Жизненный цикл программного обеспечения. Модели ЖЦ ПО.

Содержание доклада

- Жизненный цикл ПО.
- Стандарт ISO/IEC 12207.
- Классические модели ЖЦ ПО.
- Методология гибкого программирования.

Введение

- В начале развития компьютерной индустрии проблема написания программ относилась к области искусства.
- Важным толчком к совершенствованию методов разработки программ стало разделение пользователей на два класса: тех, кто пишет программы, и тех, кто решает с помощью этих программ прикладные задачи.

Методология

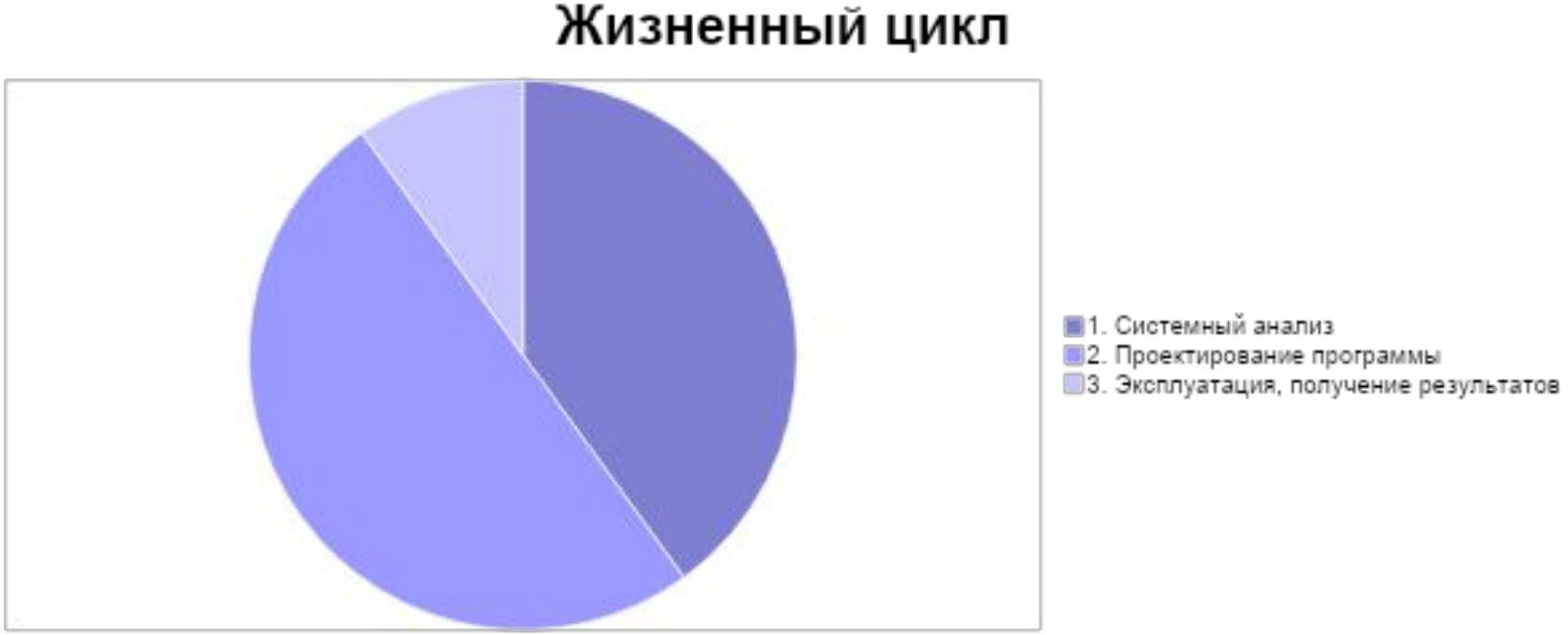
- Основная задача профессионального программирования состоит в создании высококачественного ПО. Чтобы решать эту задачу на должном уровне необходимо определить методологию программирования и сформировать технологический подход

Жизненный цикл

- Жизненный цикл программного обеспечения (ПО) включает в себя все этапы его развития: от возникновения потребности в нем до полного прекращения его использования вследствие морального старения или потери необходимости решения соответствующих задач.
- По длительности жизненного цикла программные изделия можно разделить на два класса: с ***малым*** и ***большим временем жизни***

