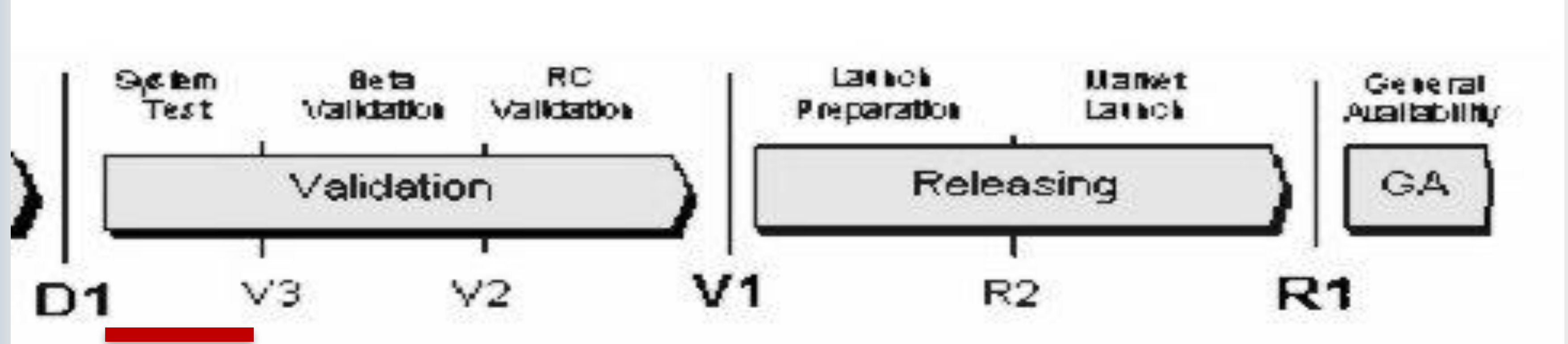
# Разработка продукта (Development) - 1

- SWA разрабатывает на утверждение SG дизайн продукта (Design Description) и спецификацию по Интерфейсу пользователя (UI description), проводит декомпозицию на модули, описывает все в удобном для разработки виде (напр. UML),
- PjM планирует сроки и расстановку сил по разработке каждого модуля
- CQA начинает подготовку Test Plan и Test Specification
- Тестовая спецификация строится с учетом требований. Она описывает методы тестирования, Test Cases, их важность и критерии проверки.
- Milestone **DA**: дизайн утверждается SG (Руководящей группой).

# Разработка продукта (Development) - 2

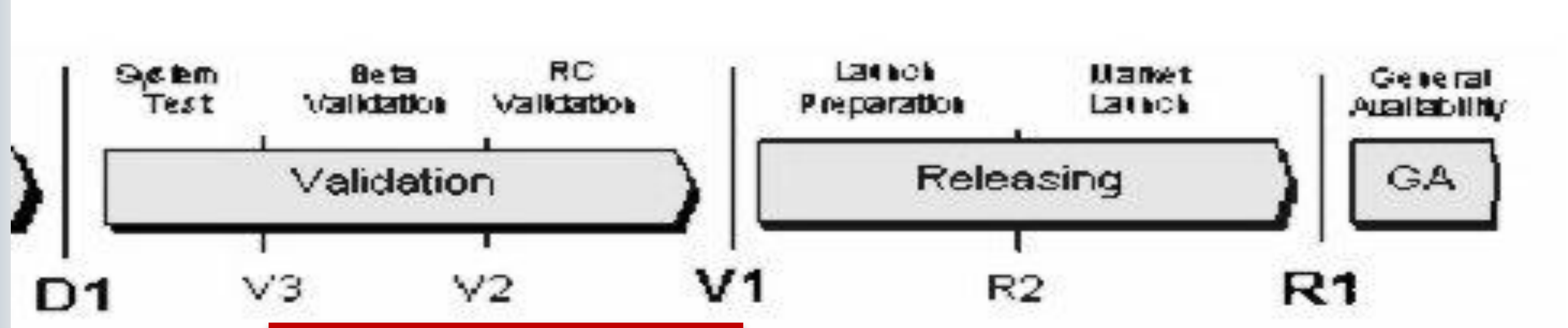
- Выполняется итеративно: анализ, дизайн, программирование, тестирование.
- Milestones Dn – D1: завершение билда N, …, 1.
- Milestone **D1**:
  - Фиксация - Code & feature freeze (alpha version)
  - Нет серьезных дефектов - No any urgent bugs
  - CQA подготовил тестовую спецификацию
  - Первая версия. TWriter подготовил черновик руководства пользователя
  - Продукт готов к системному тестированию.

# Альфа-тестирование




- Итеративное тестирование продукта тестерами под руководством CQA. Как только серьезных проблем больше не обнаруживается, продукт переходит в статус beta version.
- Milestone **V3**: product beta-version & draft of User Guide, нет серьезных проблем и отклонений от требований

# Бета-тестирование




- Продукт отсылается на ознакомление и тестирование ограниченному набору пользователей (User Support team, beta testers, sales engineers, external partners).

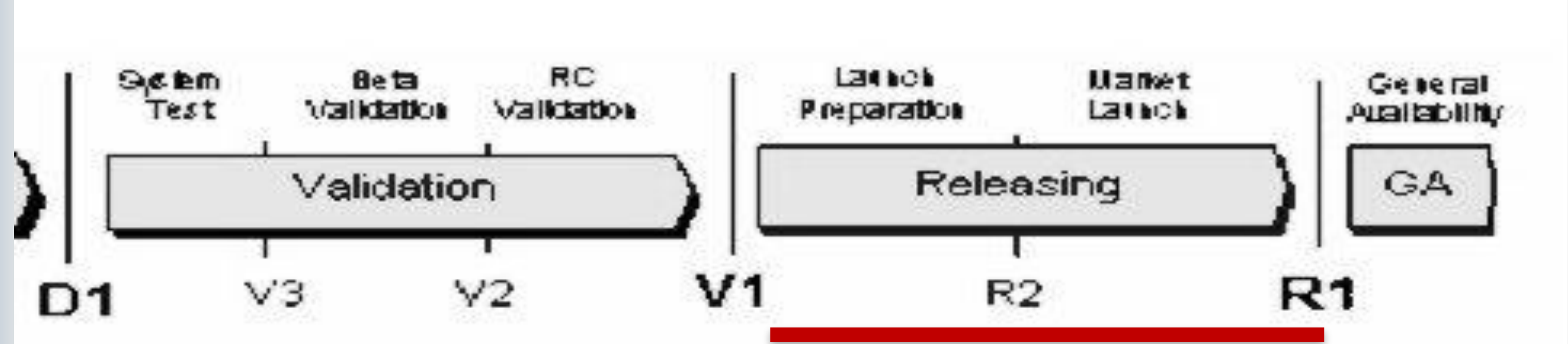
Milestone **V2**: готов Release Candidate, no any unresolved problems found.

Тестирование окончательной версии:

- Release candidate version отсылается избранным заказчикам.

Milestone **V1**: Руководящая группа принимает решение о том, что продукт готов к выходу.

# Подготовка к выпуску и выпуск




PdM и HPdM проверяют, что продукт готов к выходу на рынок (все собрано, документация подготовлена, отделы поддержки и тренинга готовы, реклама дана, произведена Интернет-подготовка, завод готов отштамповать диски, отдел доставки готов их доставить, определены цены, согласовано с продавцами, и т.п.).

- Milestone **R2**: все подготовлено и согласовано, назначена точная дата выхода.

Выпуск (**R2**)

- Продукт заливается на болванки, доставляется в магазины. Дается контрольная отмашка о выходе продукта в свет.

- все!

# CASE-технологии

- Computer Aided Software/System Engineering – автоматизированная разработка ПО/систем
- Существуют CASE-технологии, поддерживающие как структурный, так и объектный (в т. ч. компонентный) подход
- CASE-средства повышают производительность труда программистов и улучшают качество программного обеспечения. Они:
  - обеспечивают автоматизированный контроль совместимости спецификаций проекта;
  - уменьшают время создания прототипа системы;
  - ускоряют процесс проектирования и разработки;
  - автоматизируют формирование проектной документации для всех этапов жизненного цикла;
  - частично генерируют коды программ для различных платформ разработки;
  - поддерживают технологии повторного использования компонентов системы;
  - обеспечивают возможность восстановления проектной документации по имеющимся исходным кодам.

# Компонентный подход и CASE-технологии

- Компонентный подход предполагает построение программного обеспечения из отдельных компонентов — физически отдельно существующих частей программного обеспечения, которые взаимодействуют между собой через *стандартизованные двоичные интерфейсы*.
- В отличие от обычных объектов, объекты-компоненты можно собрать в динамически вызываемые библиотеки или исполняемые файлы, распространять в двоичном виде (без исходных текстов) и использовать в любом языке программирования, поддерживающем соответствующую технологию.
- Компонентный подход лежит в основе технологий, разработанных на базе COM и CORBA.

# Технологии COM

- Технология COM определяет *общий принцип взаимодействия программ любых типов:* библиотек, приложений, операционной системы, т. е. позволяет одной части программного обеспечения использовать функции *(службы),* предоставляемые другой, независимо от того, функционируют ли эти части в пределах одного процесса, в разных процессах на одном компьютере или на разных компьютерах. Модификация COM, обеспечивающая передачу вызовов между компьютерами, называется DCOM
- По технологии COM приложение предоставляет свои службы, используя *объекты* COM, которые являются экземплярами *классов COM.* Объект COM может реализовывать несколько интерфейсов.

# Технологии COM

На базе технологии COM были разработаны компонентные технологии, решающие различные задачи разработки программного обеспечения.

- ***OLE-automation*** — технология создания приложений, обеспечивающая доступ к их внутренним службам. Например, ее поддерживает Microsoft Excel, предоставляя другим приложениям свои службы.
- ***ActiveX*** — технология, построенная на базе OLE-automation, предназначена для создания как распределенного в сети, так и сосредоточенного на одном компьютере программного обеспечения. Предполагает использование визуального программирования для создания компонентов — элементов управления ActiveX. Полученные таким образом элементы управления можно устанавливать на компьютер дистанционно с удаленного сервера, причем устанавливаемый код зависит от используемой операционной системы.

# Технологии COM

- ***MTS*** (Microsoft Transaction Server — сервер управления транзакциями) — технология, обеспечивающая безопасность и стабильную работу распределенных приложений при больших объемах передаваемых данных.
- ***MIDAS*** (Multilier Distributed Application Server — сервер многозвенных распределенных приложений) — технология, организующая доступ к данным разных компьютеров с учетом балансировки нагрузки сети.

Все указанные технологии реализуют компонентный подход, заложенный в COM.

# Технология CORBA

- Технология CORBA, разработанная группой компаний OMG, реализует подход, аналогичный COM, на базе объектов и интерфейсов CORBA.
  Программное ядро CORBA реализовано для всех основных аппаратных и программных платформ и потому эту технологию можно использовать для создания распределенного программного обеспечения в разнородной вычислительной среде.
- Организация взаимодействия между объектами клиента и сервера в CORBA осуществляется с помощью специального посредника, названного VisiBroker, и другого специализированного программного обеспечения.

# ЛЕКЦИЯ 2

**Основные темы:**

- Анализ предметной области и требования к ПО
- Качество ПО и методы его контроля
- Тестирование
- Архитектура программного обеспечения

# АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Для выявления этих потребностей, а также для выяснения смысла сказанных требований приходится проводить достаточно большую дополнительную работу, которая называется **анализом предметной области** или **бизнес-моделированием** , если речь идет о потребностях коммерческой организации.

Анализом предметной области занимаются **системные аналитики** или **бизнес-аналитики.**

# СИСТЕМАТИЗАЦИЯ ИНФОРМАЦИИ О ПРЕДПРИЯТИИ

Для систематизации сбора информации о больших организациях и дальнйшей разработки систем, поддерживающих их деятельность, применяется схема Захманана(автор — John Zachman) или архитектурная схема предприятия (enterprise architecture framework).

