Цикл разработки ПО и роль тестера на каждом этапе

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)

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

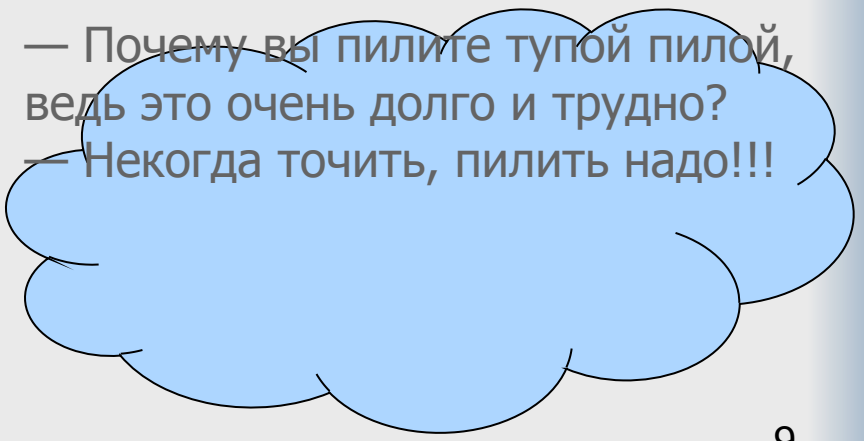
	Водопадная	*Итеративные*	
		Инкрементная	*Эволюционная*
В начале определены все требования?	+	+	-
Циклов конструирования	1	>1	>1
Промежуточное ПО распространяется?	-	**±**	+