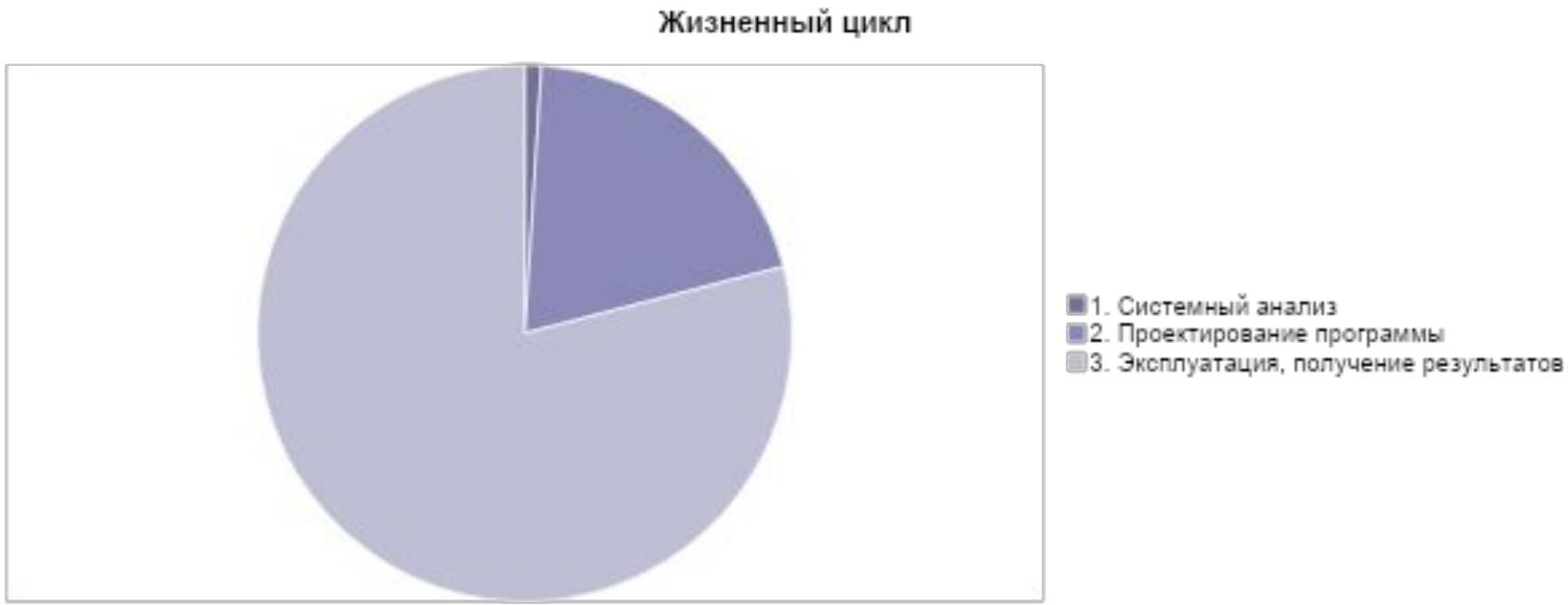
ПО с малым временем жизни

Программные изделия с малой длительностью эксплуатации создаются в основном для решения научных и инженерных задач, для получения конкретных результатов вычислений. Такие программы обычно относительно невелики.



Программные изделия с большой длительностью эксплуатации

Программные изделия с большой длительностью эксплуатации создаются для регулярной обработки информации и управления. Структура таких программ сложная. Их размеры могут изменяться в широких пределах (1...1000 тыс. команд), однако все они обладают свойствами познаваемости и возможности модификации в процессе длительного сопровождения и использования различными специалистами.



Основные причины изучения моделирования ЖЦ

- Это знание помогает понять, на что можно рассчитывать при заказе или приобретении ПО и что нереально требовать от него.
- Модели ЖЦ — основа знания технологий программирования и инструментария, поддерживающего их.
- Общие знания того, как развивается программный проект, дают наиболее надежные ориентиры для его планирования.

Обобщенная модель жизненного цикла

Системный анализ

Проектирование программного обеспечения:

Оценка (испытания) программного обеспечения.

Использование программного обеспечения

Стандарт PSS 05_0 организации ESA

- В данном документе описаны все 6 фаз ЖЦ ПО. Также в этом документе описаны некоторые основные модели ЖЦ ПО.
- Документ состоит из семи частей. Первая часть – вводная. Остальные 6 частей точно описывают 6 фаз ЖЦ ПО.