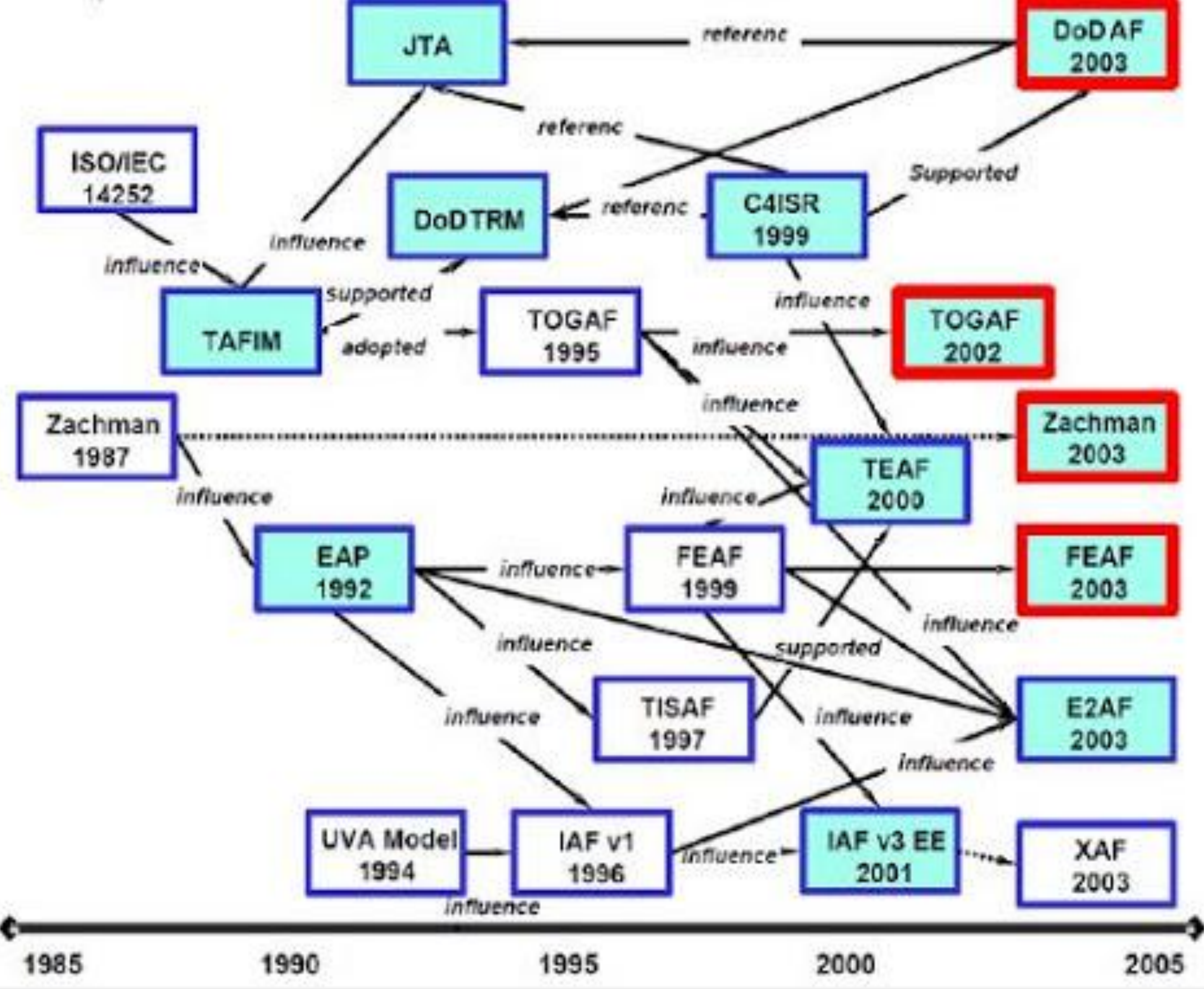
В основе ***схемы Захмана*** лежит следующая идея: *деятельность* даже очень большой организации можно описать, используя ответы на простые вопросы — зачем, кто, что, как, где и когда — и разные уровни рассмотрения. Обозначенные 6 вопросов определяют 6 аспектов рассмотрения.

- Цели организации и базовые правила, по которым она работает.
- Персонал, подразделения и другие элементы организационной структуры, связи между ними.
- Сущности и данные, с которыми имеет дело организация.
- Выполняемые организацией и различными ее подразделениями функции и операции над данными.
- Географическое распределение элементов организации и связи между географически разделенными ее частями.
- Временные характеристики и ограничения на деятельность организации, значимые для ее деятельности события.

# АРХИТЕКТУРА ОРГАНИЗАЦИИ - The Casewise framework.

## Миссия, Видение, Основные ценности

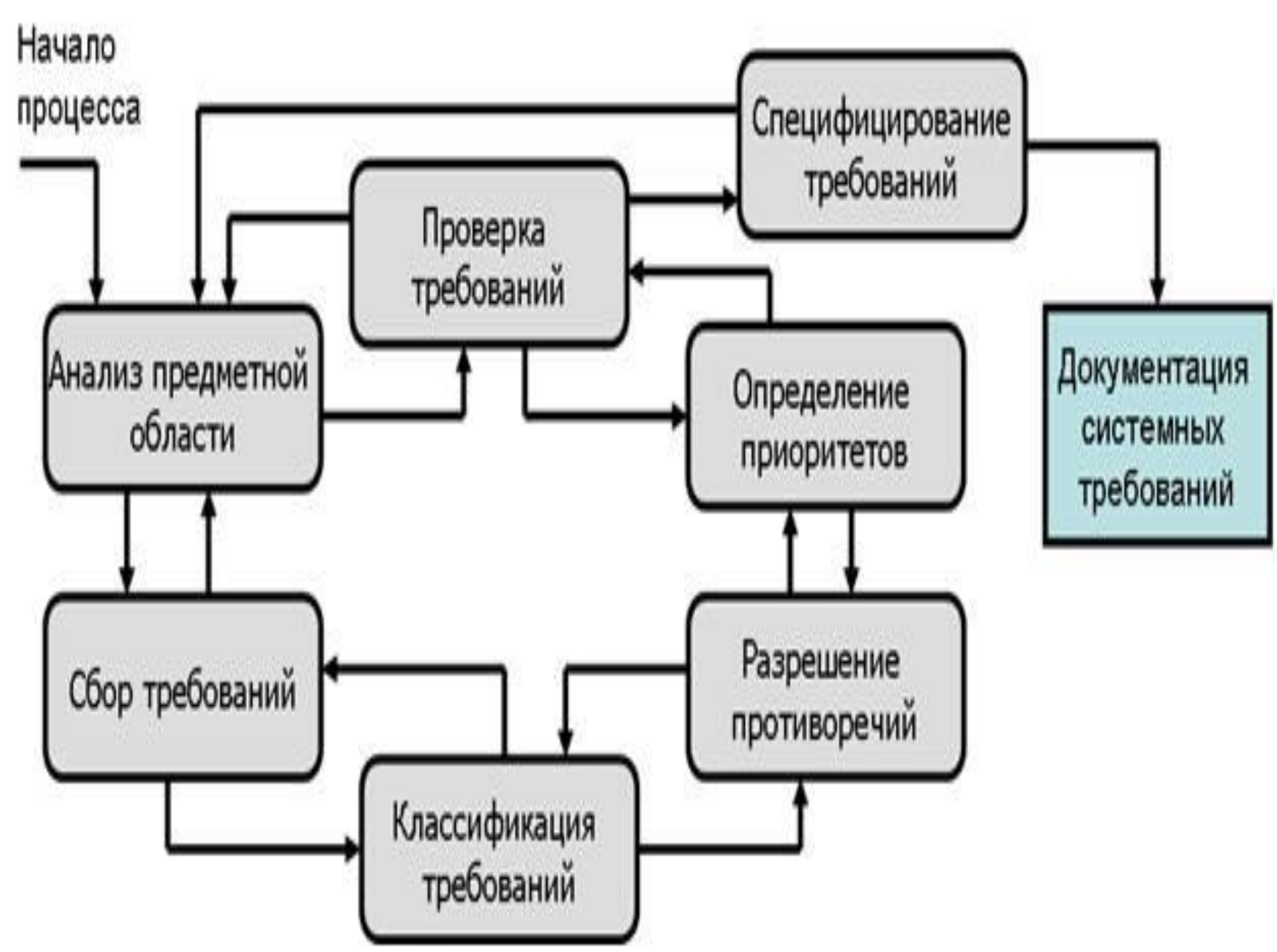
| | Мотивация [Почему] | Процессы [Как] | Люди [Кто] | Местоположения [Где] | Данные [Что] | Время [Когда] |
|---|---|---|---|---|---|---|
| **Область: Контекстуальный** [Планировщик] | Список стратегических бизнес-целей | Список бизнес-процессов верхнего уровня | Список основных типов организационных единиц | Список местоположений организации | Список важных для бизнеса данных | Список значимых для бизнеса событий |
| **Организация: Концептуальный** [Владелец] | Бизнес-план | (BDM) Модель динамики бизнеса | Организационная структура, основанная на ролях | Система бизнес-логистики | Концептуальная модель данных | Основной план / Временны ограничения |
| **Модель системы: Логический** [Проектировщик] | Отчёт о всех событиях и результатах уровня SDM | (SDM) Модель динамики системы | Фактическая организационная структура и матрицы | Логическая модель сети | Логическая модель данных | Имитация моделей динамики системы |
| **Модель технологий: Физический** [Разработчик] | Отчёт о всех событиях и результатах уровня FDM | (FDM) Модель динамики функций - связи с живыми системами | Матрицы "Орг. единицы / Процессы" | Детальная архитектура технологий | Физическая модель данных | Имитация моделей динамики функций |
| **Детальное представление: Вне контекста** [Субподрядчик] | Например, Почему всё физические системы уместны? | Например, компьютерные программы, инструкции ... | Например, HR-файлы и системы... | Например, фактическая сеть и местоположения... | Например, базы данных... | Например, восходящая цепочка поставо... |


JTA
referenc
DoDAF
2003
referenc
ISO/IEC
14252
Supported
DoDTRM
referenc
C4ISR
1999
influence
influence
supported
influence
TAFIM
adopted
TOGAF
1995
influence
TOGAF
2002
influence
Zachman
1987
Zachman
2003
TEAF
2000
influence
influence
EAP
1992
influence
FEAF
1999
FEAF
2003
influence
influence
supported
TISAF
1997
influence
influence
E2AF
2003
influence
UVA Model
1994
IAF v1
1996
influence
IAF v3 EE
2001
XAF
2003
influence
1985
1990
1995
2000
2005

# ФОРМИРОВАНИЕ ТРЕБОВАНИЙ

***Потребности, требования*** определяются на основе наиболее актуальных проблем и задач, которые пользователи и заказчики видят перед собой. При этом требуется аккуратное выявление значимых проблем, определение того, насколько хорошо они решаются при текущем положении дел, и расстановка приоритетов при рассмотрении недостаточно хорошо решаемых, поскольку чаще всего решить сразу все проблемы невозможно.

Имея набор функций, достаточно хорошо поддерживающий решение наиболее существенных задач, с которыми придется работать разрабатываемой системе, можно составлять *требования* к ней, представляющие собой детализацию работы этих функций. Соотношение между проблемами, потребностями, функциями и требованиями:


Начало
процесса
Спецификирование
требований
Проверка
требований
Анализ предметной
области
Определение
приоритетов
Документация
системных
требований
Сбор требований
Разрешение
противоречий
Классификация
требований

# СТАНДАРТЫ РАБОТЫ С ТРЕБОВАНИЯМИ ПО

**IEEE830-1998 Recommended Practice for Software Requirements Specifications**

Описывает структуру документов для фиксации требований к ПО. Кроме того, он определяет характеристики, которыми должен обладать правильно составленный набор требований.

o Корректность или адекватность (соответствие реальным потребностям).

o Недвусмысленность (однозначность понимания).

o Полнота (отражение всех выделенных потребностей и всех возможных ситуаций, в которых придется работать системе).

o Непротиворечивость (согласованность между различными элементами).

o Упорядоченность по важности и стабильности.

o Проверяемость (выполнение каждого требования нужно уметь проверять некоторым достаточно эффективным способом — непроверяемые требования должны быть удалены из рассмотрения или сведены к проверяемым вариантам).

o Модифицируемость (оформление в удобных для внесения изменений структуре и стилях).

o Прослеживаемость в ходе разработки (возможность увязать требование с подсистемами, модулями и операциям, ответственными за его выполнение, и с тестами, проверяющими его выполнение).