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
- …



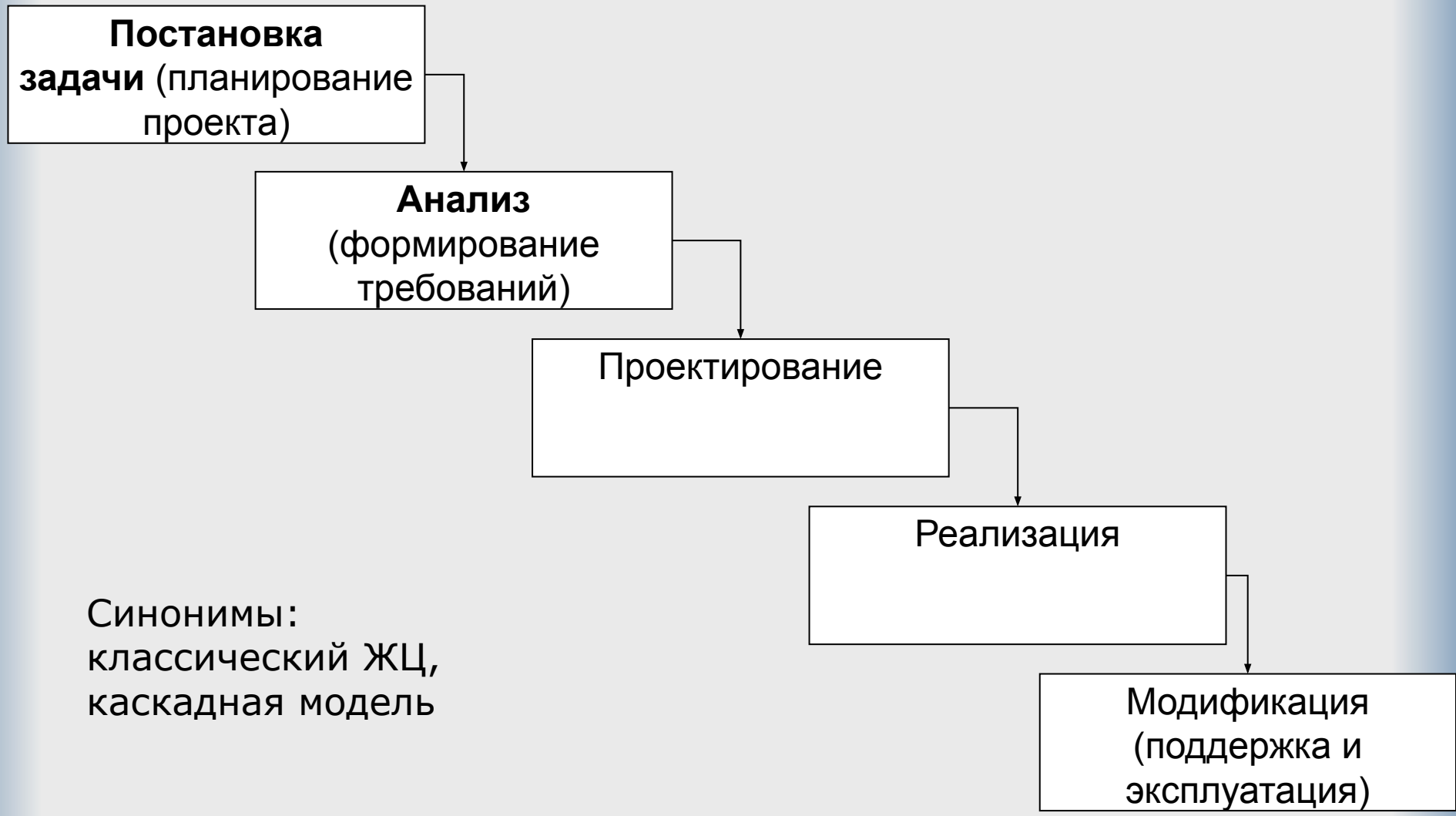
Источники сложности проекта

- 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

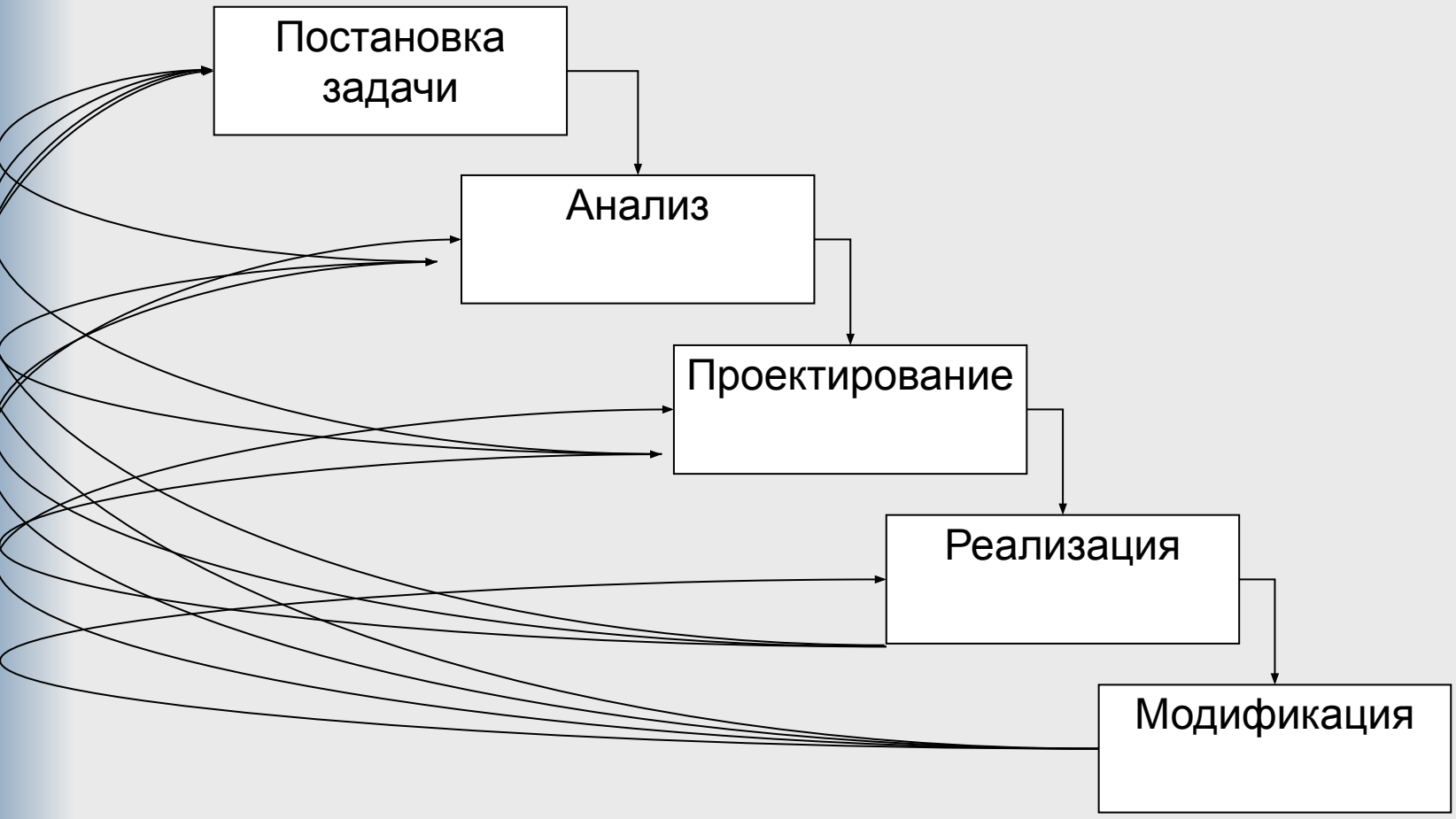
- Многие процессы разработки неуправляемы. Их исходные данные и желаемый результат неизвестны или определены очень нечетко.
- Процесс достижения желаемого результата не поддается формализации (например, разработка архитектуры и исчерпывающее тестирование продукта).
- Идентифицированные процессы разработки 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:



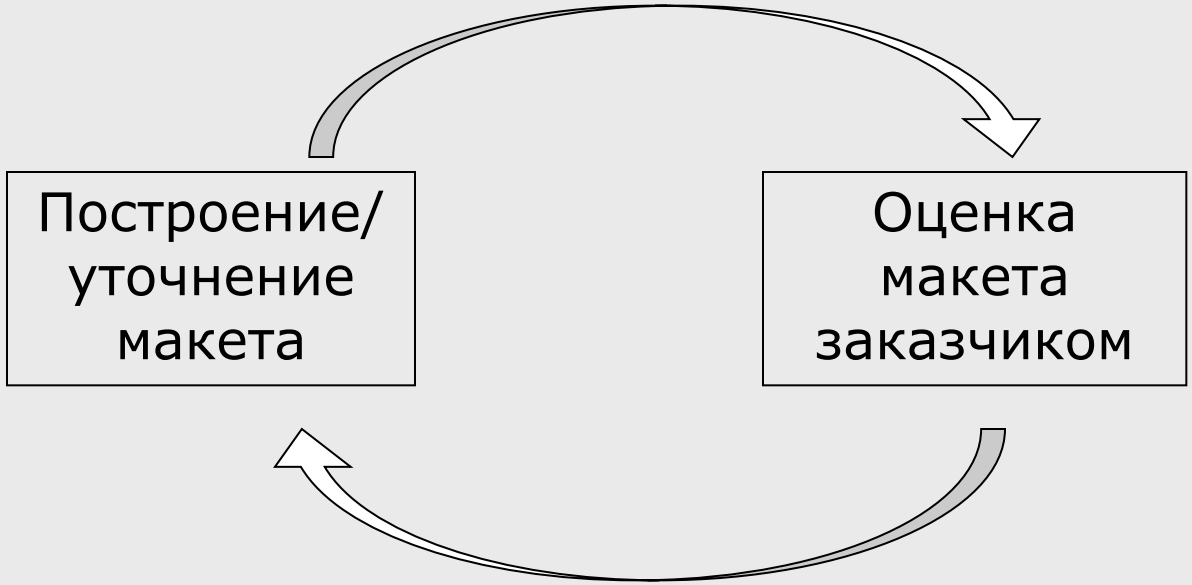
Синонимы:
классический ЖЦ,
каскадная модель

Модель с промежуточным контролем:



Макетирование (прототипирование)

1



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**:

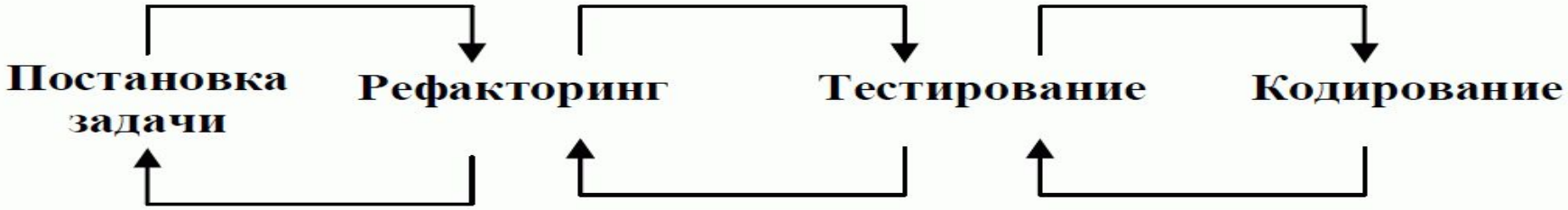
- автоматические тесты пишутся для любой части реализации, которая гипотетически «может сломаться»;
- тесты пишутся непосредственно *перед* написанием соответствующего кода;
- существующий код никогда не меняется без написания соответствующих тестов;
- выполняется регулярный запуск всех автоматических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