Фазы жизненного цикла ПО по PSS 05_0

- Vision фаза: видение проекта;
- UR-фаза: определение требований заказчика/пользователя (UR-phase);
- SR-фаза: определение требований к ПО (SR-phase);
- AD-фаза: архитектурное проектирование (AD-phase);
- DD-фаза: детальное проектирование и производство код-программы (DD-phase);
- TR-фаза: передача ПО в эксплуатацию (TR-phase);
- OM-фаза: эксплуатация и сопровождение (OM-phase).

ISO/IEC 12207

- Основной нормативный документ, регламентирующий ЖЦ ПО – международный стандарт ***ISO/IEC 12207*** (ISO, International Organization of Standardization – Международная организация по стандартизации, IEC, International Electrotechnical Commission – Международная комиссия по электротехнике). Он определяет структуру ЖЦ, содержащую процессы, действия и задачи, выполняемые во время создания ПО.

Стандарт ISO/IEC 12207:1995

основные процессы жизненного цикла

- процесс приобретения
- процесс поставки
- процесс разработки
- процесс функционирования
- процесс сопровождения

вспомогательные процессы жизненного цикла

- процесс решения проблем;
- процесс документирования;
- процесс управления конфигурацией;
- процесс обеспечения качества;
- процесс верификации;
- процесс аттестации;
- процесс совместной оценки;
- процесс аудита.

Основные процессы

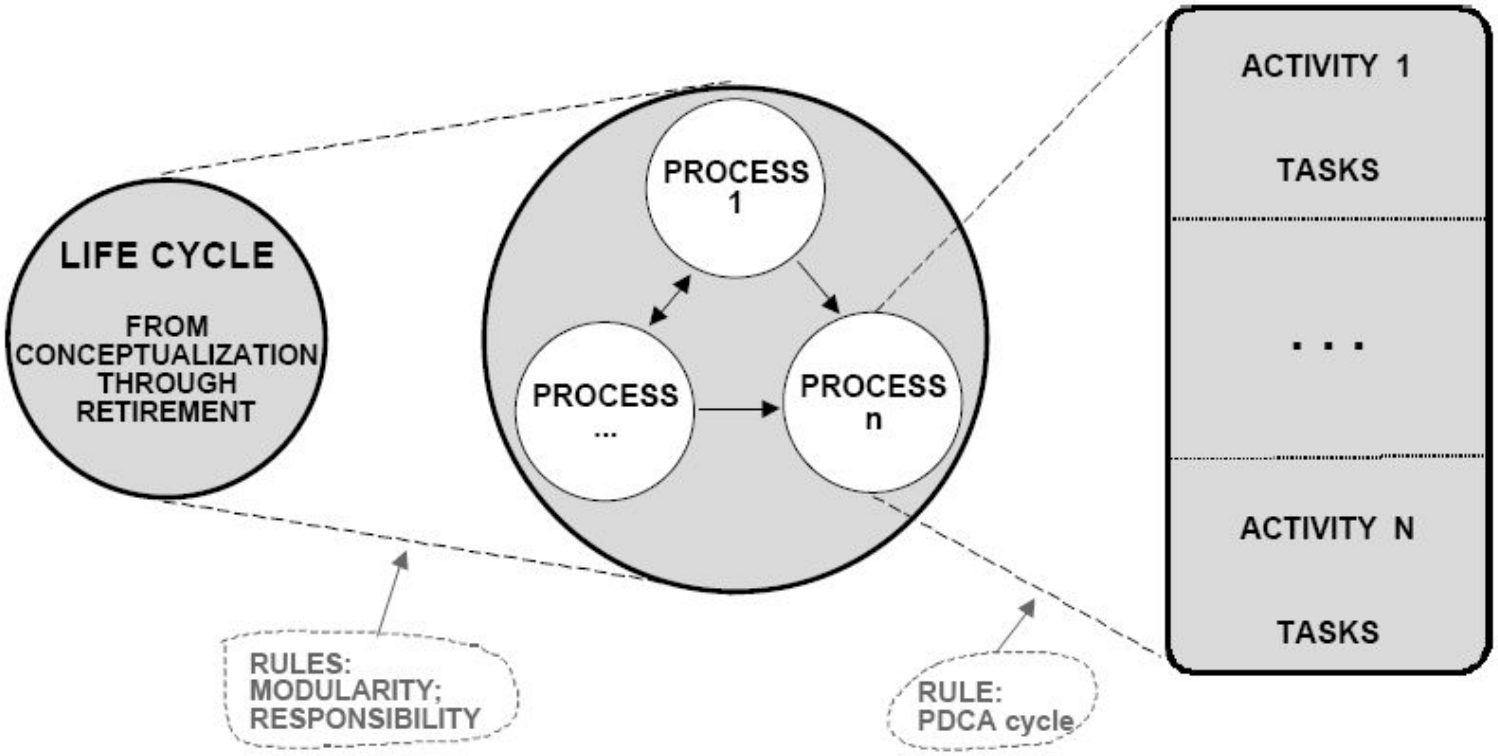
- **Процесс приобретение или заказ** состоит из работ и задач, выполняемых заказчиком. Процесс начинается с определения потребностей заказчика. Далее следуют подготовка и выпуск заявки на подряд, выбор поставщика и управление процессом заказа.
- **Процесс поставки** состоит из работ и задач, выполняемых поставщиком. Процесс продолжается определением процедур и ресурсов, необходимых для управления и обеспечения проекта.
- **Процесс разработки** состоит из работ и задач, выполняемых разработчиком. Процесс включает работы по анализу требований, проектированию, программированию, сборке, тестированию, вводу в действие.