**IEEE 1233-1998, 2002 Guide for Developing System Requirements Specifications**

Описывает правила построения требований для программно-аппаратных систем в целом. Стандарт предписывает определять следующие атрибуты для каждого требования.

o Уникальный идентификатор.

o Приоритет, важность реализации с точки зрения пользователей.

o Осуществимость с точки зрения готовности пользователей к новой функции,

имеющихся технологий и стоимости реализации.

o Риски высокой стоимости, последствий использования для окружающей среды и пользователей, конфликтов со стандартами и законодательством.

o Источник (т.е. кто предложил это требование).

Тип требования. Возможные типы определяются так (многие из них соответствуют атрибутам качества, рассматриваемым в следующей лекции):

ƒ Требования на входные данные.

ƒ Требования на выходные данные.

Надежность (reliability, например, среднее время работы между отказами).

ƒ Работоспособность

ƒ Удобство сопровождения (maintainability, например, удобство замены компонента).

ƒ Производительность (performance, например, среднее время ожидания ответа).

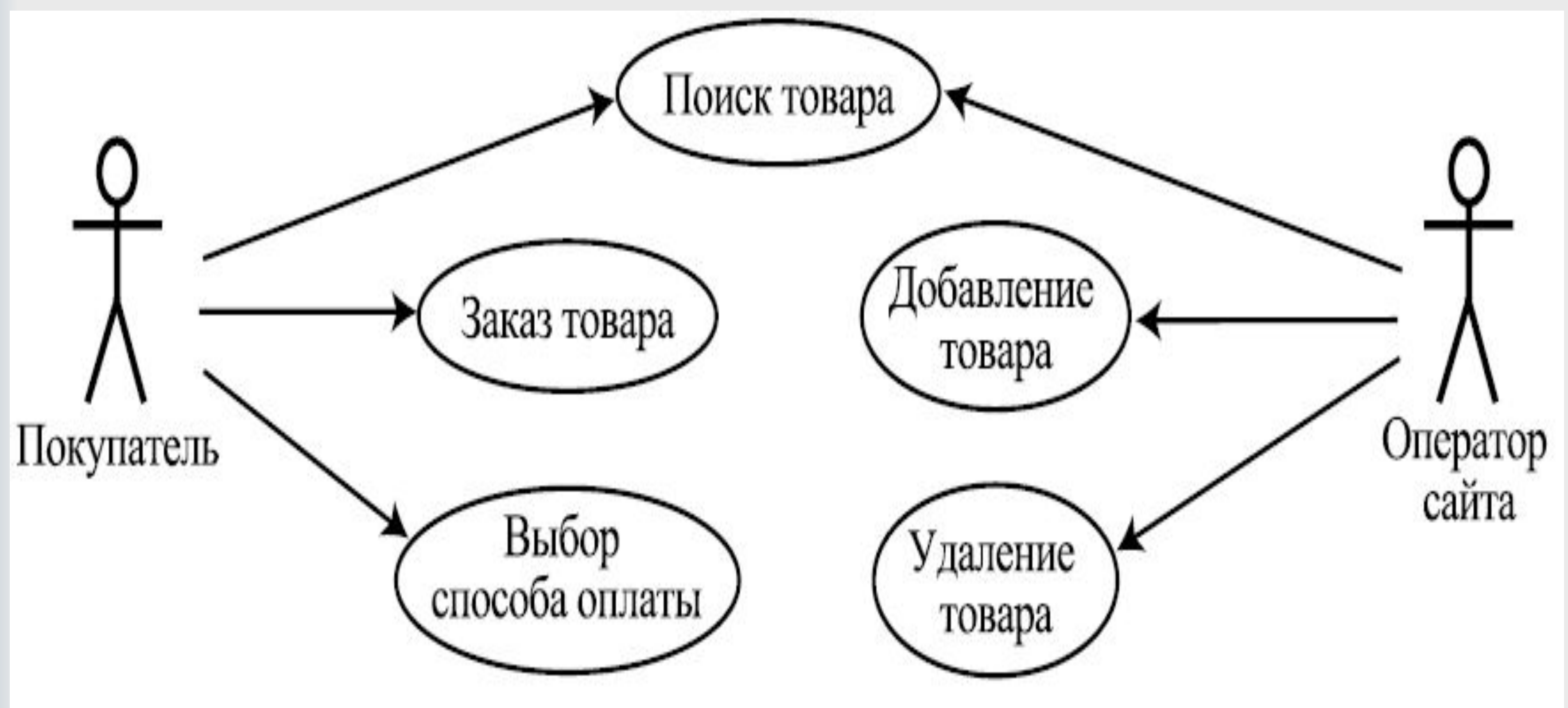
ƒ Доступность

# ТЕХНИКИ ФИКСАЦИИ ТРЕБОВАНИЙ

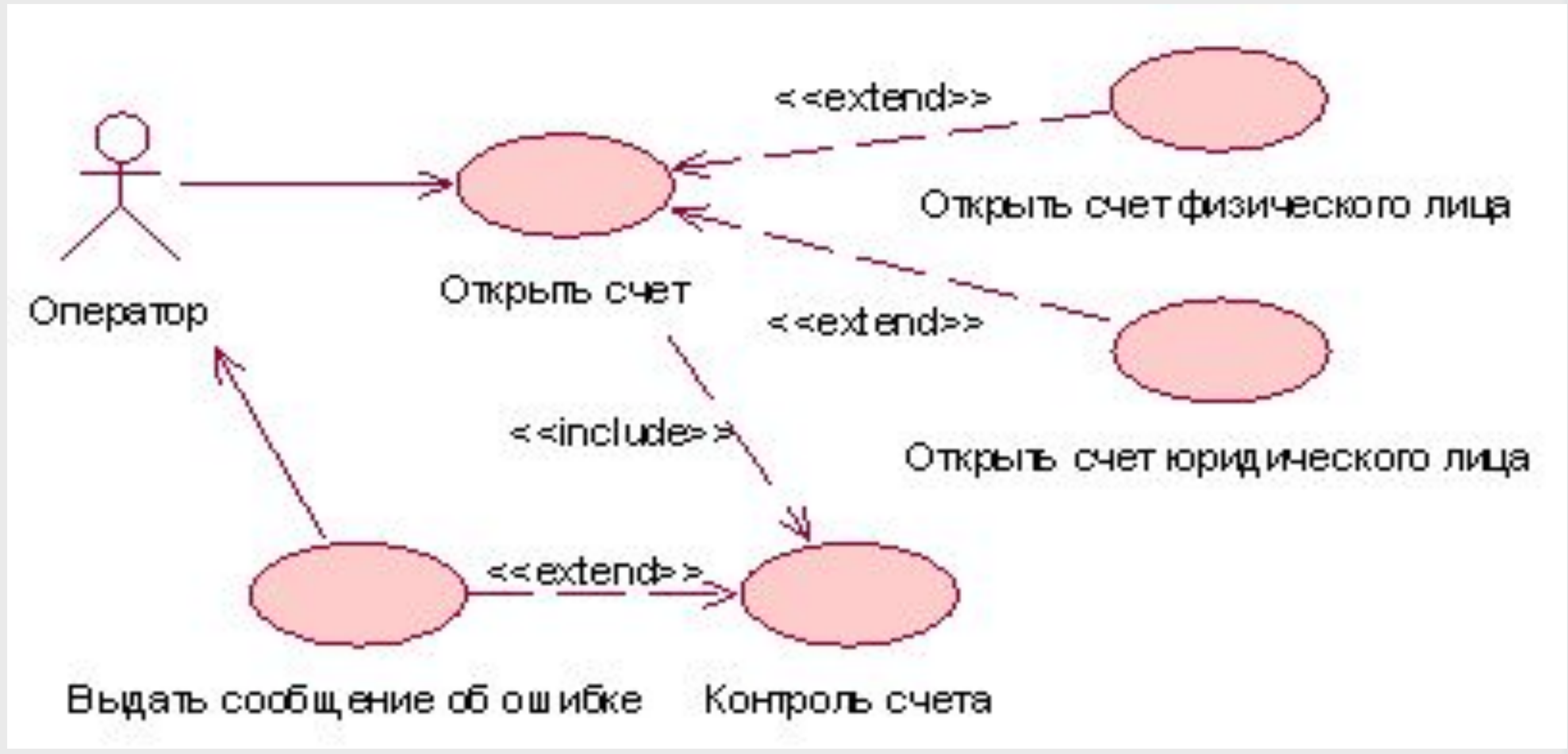
1. **Вариантом использования (use case)** называют некоторый сценарий действий системы, который обеспечивает ощутимый и значимый для ее пользователей результат.

На практике в виде одного варианта использования оформляется сценарий действий системы, который будет, скорее всего, неоднократно возникать во время ее работы и имеет достаточно четко определенные условия начала выполнения и завершения.

Варианты использования могут быть связаны друг с другом тремя видами связей: обобщением (generalization), расширением (extend relationship) и включением (include relationship). Действующие лица также могут быть связаны друг с другом с помощью связей обобщения (generalization)



Варианты использования могут быть связаны друг с другом тремя видами связей: обобщением (generalization), расширением (extend relationship) и включением (include relationship). Действующие лица также могут быть связаны друг с другом с помощью связей обобщения (generalization)



# КАЧЕСТВО ПО И МЕТОДЫ ЕГО КОНТРОЛЯ

Варианты ответов.

- Его легко использовать.
- Оно демонстрирует хорошую производительность.
- В нем нет ошибок.
- Оно не портит пользовательские данные при сбоях.
- Его можно использовать на разных платформах.
- Оно может работать 24 часа в сутки и 7 дней в неделю.
- В него легко добавлять новые возможности.
- Оно удовлетворяет потребности пользователей.
- Оно хорошо документировано.

Общие принципы обеспечения качества процессов производства во всех отраслях экономики регулируются набором стандартов ISO 9000. Наиболее важные для разработки ПО стандарты в его составе следующие:

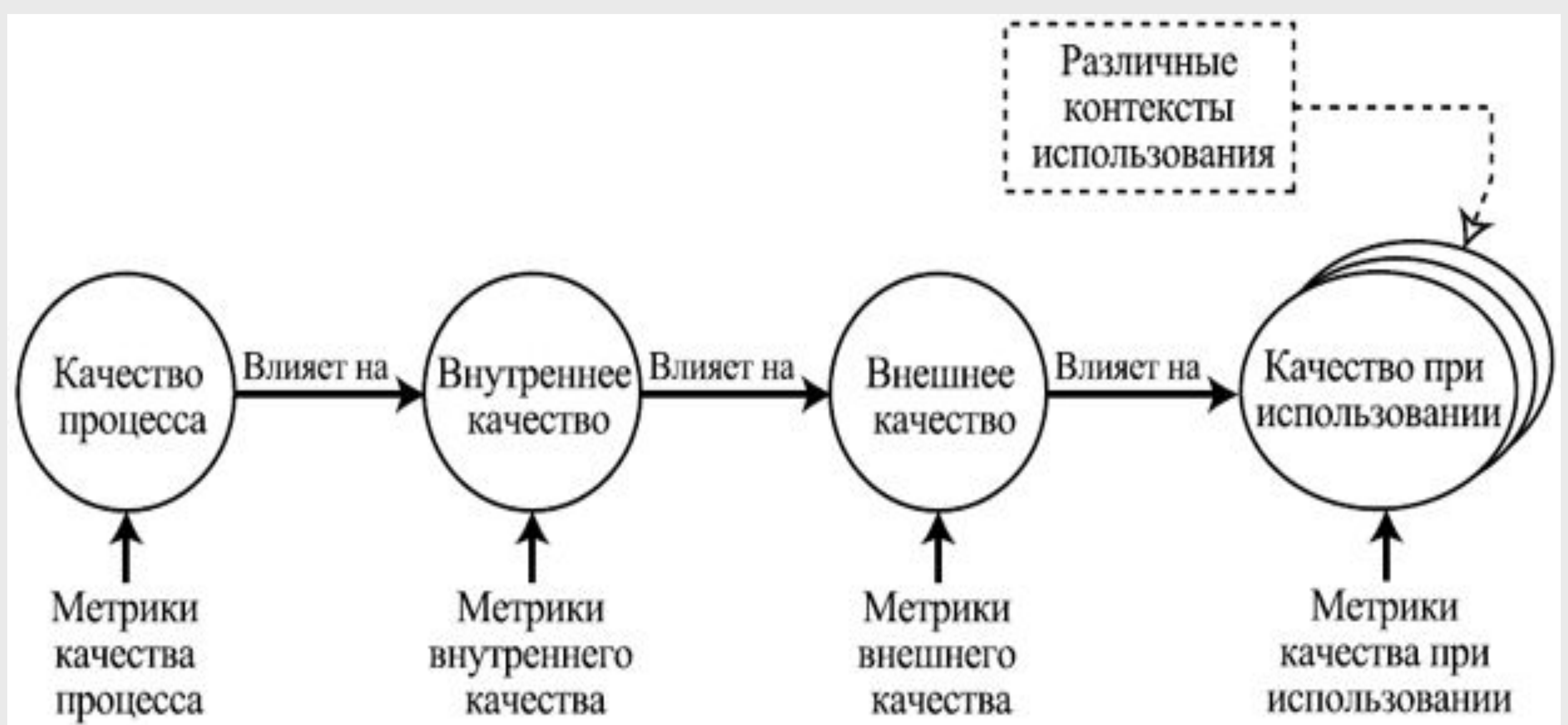
**ISO 9000:2000 Quality management systems — Fundamentals and vocabulary** Системы управления качеством — Основы и словарь. (Аналог — ГОСТ Р-2001).