- 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

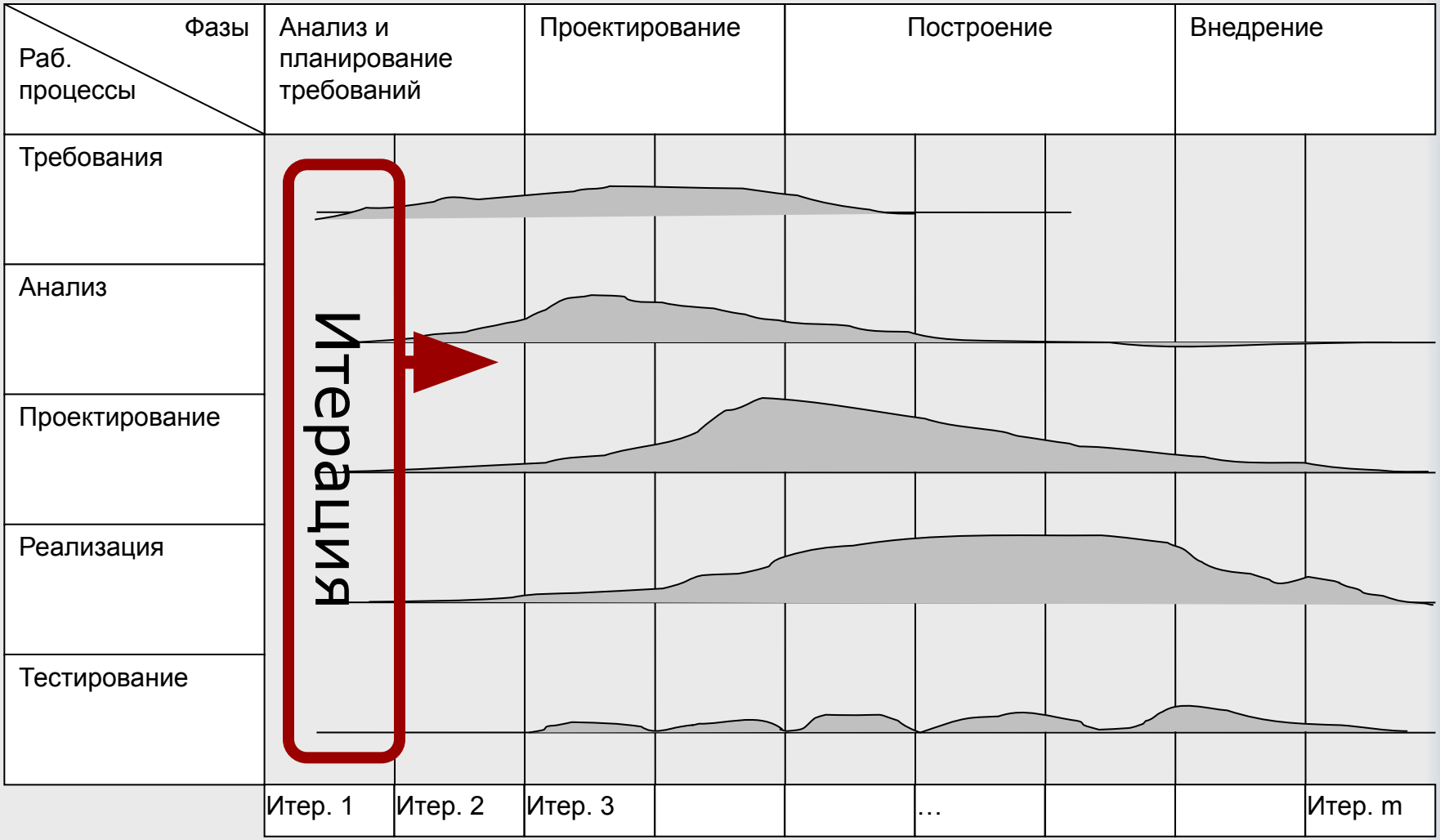
Жизненный цикл УП

- 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

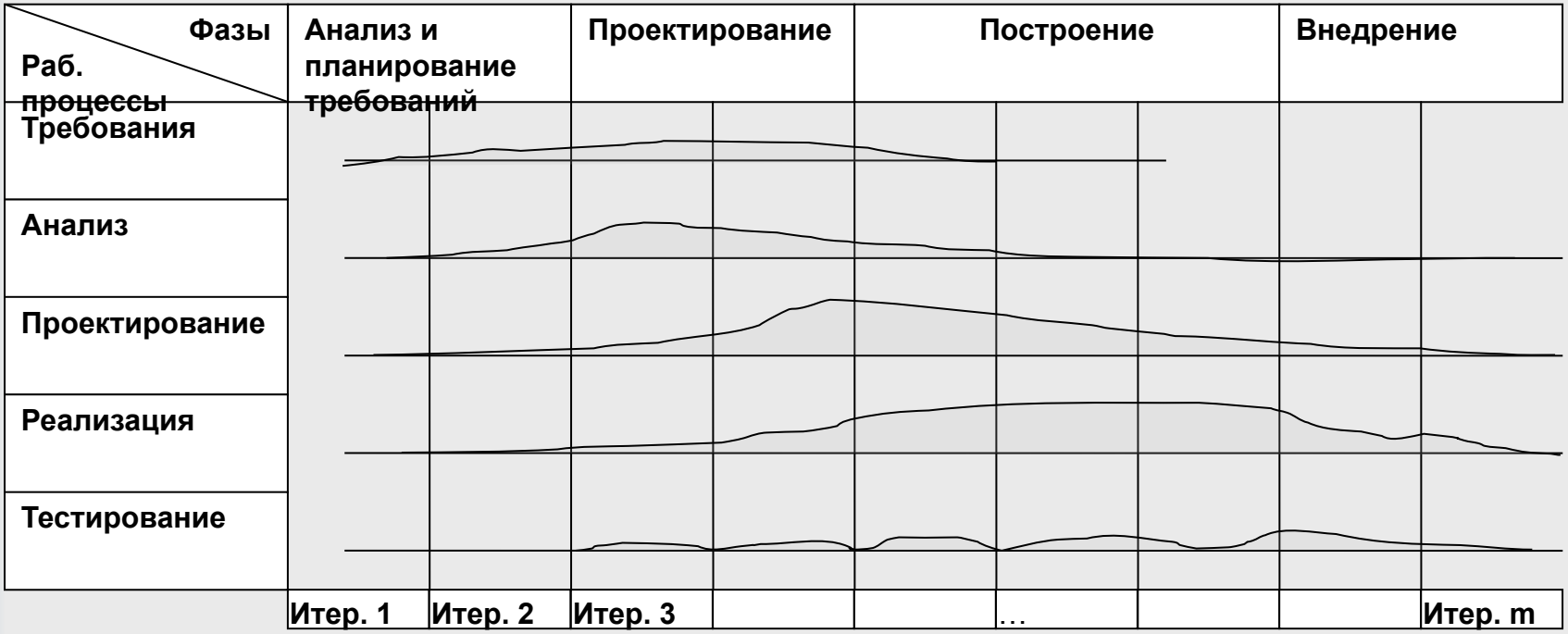
Назначение вех

- 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

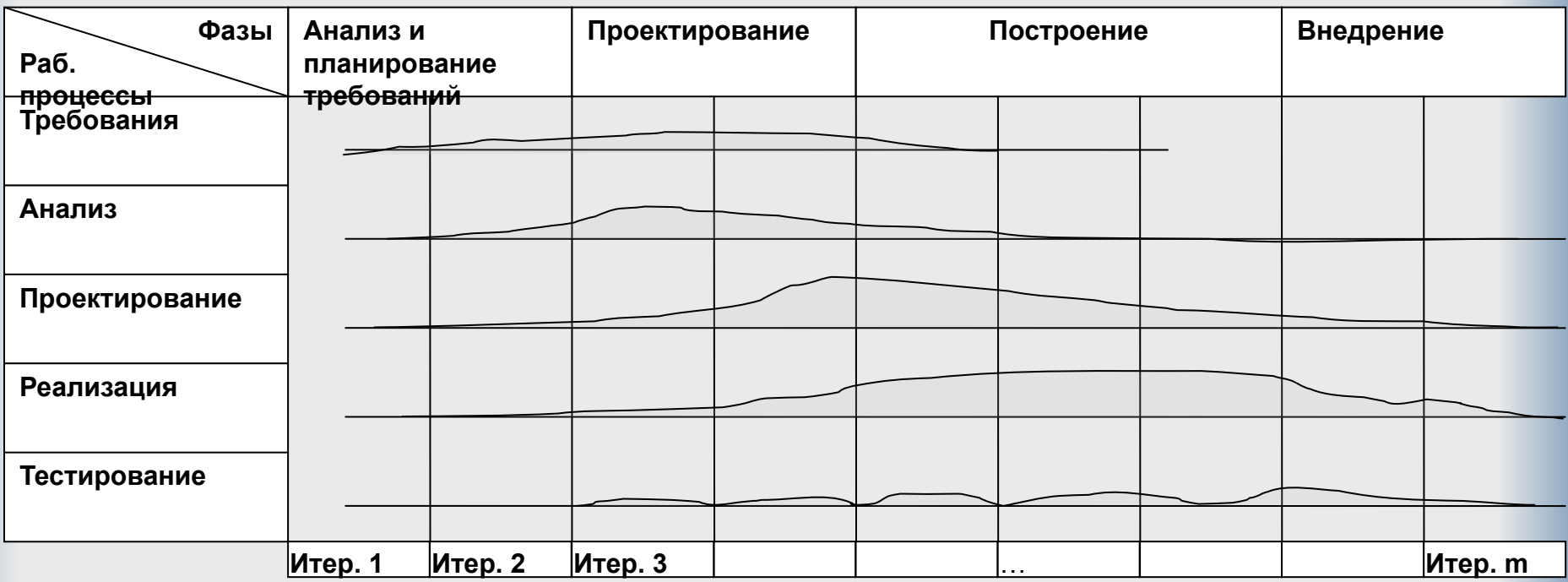