Основные процессы

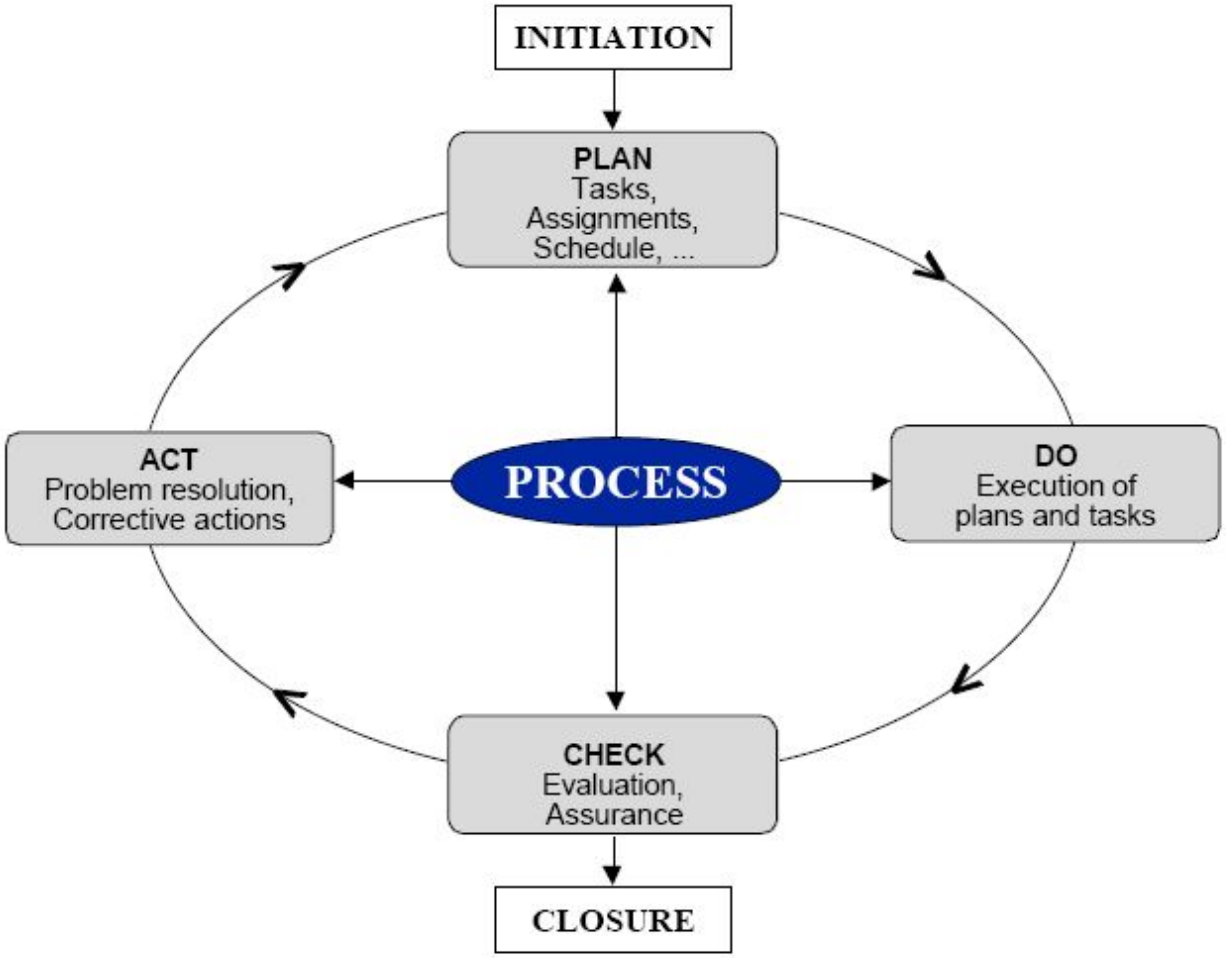
- **Процесс эксплуатации** состоит из работ и задач оператора. Процесс охватывает эксплуатацию программного продукта и поддержку пользователей в процессе эксплуатации.
- **Процесс сопровождения** состоит из работ и задач, выполняемых персоналом сопровождения. Целью процесса является изменение существующего программного продукта при сохранении его целостности.