**ISO 9001:2000 Quality management systems — Requirements. Models for quality assurance in design, development, production, installation, and servicing** Системы управления качеством — Требования. Модели для обеспечения качества при проектировании, разработке, коммерциализации, установке и обслуживании.

**ISO 9004:2000 Quality management systems — Guidelines for performance improvements** Системы управления качеством. Руководство по улучшению деятельности. (Аналог — ГОСТ Р-2001).

**ISO/IEC 90003:2004 Software engineering — Guidelines for the application of ISO 9001:2000 to computer software** Руководящие положения по применению стандарта *ISO 9001* при разработке, поставке и обслуживании программного обеспечения.

***Качество программного обеспечения*** определяется в стандарте ISO 9126 как вся совокупность его характеристик, относящихся к возможности удовлетворять высказанные или подразумеваемые потребности всех заинтересованных лиц.



Функциональность
Способность к взаимодействию
Функциональная пригодность
Соответствие стандартам
Защищенность
Точность
Переносимость
Адаптируемость
Удобство замены
Способность к сосуществованию
Соответствие стандартам
Удобство установки
Надежность
Зрелость
Способность к восстановлению
Соответствие стандартам
Устойчивость к отказам
Качество ПО
Удобство сопровождения
Удобство изменений
Соответствие стандартам
Удобство проверки
Анализируемость
Стабильность
Удобство использования
Соответствие стандартам
Удобство обучения
Привлекательность
Удобство работы
Понятность
Производительность
Соответствие стандартам
Временная эффективность
Эффективность использования ресурсов

***Функциональность (functionality)***Способность ПО в определенных условиях решать задачи, нужные пользователям. Определяет, что именно делает ПО, какие задачи оно решает.

**Функциональная пригодность (suitability)**.Способность решать нужный набор задач.

**Точность (accuracy)**.Способность выдавать нужные результаты.

**Способность к взаимодействию (interoperability)**. Способность взаимодействовать с нужным набором других систем.

**Соответствие стандартам и правилам (compliance)**. Соответствие ПО имеющимся индустриальным стандартам, нормативным и законодательным актам, другим регулирующим нормам.

**Защищенность (security)**.Способность предотвращать неавторизированный, т.е. без указания лица, пытающегося его осуществить, и неразрешенный доступ к данным и программам.

***Надежность (reliability)***.Способность ПО поддерживать определенную работоспособность в заданных условиях.

**Зрелость, завершенность (maturity)**.Величина, обратная частоте отказов ПО. Обычно измеряется средним временем работы без сбоев и величиной, обратной вероятности возникновения отказа за данный период времени.

**Устойчивость к отказам (fault tolerance)**.Способность поддерживать заданный уровень работоспособности при отказах и нарушениях правил взаимодействия с окружением.

**Способность к восстановлению (recoverability)**. Способность восстанавливать определенный уровень работоспособности и целостность данных после отказа, необходимые для этого время и ресурсы.

**Соответствие стандартам** *надежности* **(reliability compliance)**.Этот атрибут добавлен в 2001 году.

***Удобство использования* (usability) или практичность**. Способность ПО быть удобным в обучении и использовании, а также привлекательным для пользователей.

**Понятность (understandability)**.Показатель, обратный к усилиям, которые затрачиваются пользователями на восприятие основных понятий ПО и осознание их применимости для решения своих задач.

**Удобство обучения (learnability)**.Показатель, обратный усилиям, затрачиваемым пользователями на обучение работе с ПО.

**Удобство работы (operability)**.Показатель, обратный усилиям, предпринимаемым пользователями для решения своих задач с помощью ПО.

**Привлекательность (attractiveness)**.Способность ПО быть привлекательным для пользователей. Этот атрибут добавлен в 2001 году.

**Соответствие стандартам** *удобства использования* **(usability compliance)**.Этот атрибут добавлен в 2001 году.

***Производительность* (efficiency) или эффективность**. Способность ПО при заданных условиях обеспечивать необходимую работоспособность по отношению к выделяемым для этого ресурсам. Можно определить ее и как отношение получаемых с помощью ПО результатов к затрачиваемым на это ресурсам всех типов.

**Временная эффективность (time behaviour)**.Способность ПО выдавать ожидаемые результаты, а также обеспечивать передачу необходимого объема данных за отведенное время.

**Эффективность использования ресурсов (resource utilisation)**.Способность решать нужные задачи с использованием определенных объемов ресурсов определенных видов. Имеются в виду такие ресурсы, как оперативная и долговременная память, сетевые соединения, устройства ввода и вывода и пр.

**Соответствие стандартам** *производительности* **(efficiency compliance)**.Этот атрибут добавлен в 2001 году.

***Удобство сопровождения (maintainability)***.Удобство проведения всех видов деятельности, связанных с сопровождение программ.

**Анализируемость (analyzability) или удобство проведения анализа**.Удобство проведения анализа ошибок, дефектов и недостатков, а также удобство анализа необходимости изменений и их возможных последствий.

**Удобство внесения изменений (changeability)**. Показатель, обратный трудозатратам на выполнение необходимых изменений.

**Стабильность (stability)**.Показатель, обратный риску возникновения неожиданных эффектов при внесении необходимых изменений.

**Удобство проверки (testability)**.Показатель, обратный трудозатратам на проведение *тестирования* и других видов проверки того, что внесенные изменения привели к нужным результатам.

**Соответствие стандартам** *удобства сопровождения* **(maintainability compliance)**.Этот атрибут добавлен в 2001 году.

***Переносимость (portability)***.Способность ПО сохранять работоспособность при переносе из одного окружения в другое, включая организационные, аппаратные и программные аспекты окружения.

Иногда эта характеристика называется в русскоязычной литературе мобильностью. **Адаптируемость (adaptability)**.Способность ПО приспосабливаться различным окружениям без проведения для этого действий, помимо заранее предусмотренных.

**Удобство установки (installability)**.Способность ПО быть установленным или развернутым в определенном окружении.

**Способность к сосуществованию (coexistence)**.Способность ПО сосуществовать с другими программами в общем окружении, деля с ними ресурсы.

**Удобство замены (replaceability) другого ПО данным**. Возможность применения данного ПО вместо других программных систем для решения тех же задач в определенном окружении.

**Соответствие стандартам** *переносимости* **(portability compliance)**. Этот атрибут добавлен в 2001 году.

# МЕТОДЫ КОНТРОЛЯ КАЧЕСТВА

Ответы на эти вопросы можно получить с помощью процессов верификации и валидации.

**Верификация** обозначает проверку того, что ПО разработано в соответствии со всеми требованиями к нему, или что результаты очередного этапа разработки соответствуют ограничениям, сформулированным на предшествующих этапах.

**Валидация** — это проверка того, что сам продукт правилен, т.е. подтверждение того, что он действительно удовлетворяет потребностям и ожиданиям пользователей, заказчиков и других заинтересованных сторон.

***Методы контроля качества*** позволяют убедиться, что определенные характеристики *качества ПО* достигнуты. *Методы контроля качества ПО* можно классифицировать следующим образом:

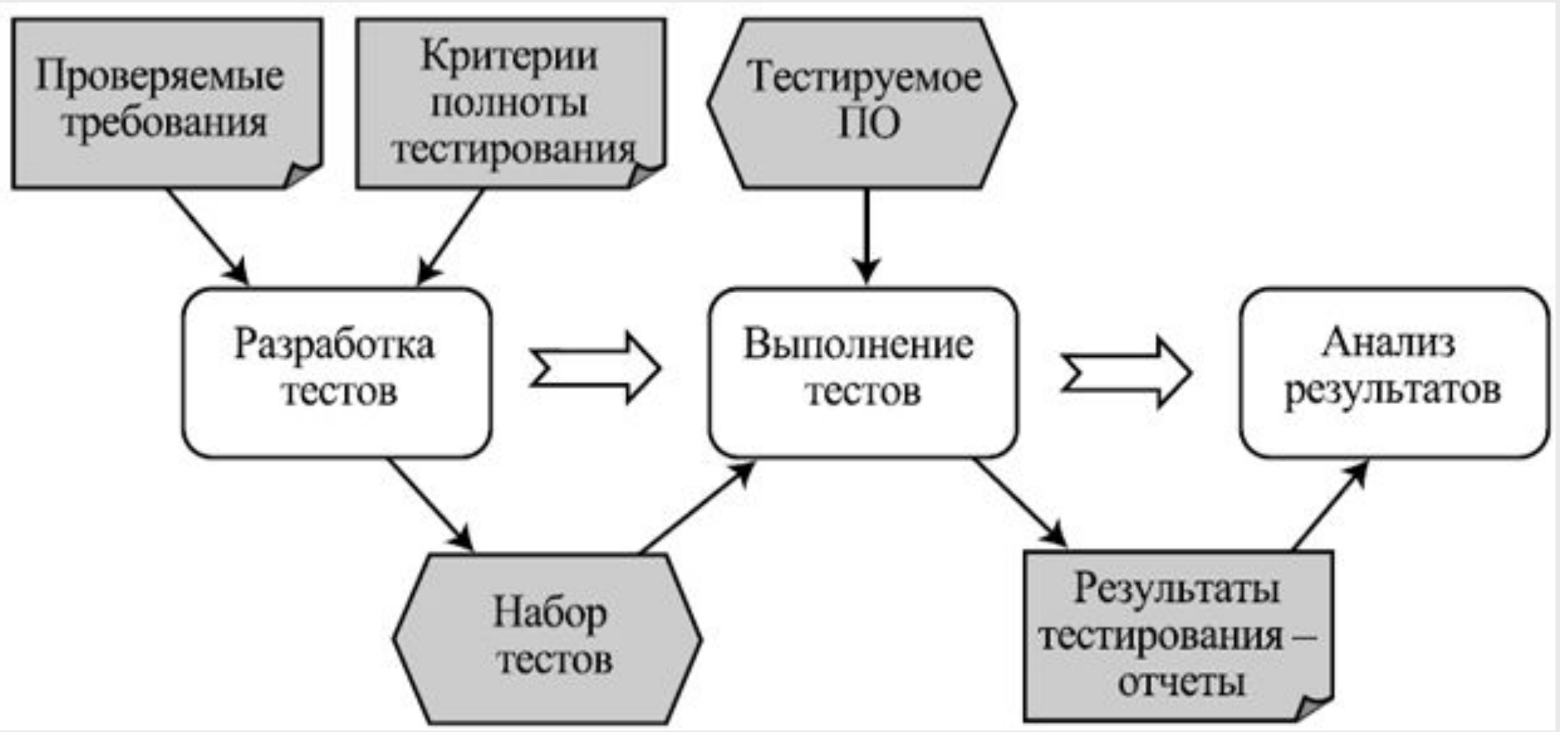
Методы и техники, связанные с выяснением свойств ПО во время его работы.Это, прежде всего, все виды *тестирования*, а также **профилирование** и измерение количественных показателей качества, которые можно определить по результатам работы ПО — эффективности по времени и другим ресурсам, *надежности*, доступности и пр.

Методы и техники определения показателей качества на основе симуляции работы ПО с помощью моделей разного рода.К этому виду относятся **проверка на моделях (model checking)**, а также **прототипирование (макетирование)**, используемое для оценки качества принимаемых решений.