Анализ и планирование требований:

- 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



- 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

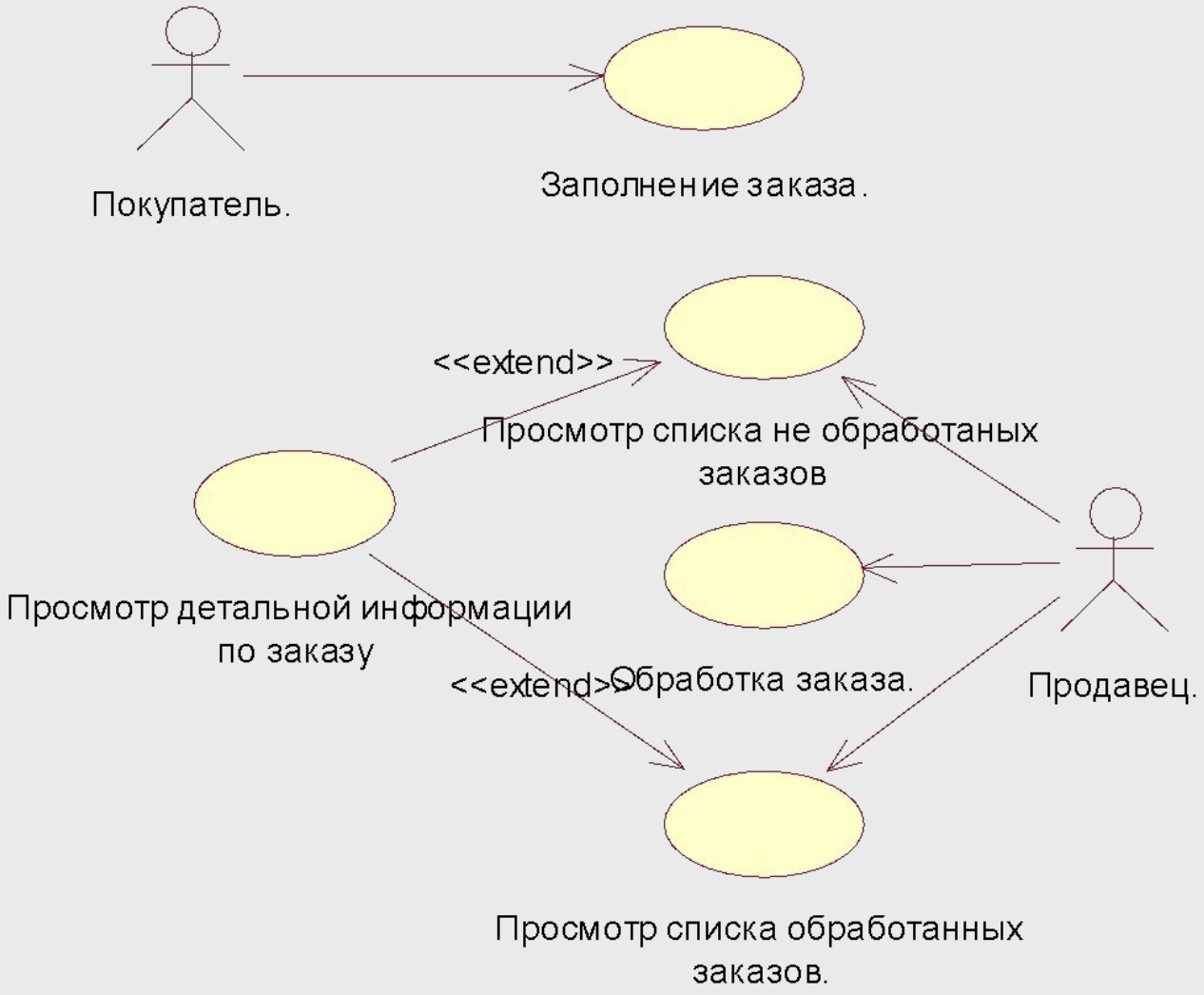
Язык для специфицирования, визуализации, конструирования и документирования программных продуктов.