Методы и техники, нацеленные на выявление нарушений формализованных правил построения исходного кода ПО, проектных моделей и документации. К методам такого рода относится **инспектирование кода**, заключающееся в целенаправленном поиске определенных дефектов и нарушений требований в коде на основе набора шаблонов, автоматизированные *методы поиска ошибок* в коде.

Методы и техники обычного или формализованного анализа проектной документации и исходного кода для выявления их свойств.К этой группе относятся многочисленные **методы анализа архитектуры ПО**.

***Тестирование*** — это проверка соответствия ПО требованиям, осуществляемая с помощью наблюдения за его работой в специальных, искусственно построенных ситуациях. Такого рода ситуации называют тестовыми или просто **тестами**.



***Тестирование*** — наиболее широко применяемый *метод контроля качества*. Для оценки многих атрибутов качества не существует других эффективных способов, кроме *тестирования*.

Организация *тестирования* ПО регулируется следующими стандартами:

IEEE 829-1998 Standard for Software Test Documentation. Описывает виды документов, служащих для подготовки тестов.

IEEE 1008-1987 (R1993, R2002) Standard for Software *Unit Testing*.Описывает организацию модульного *тестирования* (см. ниже).

ISO/IEC 12119:1994 (аналог AS/NZS 4366:1996 и ГОСТ Р-2000, также принят IEEE под номером IEEE 1465-1998) Information Technology. *Software packages* — Quality requirements and testing. Описывает требования к процедурам *тестирования* программных систем.

Выделяют виды *тестирования*, связанные с проверкой определенных характеристик и атрибутов качества — *тестирование функциональности*, *надежности*, *удобства использования*, *переносимости* и *производительности*, а также *тестирование* защищенности, функциональной пригодности и пр. Кроме того, особо выделяют **нагрузочное** или **стрессовое** *тестирование*, проверяющее работоспособность ПО и показатели его *производительности* в условиях повышенных нагрузок — при большом количестве пользователей, интенсивном обмене данными с другими системами, большом объеме передаваемых или используемых данных и пр.

***Проверка свойств на моделях (model checking)*** — проверка соответствия ПО требованиям при помощи формализации проверяемых свойств, построения формальных моделей проверяемого ПО (чаще всего в виде автоматов различных видов) и автоматической проверки выполнения этих свойств на построенных моделях. *Проверка свойств на моделях* позволяет проверять достаточно сложные свойства автоматически, при минимальном участии человека. Однако она оставляет открытым вопрос о том, насколько выявленные свойства модели можно переносить на само ПО.

Обычно при помощи *проверки свойств на моделях* анализируют два вида свойств алгоритмов, использованных при построении ПО. **Свойства безопасности (safety properties)** утверждают, что нечто нежелательное никогда не случится в ходе работы ПО. **Свойства живучести (liveness properties)** утверждают, наоборот, что нечто желательное при любом развитии событий произойдет в ходе его работы.

***Ошибками в ПО***, вообще говоря, являются все возможные несоответствия между демонстрируемыми характеристиками его качества и сформулированными или подразумеваемыми требованиями и ожиданиями пользователей.

В англоязычной литературе используется несколько терминов, часто одинаково переводящихся как "ошибка" на русский язык:

**defect** — самое общее нарушение каких-либо требований или ожиданий, не обязательно проявляющееся вовне (к дефектам относятся нарушения стандартов кодирования, недостаточная гибкость системы и пр.).

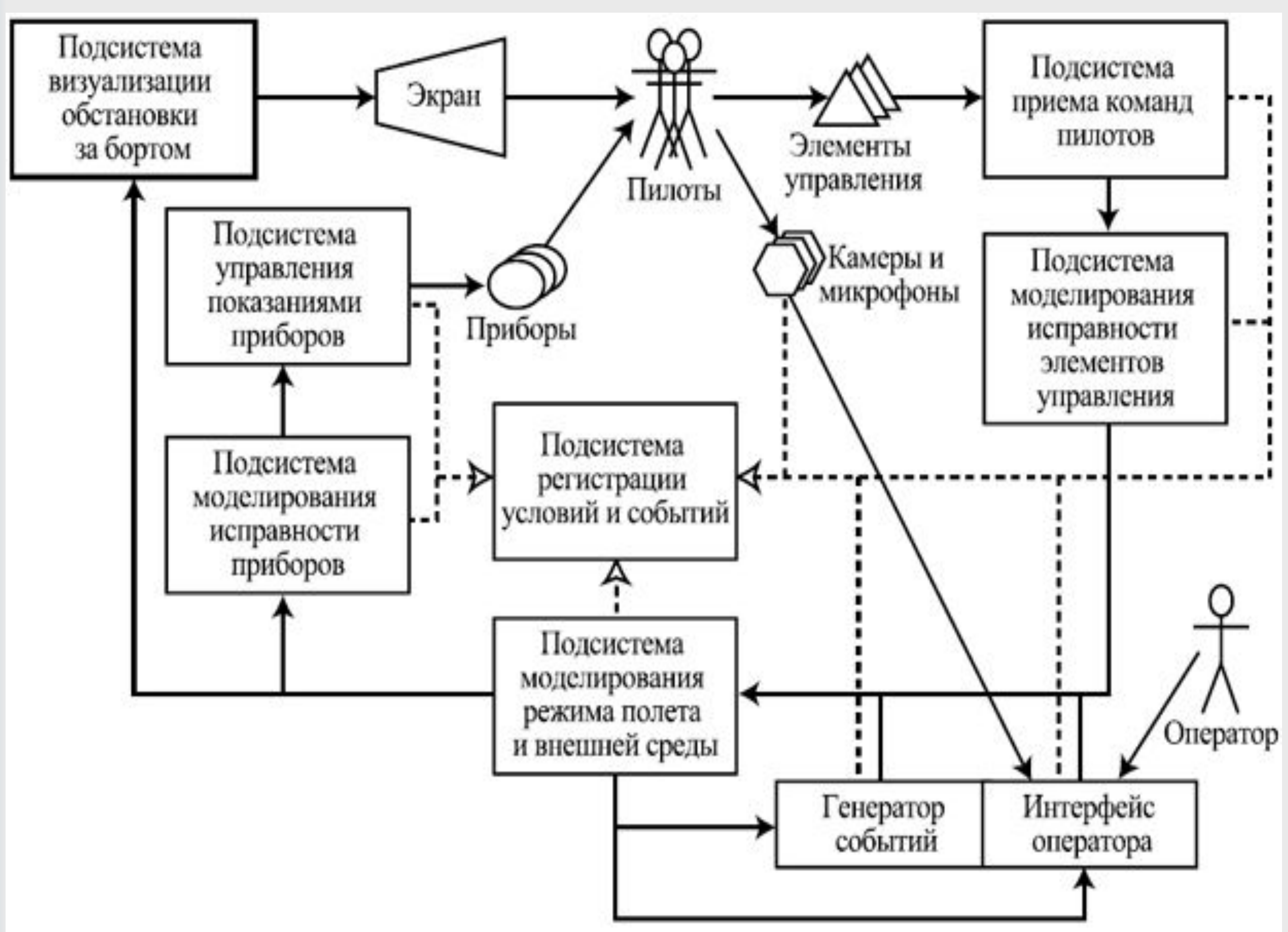
**failure** — наблюдаемое нарушение требований, проявляющееся при каком-то реальном сценарии работы ПО. Это можно назвать проявлением ошибки.

**fault** — ошибка в коде программы, вызывающая нарушения требований при работе (failures), то место, которое надо исправить. Хотя это понятие используется довольно часто, оно, вообще говоря, не вполне четкое, поскольку для устранения нарушения можно исправить программу в нескольких местах. Что именно надо исправлять, зависит от дополнительных условий, выполнение которых мы хотим при этом обеспечить, хотя в некоторых ситуациях наложение дополнительных ограничений не устраняет неоднозначность.

**error** — используется в двух смыслах.

# АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Под ***архитектурой ПО*** понимают набор внутренних структур *ПО*, которые видны с различных точек зрения и состоят из *компонентов*, их связей и возможных взаимодействий между *компонентами*, а также доступных извне свойств этих *компонентов*



Подсистема визуализации обстановки за бортом
Экран
Пилоты
Элементы управления
Подсистема приема команд пилотов
Подсистема управления показаниями приборов
Приборы
Камеры и микрофоны
Подсистема моделирования исправности элементов управления
Подсистема моделирования исправности приборов
Подсистема регистрации условий и событий
Подсистема моделирования режима полета и внешней среды
Оператор
Генератор событий
Интерфейс оператора

*Список* стандартов, регламентирующих описание архитектуры, которое является основной составляющей проектной документации на *ПО*, выглядит так:

**IEEE 1016-1998 Recommended Practice for Software Design Descriptions** (рекомендуемые методы описаний проектных решений для ПО).

**IEEE 1471-2000 Recommended Practice for Architectural Description of Software-Intensive Systems** (рекомендуемые методы описания архитектуры программных систем).

Стандарт *IEEE 1471* отмечает необходимость использования архитектуры системы для решения таких задач, как следующие:

Анализ альтернативных проектов системы.

Планирование перепроектирования системы, внесения изменений в ее организацию.

Общение по поводу системы между различными организациями, вовлеченными в ее разработку, эксплуатацию, сопровождение, приобретающими систему или продающими ее.

Выработка критериев приемки системы при ее сдаче в эксплуатацию.

Разработка документации по ее использованию и сопровождению, включая обучающие и маркетинговые материалы.

Проектирование и разработка отдельных элементов системы.

Сопровождение, эксплуатация, управление конфигурациями и внесение изменений и поправок.

Планирование бюджета и использования других ресурсов в проектах, связанных с разработкой, сопровождением или эксплуатацией системы.

Проведение обзоров, анализ и оценка качества системы.

# РАЗРАБОТКА И ОЦЕНКА АРХИТЕКТУРЫ НА ОСНОВЕ СЦЕНАРИЕВ

- Выделение компонентов

Выбирается набор "основных" *сценариев использования* — наиболее существенных и выполняемых чаще других.

Исходя из опыта проектировщиков, выбранного *архитектурного стиля* (см. следующую лекцию) и требований к переносимости и *удобству сопровождения* системы определяются компоненты, отвечающие за определенные действия в рамках этих *сценариев*, т.е. за решение определенных подзадач.

Каждый *сценарий использования* системы представляется в виде последовательности обмена сообщениями между полученными компонентами.

При возникновении дополнительных хорошо *выделенных подзадач* добавляются новые компоненты, и сценарии уточняются.

# РАЗРАБОТКА И ОЦЕНКА АРХИТЕКТУРЫ НА ОСНОВЕ СЦЕНАРИЕВ

- Определение интерфейсов компонентов

Для каждого компонента в результате выделяется его интерфейс — набор сообщений, которые он принимает от других компонентов и посылает им.

Рассматриваются "неосновные" сценарии, которые так же разбиваются на последовательности обмена сообщениями с использованием, по возможности, уже определенных интерфейсов.

Если интерфейсы недостаточны, они расширяются.

Если интерфейс компонента слишком велик, или компонент отвечает за слишком многое, он разбивается на более мелкие.

# РАЗРАБОТКА И ОЦЕНКА АРХИТЕКТУРЫ НА ОСНОВЕ СЦЕНАРИЕВ

- Уточнение набора компонентов

  Там, где это необходимо в силу требований эффективности или *удобства сопровождения*, несколько компонентов могут быть объединены в один.

  Там, где это необходимо для *удобства сопровождения* или надежности, один компонент может быть разделен на несколько.

- Достижение нужных свойств.

Все это делается до тех пор, пока не выполнятся следующие условия:

  Все *сценарии использования* реализуются в виде последовательностей обмена сообщениями между компонентами в рамках их интерфейсов.

  Набор компонентов достаточен для обеспечения всей нужной функциональности, удобен для сопровождения или портирования на другие платформы и не вызывает заметных проблем производительности.

  Каждый компонент имеет небольшой и четко очерченный круг решаемых задач и строго определенный, сбалансированный по размеру интерфейс.

# UML. ВИДЫ ДИАГРАММ UML

Для *представления архитектуры*, а, точнее, различных входящих в нее структур, удобно использовать графические языки. На настоящий момент наиболее проработанным и наиболее широко используемым из них является **унифицированный язык моделирования (Unified Modeling Language, UML)**

Диаграммы *UML* делятся на две группы — **статические** и **динамические диаграммы**.