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

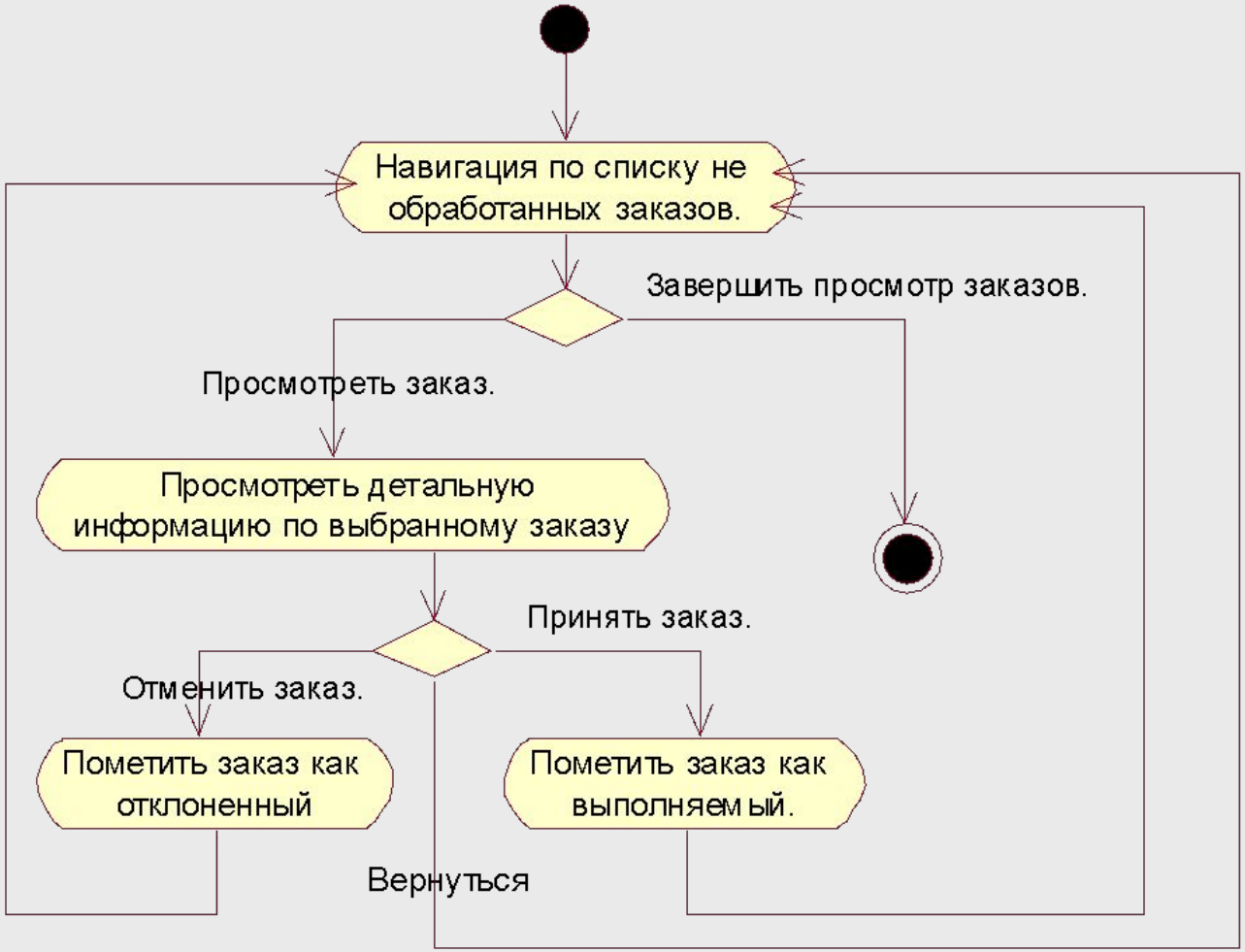
UML позволяет задавать следующие аспекты:

- 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)

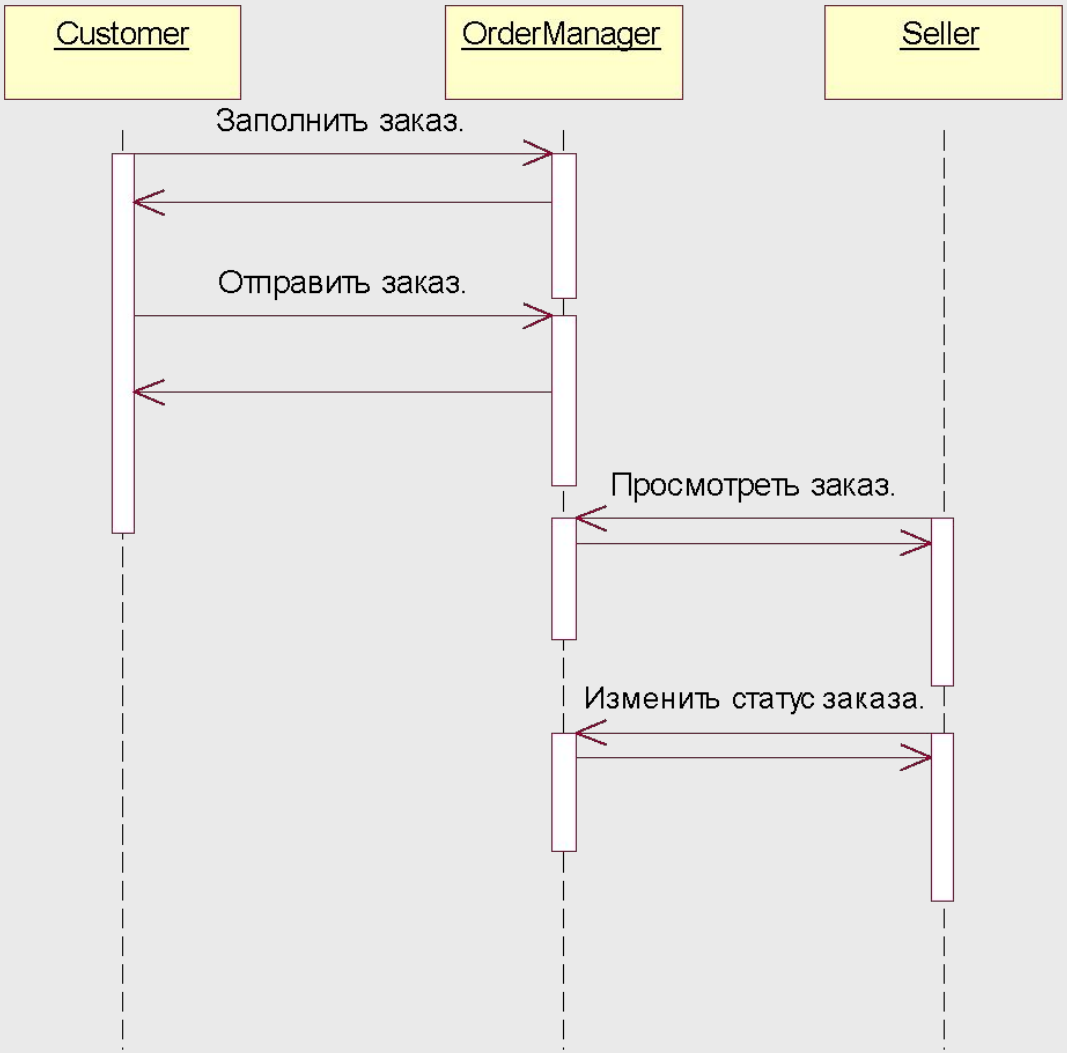
Диаграммы вариантов использования (Use case diagrams)



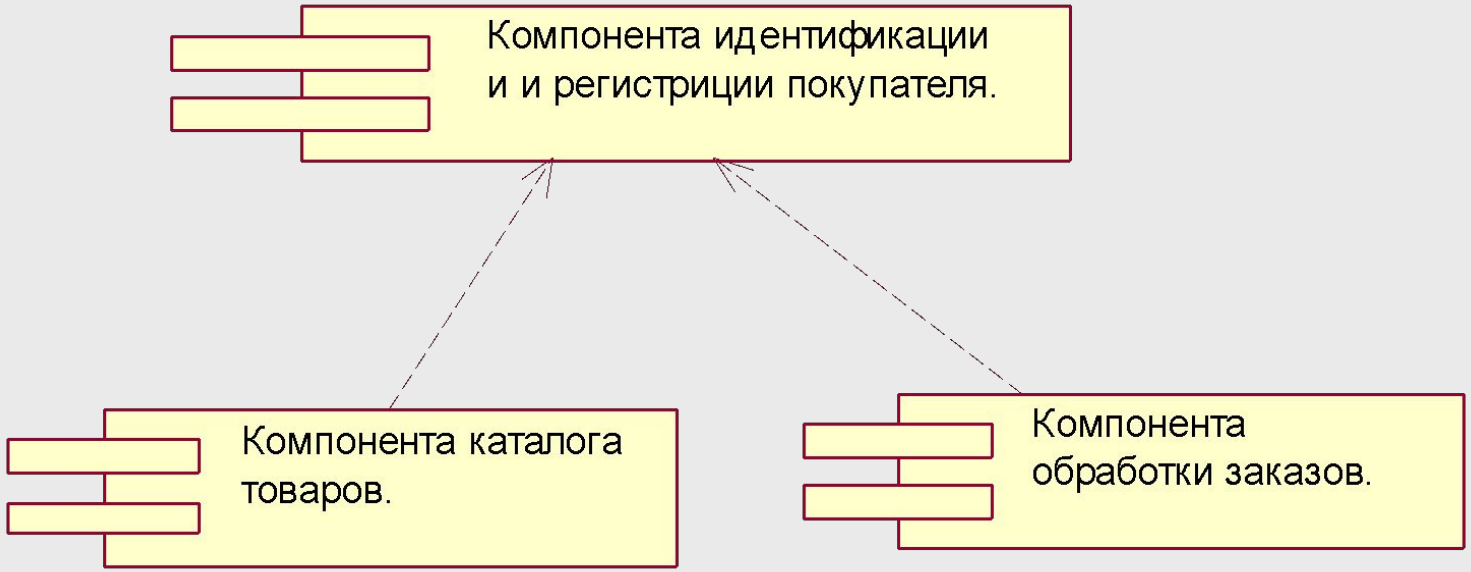
Диаграммы деятельности (Activity diagrams)



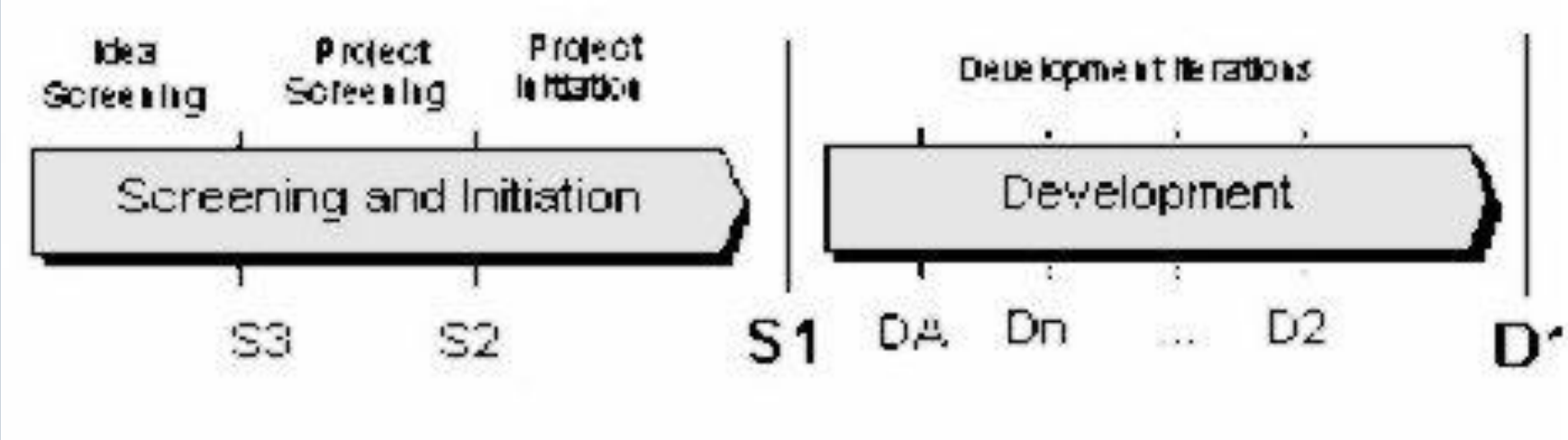
Диаграммы последовательностей действий (Sequence diagrams)

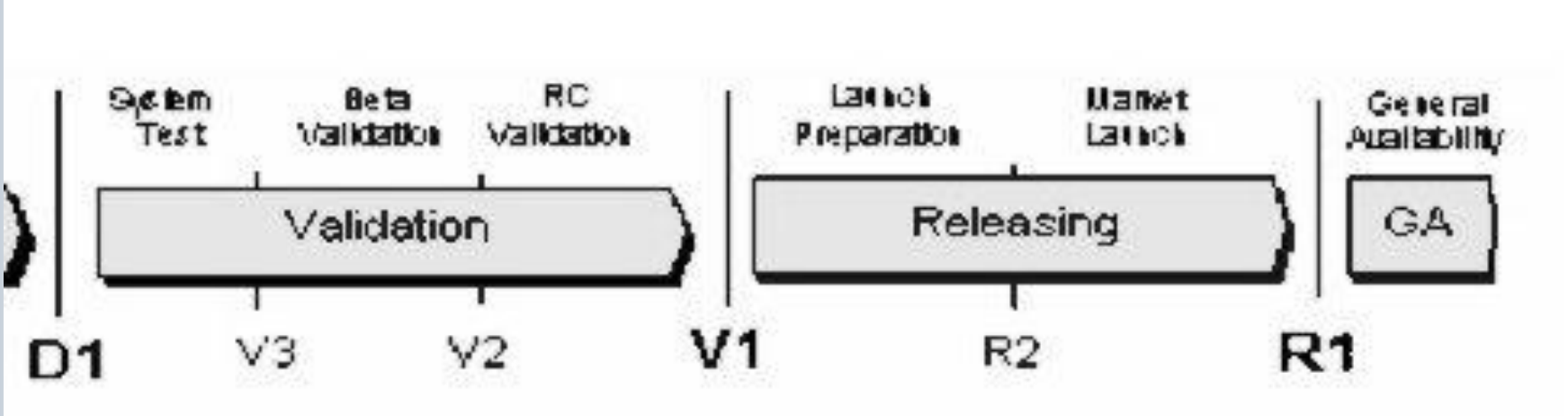


Диаграммы компонент (Component diagrams)