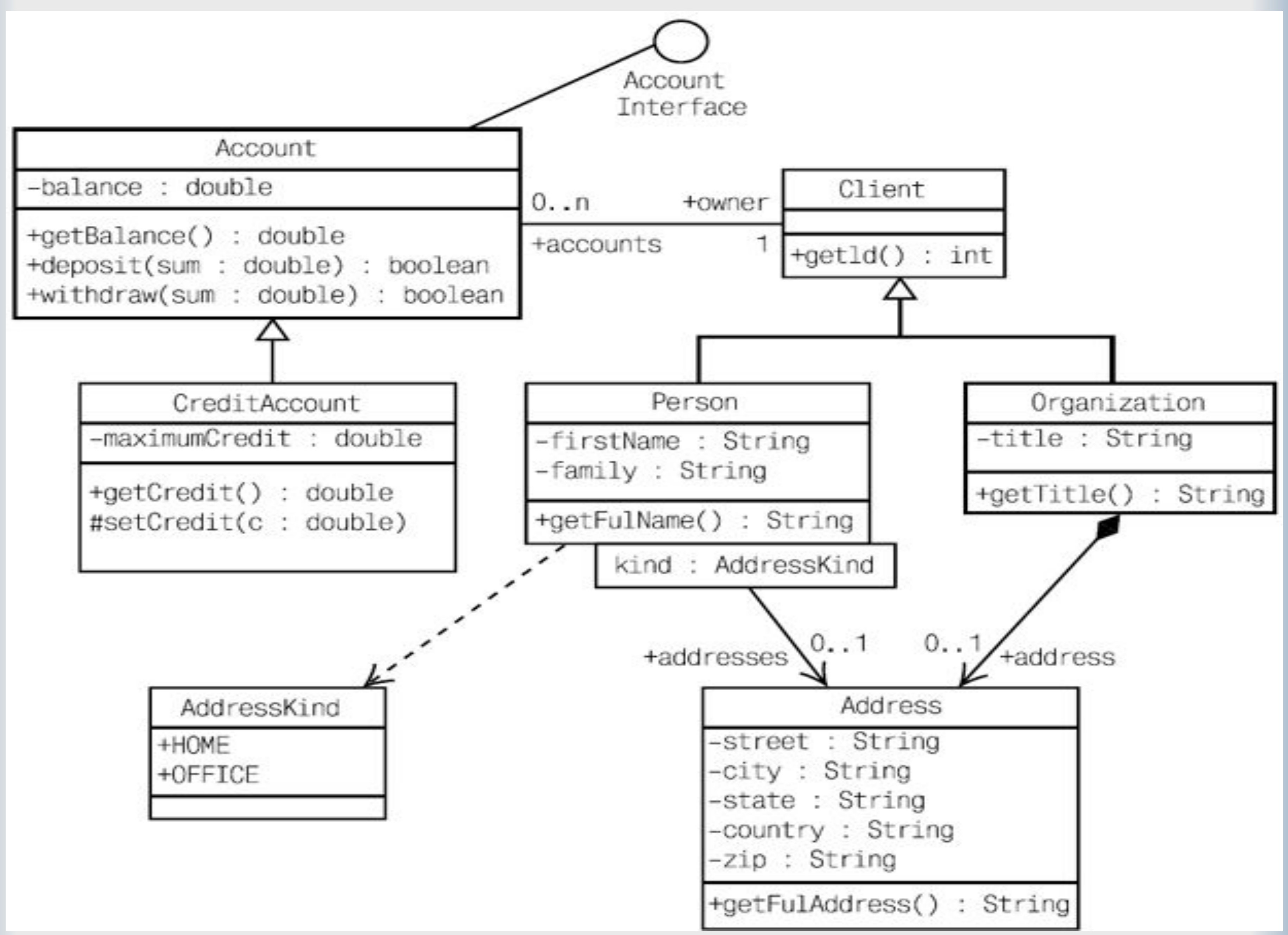
*UML* предлагает использовать для описания архитектуры 8 видов диаграмм. 9-й вид *UML* диаграмм, *диаграммы вариантов использования* , не относится к *архитектурным представлениям*. Кроме того, и другие виды диаграмм можно использовать для описания внутренней структуры компонентов или сценариев действий пользователей и прочих элементов, к архитектуре часто не относящихся.

# СТАТИЧЕСКИЕ ДИАГРАММЫ

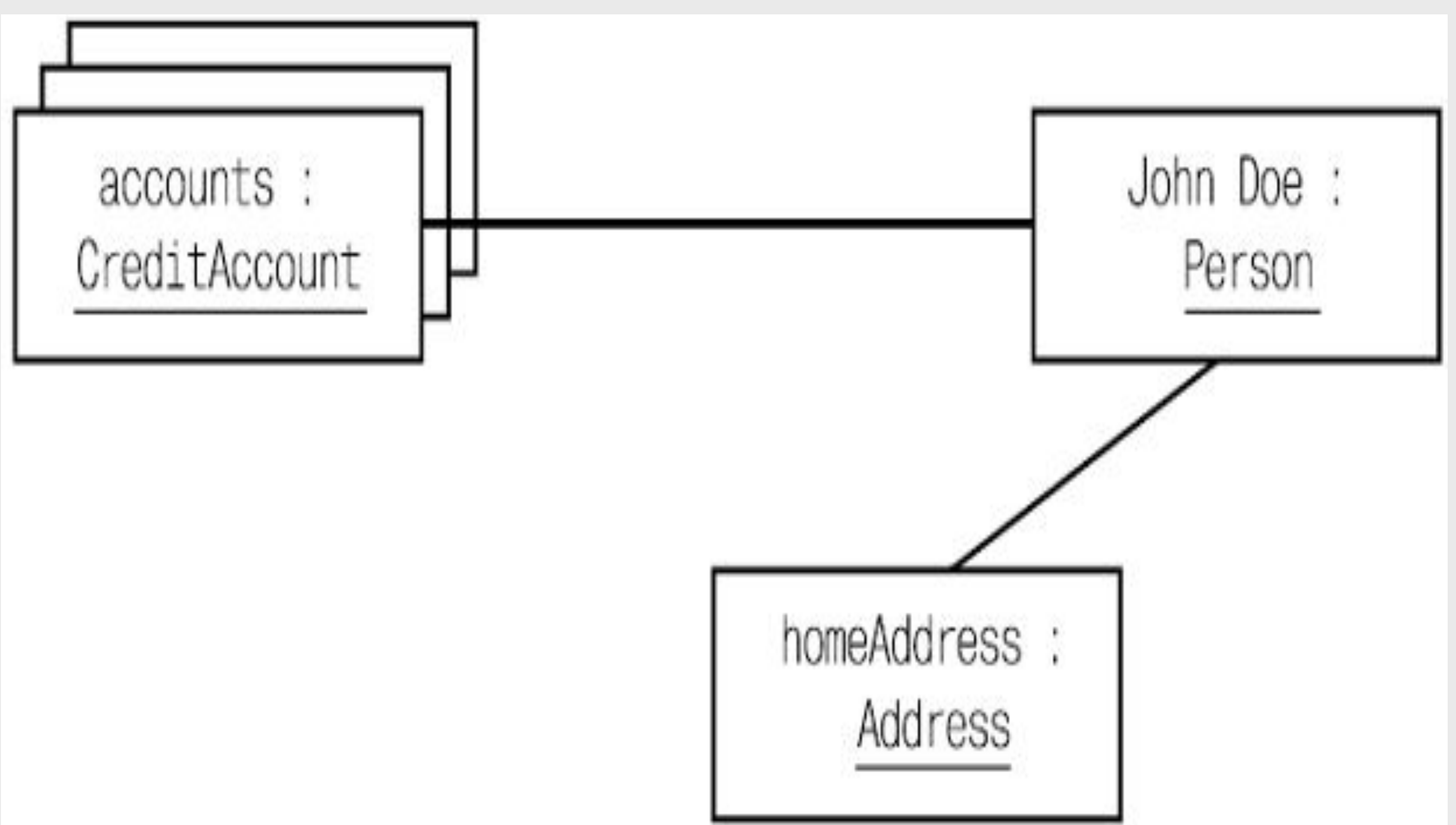
Статические диаграммы представляют либо постоянно присутствующие в системе сущности и связи между ними, либо суммарную информацию о сущностях и связях, либо сущности и связи, существующие в какой-то определенный момент времени. Они не показывают способов поведения этих сущностей. К этому типу относятся *диаграммы классов*, **объектов, компонентов** и *диаграммы развертывания*.

***Диаграммы классов (class diagrams)*** показывают **классы** или **типы** сущностей системы, характеристики классов ( **поля** и **операции** ) и возможные связи между ними.

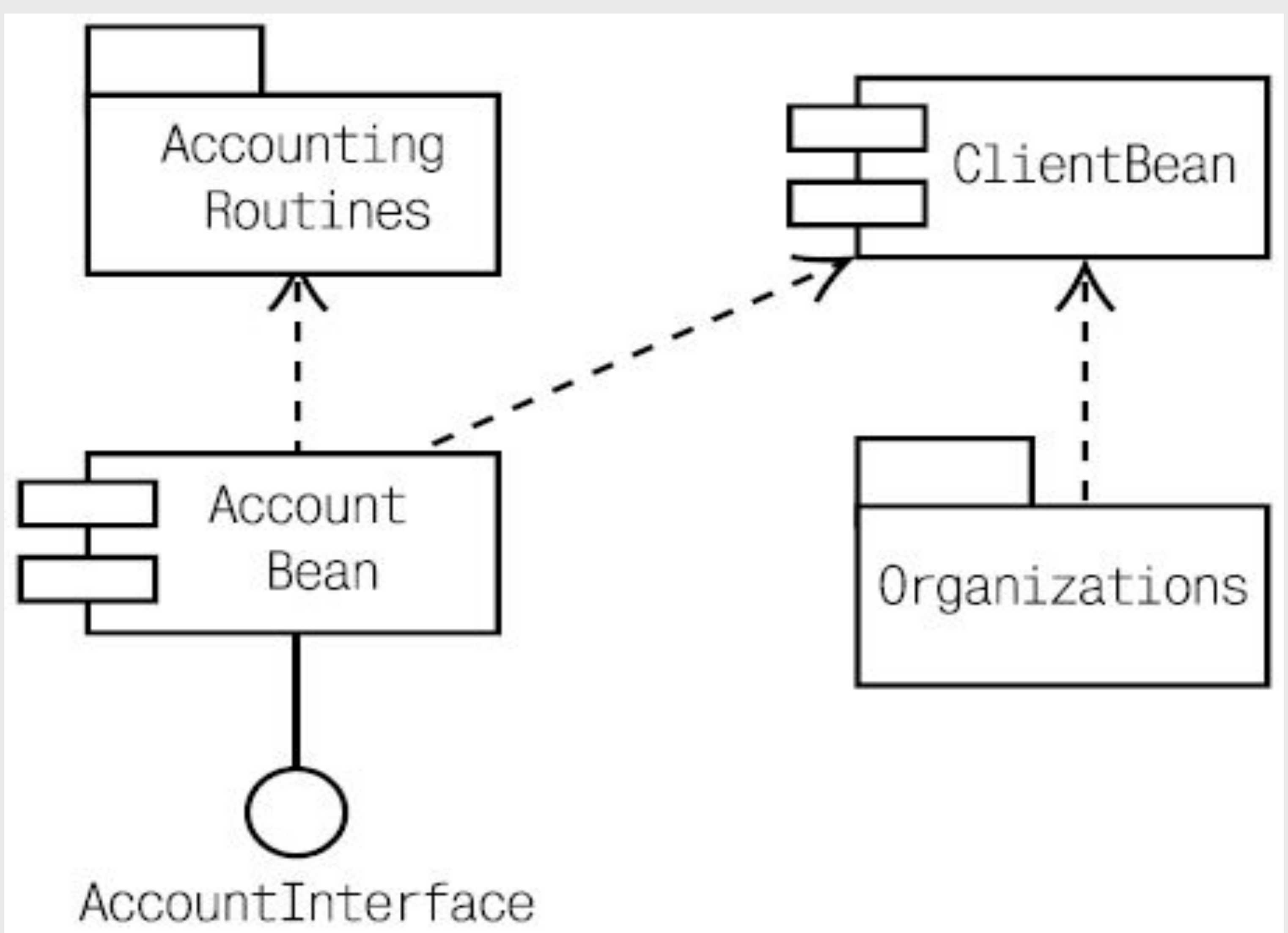
Классы представляются прямоугольниками, поделенными на три части. В верхней части показывают имя класса, в средней — набор его полей, с именами, типами, модификаторами доступа ( public ‘+’, protected ‘#’, private ‘-’ ) и начальными значениями, в нижней — набор операций класса. Для каждой операции показывается ее модификатор доступа и сигнатура.