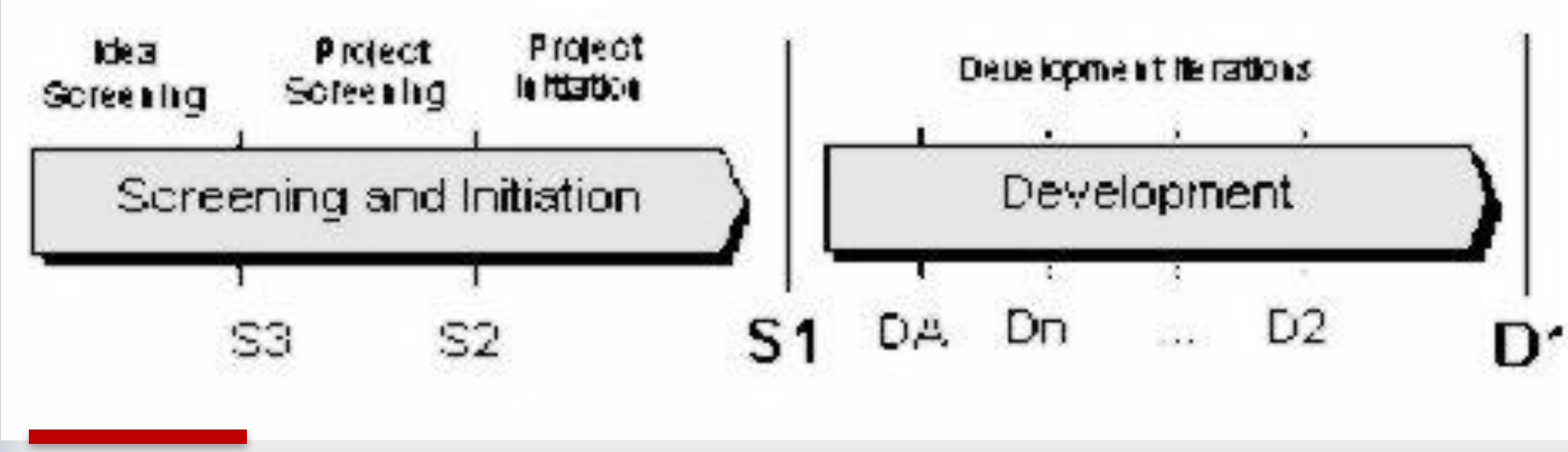


Пример реального процесса разработки ПО



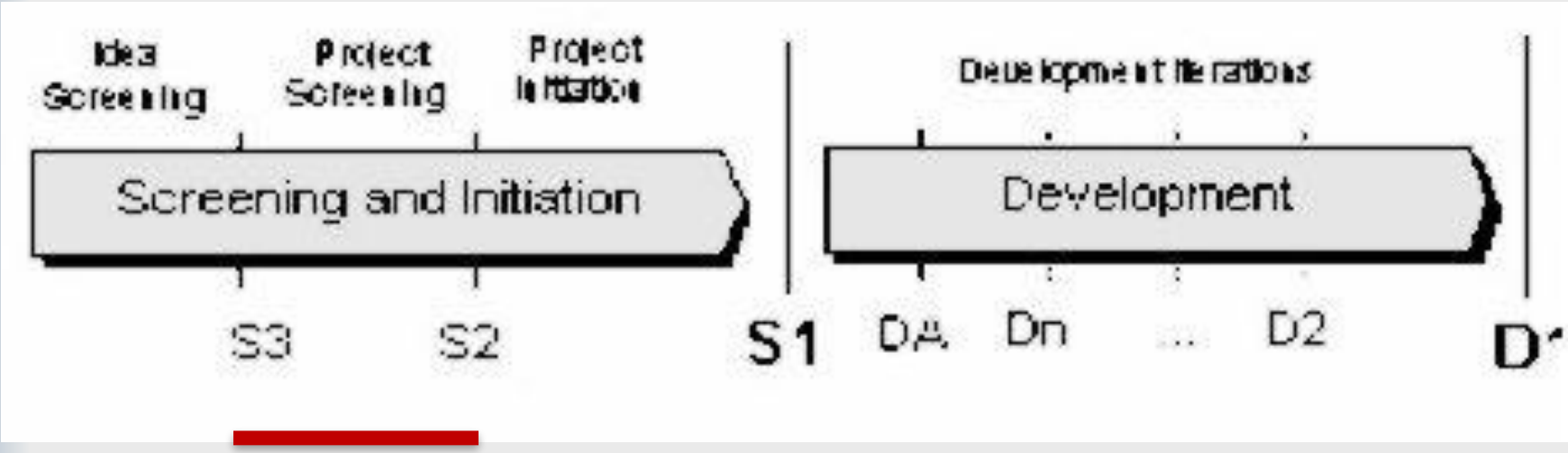


Обзор идеи



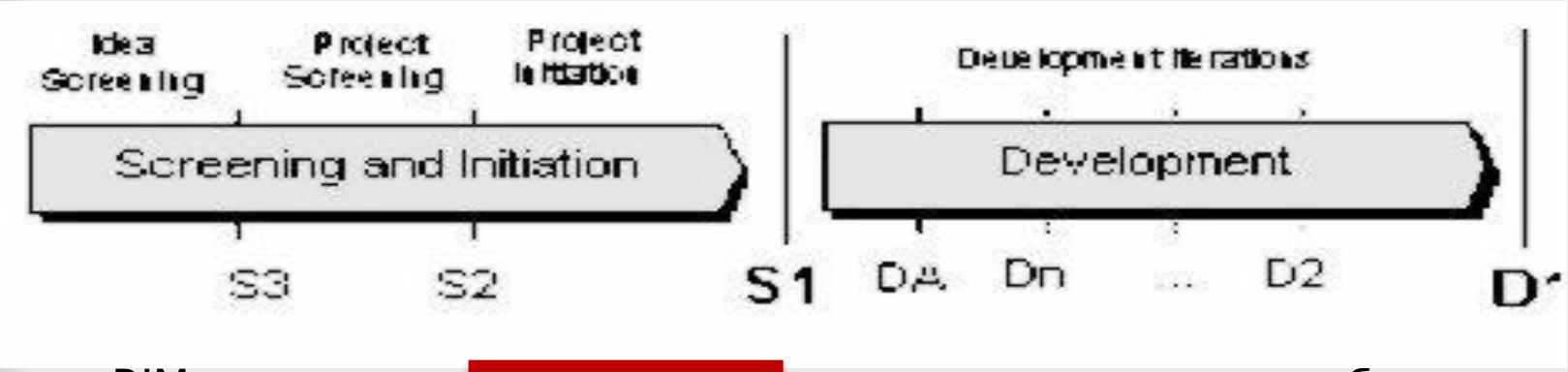
- Выдвигается идея нового продукта
- Назначается менеджер по продукту (PdM). Он оценивает идею и составляет ее краткий обзор, который направляет на утверждение HBU и HPdM.
- Назначается PjM
- Milestone **S3**: HBU или HPdM принимают решение о дальнейшем анализе бизнес-идеи

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.