Account
Interface
Account
-balance : double
+getBalance() : double
+deposit(sum : double) : boolean
+withdraw(sum : double) : boolean
0..n
+accounts
+owner
1
Client
+getId() : int
CreditAccount
-maximumCredit : double
+getCredit() : double
#setCredit(c : double)
Person
-firstName : String
-family : String
+getFulName() : String
Organization
-title : String
+getTitle() : String
kind : AddressKind
+addresses
0..1
0..1
+address
AddressKind
+HOME
+OFFICE
Address
-street : String
-city : String
-state : String
-country : String
-zip : String
+getFulAddress() : String

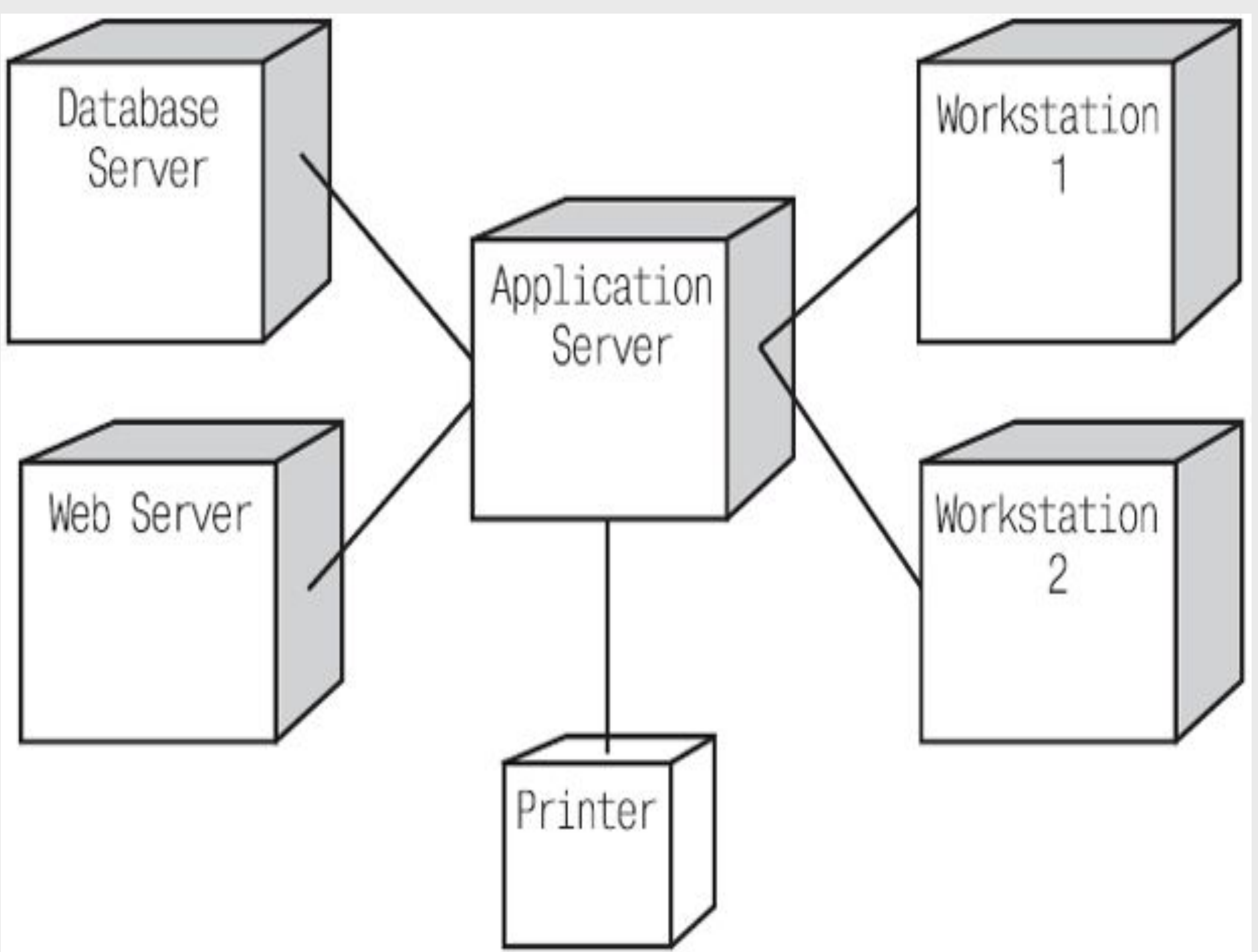
**Диаграммы объектов (object diagrams)** показывают часть объектов системы и связи между ними в некотором конкретном состоянии или суммарно, за некоторый интервал времени. Объекты изображаются прямоугольниками с идентификаторами ролей объектов (в контексте тех состояний, которые изображены на диаграмме) и типами. Однородные коллекции объектов могут изображаться накладывающимися друг на друга прямоугольниками.


accounts :
CreditAccount
John Doe :
Person
homeAddress :
Address

***Диаграммы компонентов (component diagrams)*** представляют компоненты в нескольких смыслах — атомарные составляющие системы с точки зрения ее сборки, *конфигурационного управления* и развертывания. Компоненты сборки и *конфигурационного управления* обычно представляют собой файлы с исходным кодом, динамически подгружаемые библиотеки, HTML-странички и пр., компоненты развертывания — это компоненты *JavaBeans*, CORBA, COM и т.д. Подробнее о таких компонентах см. лекцию 12.Компонент изображается в виде прямоугольника с несколькими прямоугольными или другой формы "зубами" на левой стороне.


Accounting
Routines
ClientBean
Account
Bean
Organizations
AccountInterface

***Диаграммы развертывания (deployment diagrams)*** показывают декомпозицию системы на физические устройства различных видов — серверы, рабочие станции, терминалы, принтеры, маршрутизаторы и пр. — и связи между ними, представленные различного рода сетевыми и индивидуальными соединениями.Физические устройства, называемые **узлами** системы ( **nodes** ), изображаются в виде кубов или параллелепипедов, а физические соединения между ними — в виде линий.

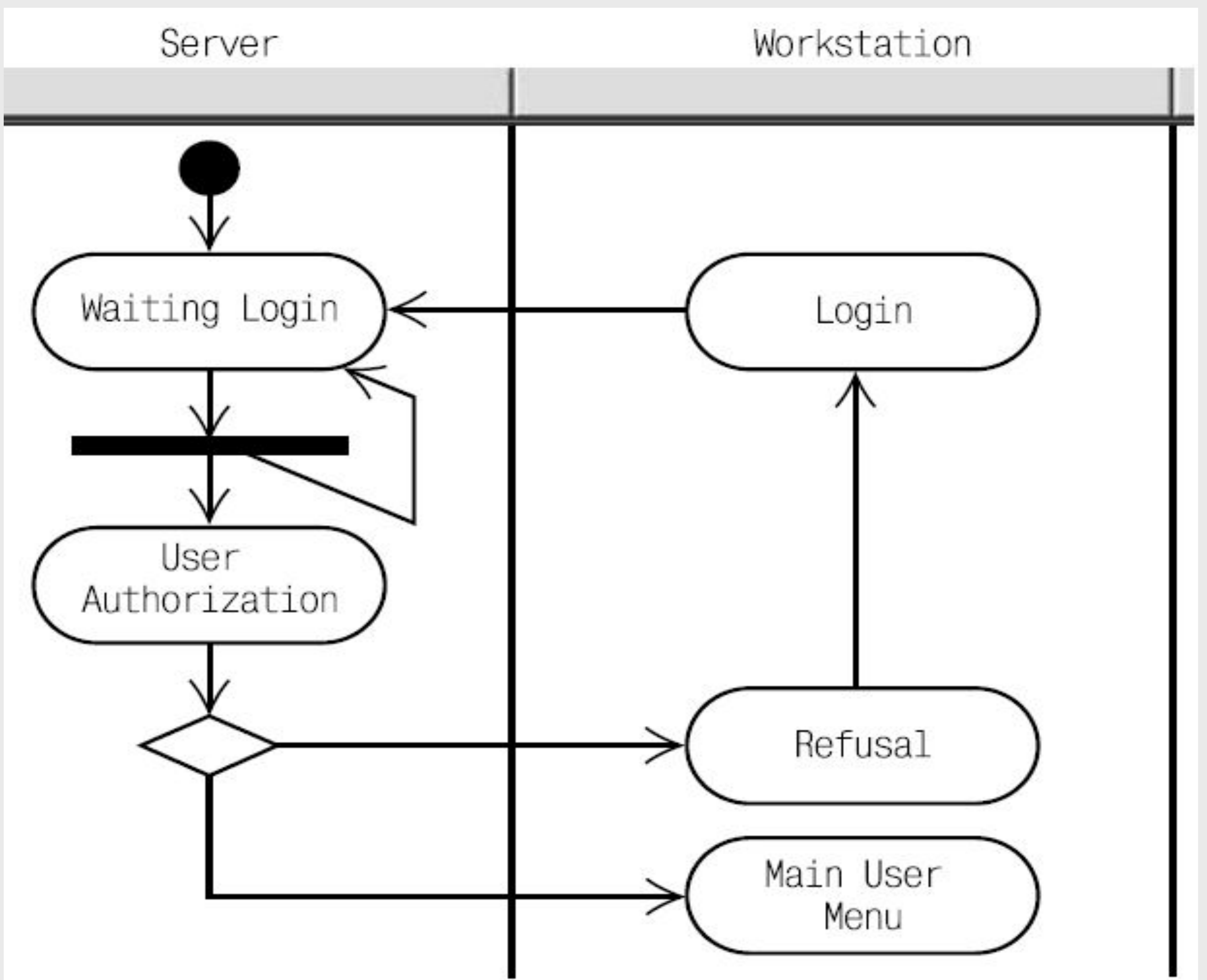

Database Server
Application Server
Workstation 1
Web Server
Workstation 2
Printer

# ДИНАМИЧЕСКИЕ ДИАГРАММЫ

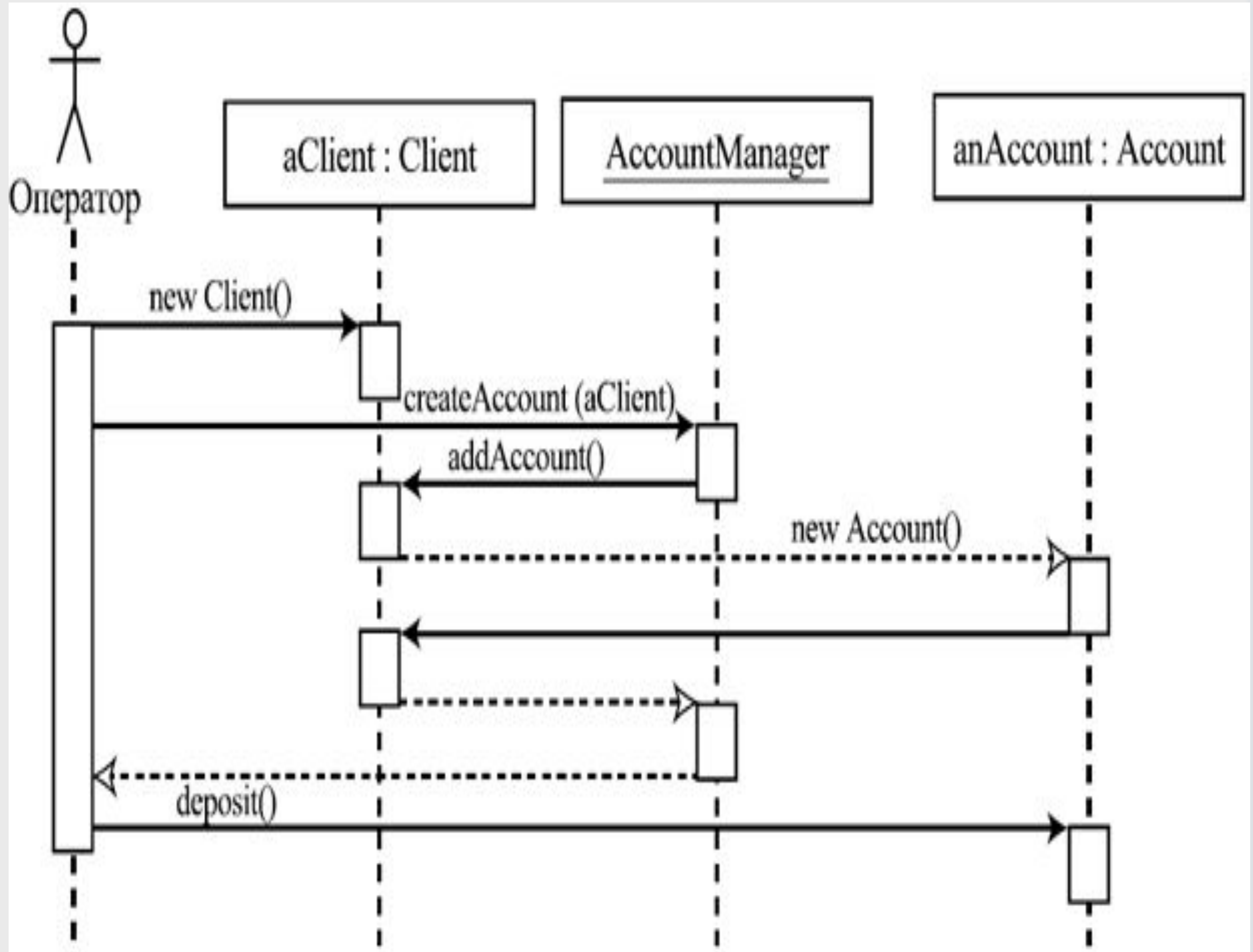
Динамические диаграммы описывают происходящие в системе процессы. К ним относятся **диаграммы деятельности, сценариев**,*диаграммы взаимодействия* и **диаграммы состояний**.

**Диаграммы деятельности (activity diagrams)** иллюстрируют набор процессов-деятельностей и потоки данных между ними, а также возможные их синхронизации друг с другом. Деятельность изображается в виде прямоугольника с закругленными сторонами, слева и справа, помеченного именем деятельности.

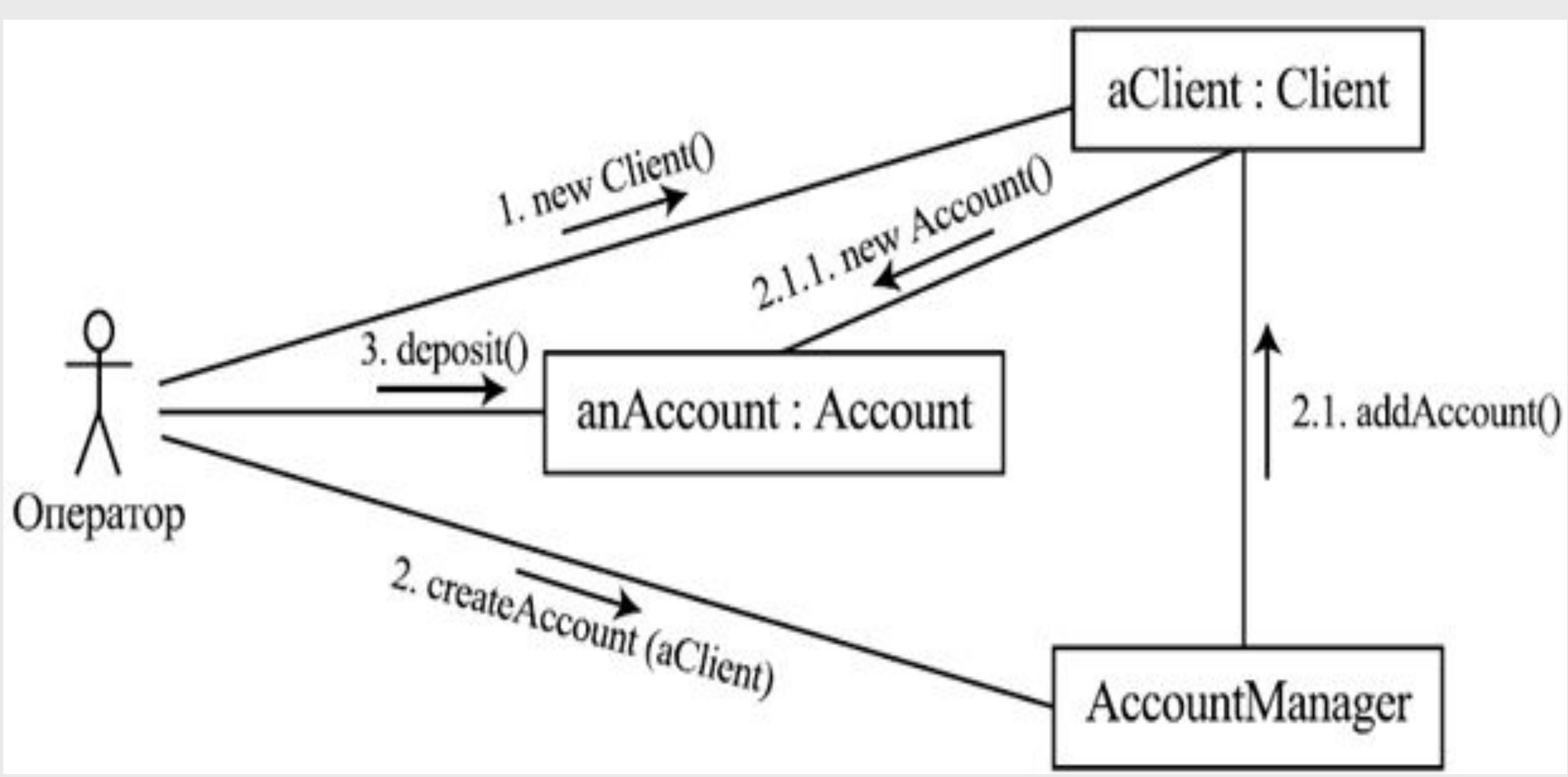
Потоки данных показываются в виде стрелок. Синхронизации двух видов — **развилки (forks)** и **слияния (joins)** — показываются жирными короткими линиями (кто-то может посчитать их и тонкими закрашенными прямоугольниками), к которым сходятся или от которых расходятся потоки данных. Кроме синхронизаций, на диаграммах деятельности могут быть показаны разветвления потоков данных, связанных с выбором того или иного направления в зависимости от некоторого условия. Такие разветвления показываются в виде небольших ромбов.


Server
Workstation
Waiting Login
Login
User Authorization
Refusal
Main User Menu

***Диаграммы сценариев*** **(или диаграммы последовательности, sequence diagrams)** показывают возможные сценарии обмена сообщениями или вызовами во времени между различными компонентами системы (здесь имеются в виду архитектурные компоненты, компоненты в широком смысле — это могут быть компоненты развертывания, обычные объекты, подсистемы и пр.). Эти диаграммы являются подмножеством специального графического языка — языка диаграмм **последовательностей сообщений (Message Sequence Charts, MSC)**, который был придуман раньше UML и достаточно долго развивается параллельно ему.

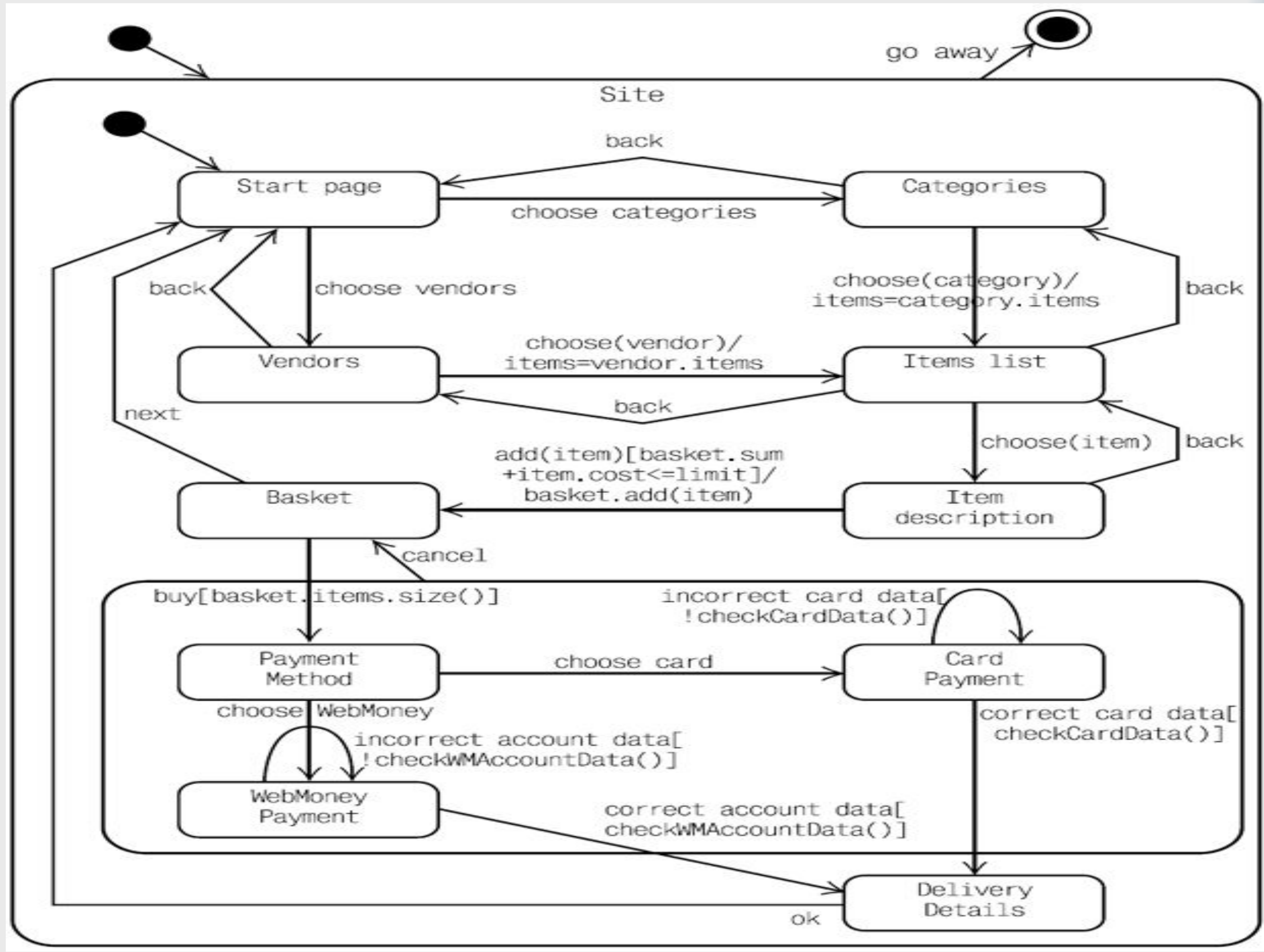

Оператор
aClient : Client
AccountManager
anAccount : Account
new Client()
createAccount (aClient)
addAccount()
new Account()
deposit()

***Диаграммы взаимодействия (collaboration diagrams)*** показывают ту же информацию, что и *диаграммы сценариев*, но привязывают обмен сообщениями/вызовами не к времени, а к связям между компонентами



**Диаграммы состояний (statechart diagrams)** показывают возможные состояния отдельных компонентов или системы в целом, переходы между ними в ответ на какие-либо события и выполняемые при этом действия.

Состояния могут быть устроены иерархически: они могут включать в себя другие состояния, даже целые отдельные диаграммы вложенных состояний и переходов между ними. Пребывая в таком состоянии, система находится ровно в одном из его**подсостояний**.


go away
Site
back
Start page
choose categories
Categories
back
choose vendors
choose(category)/
items=category.items
back
choose(vendor)/
items=vendor.items
Vendors
Items list
next
back
choose(item)
back
add(item)[basket.sum
+item.cost<=limit]/
basket.add(item)
Basket
Item
description
cancel
buy[basket.items.size()]
incorrect card data[
!checkCardData()]
Payment
Method
choose card
Card
Payment
choose WebMoney
incorrect account data[
!checkWMAccountData()]
correct card data[
checkCardData()]
WebMoney
Payment
correct account data[
checkWMAccountData()]
Delivery
Details
ok

всё