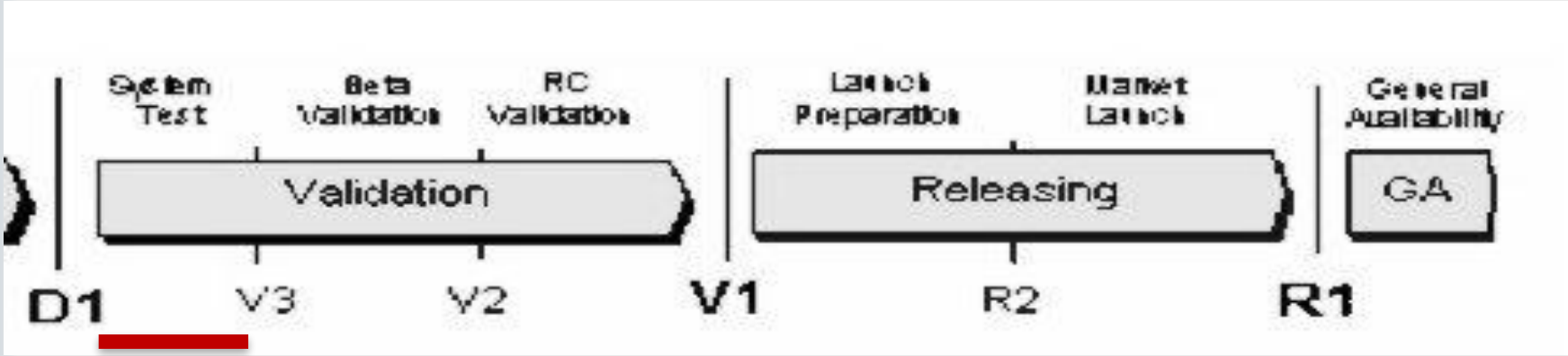
Разработка продукта (Development) - 1

- 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

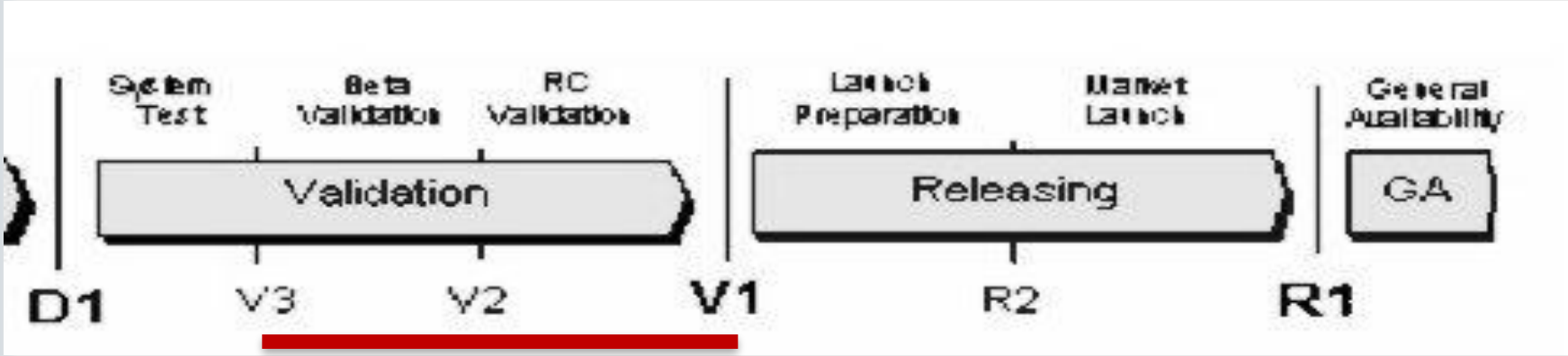
- Выполняется итеративно: анализ, дизайн, программирование, тестирование.
- Milestones Dn – D1: завершение билда N, …, 1.
- Milestone **D1**:
  - Фиксация - Code & feature freeze (alpha version)
  - Нет серьезных дефектов - No any urgent bugs
  - CQA подготовил тестовую спецификацию
  - Первая версия.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

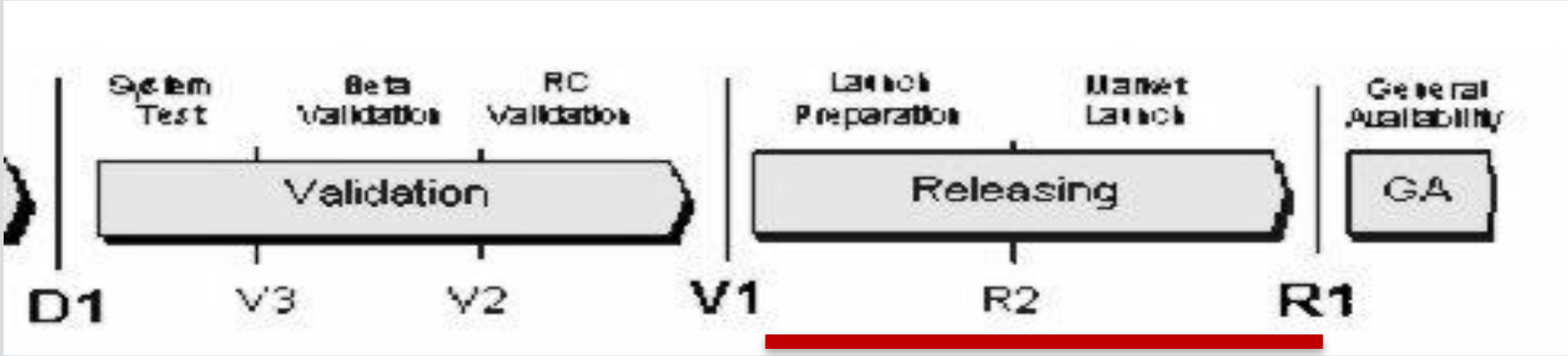
Milestone **V2**: готов Release Candidate, no any unresolved problems found.

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.