Процессы -> Задачи -> Работы

- **THE ARCHITECTURING OF THE LIFE CYCLE:**



Процесс


INITIATION
PLAN
Tasks,
Assignments,
Schedule, ...
ACT
Problem resolution,
Corrective actions
PROCESS
DO
Execution of
plans and tasks
CHECK
Evaluation,
Assurance
CLOSURE

ДСТУ 3918

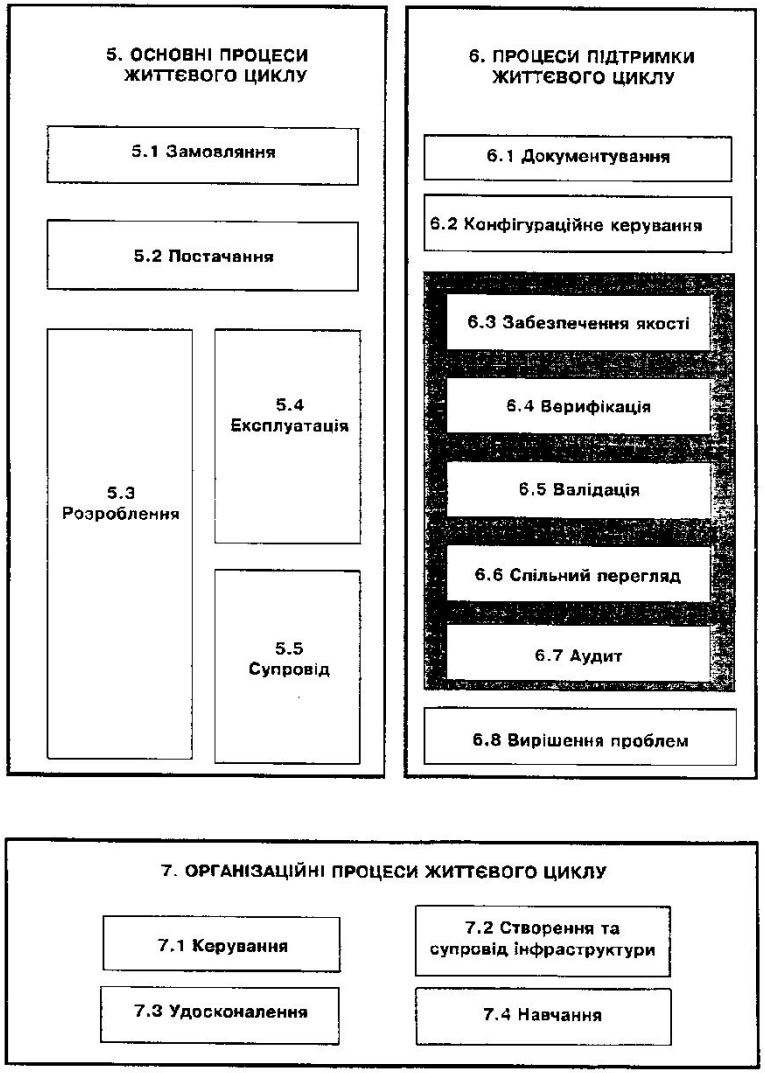
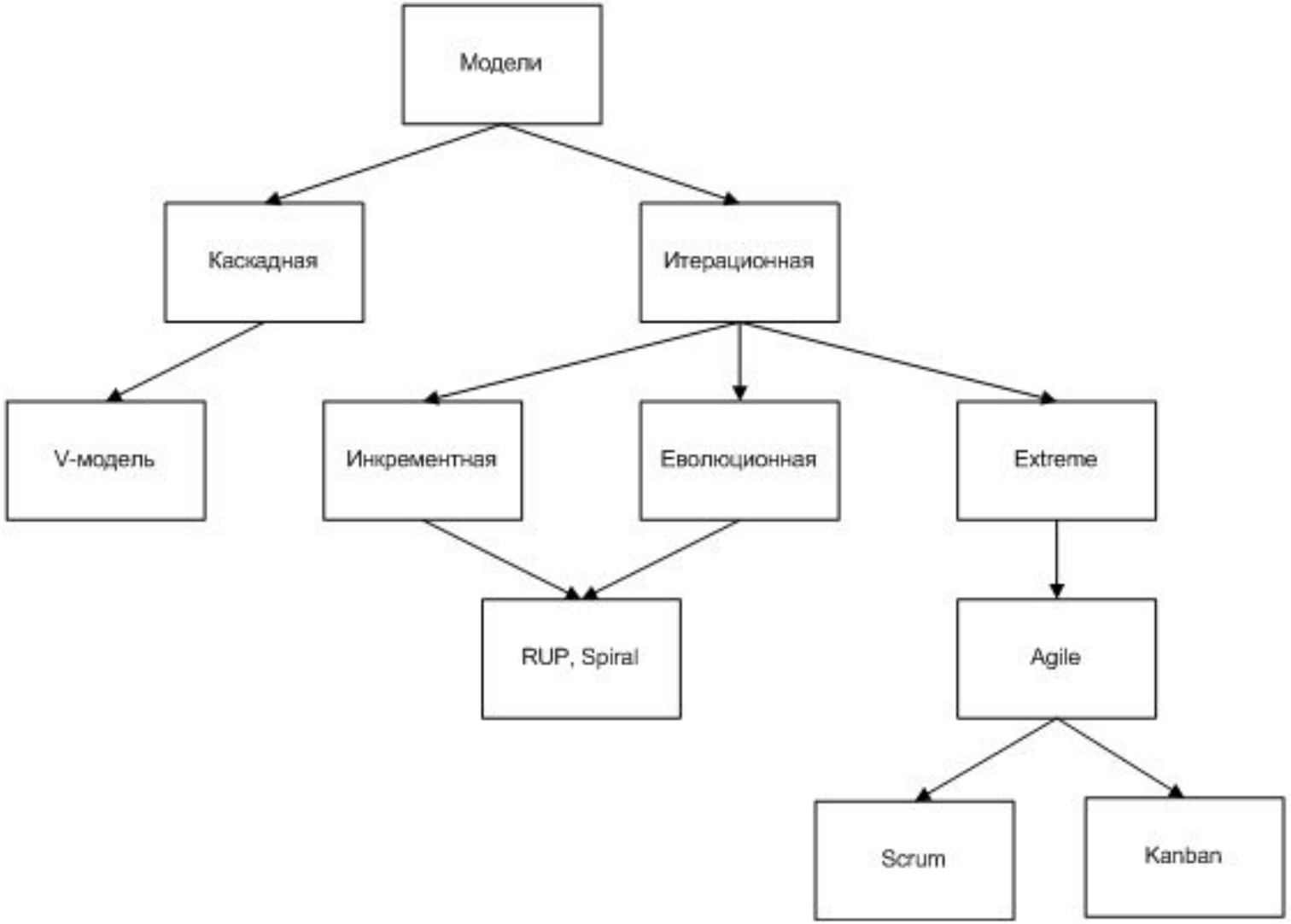


Рисунок 1 — Структура стандарту

Фрагмент ДСТУ 3918

- **5.3 Процес розроблення**
- **5.3.1 Реалізація процесу**
- Ця дія полягає у виконанні таких завдань.
- 5.3.1.1 Якщо модель життєвого циклу програмного забезпечення не обумовлена контрактом, розробник повинен визначити або вибрати її з урахуванням сфери застосування, розмірів та складності проекту. Процеси, дії та завдання цього стандарту слід вибрати та відобразити в моделі життєвого циклу.
- 5.3.1.2 Розробник повинен: …
- **…**
- **5.3.2 Аналіз системних вимог**
- Ця дія полягає у виконанні таких завдань, які розробник повинен виконувати чи підтримувати згідно з вимогами контракту.
- 5.3.2.1 Для визначення системних вимог слід провести аналіз специфіки використання системи, що підлягає розробці.
- …

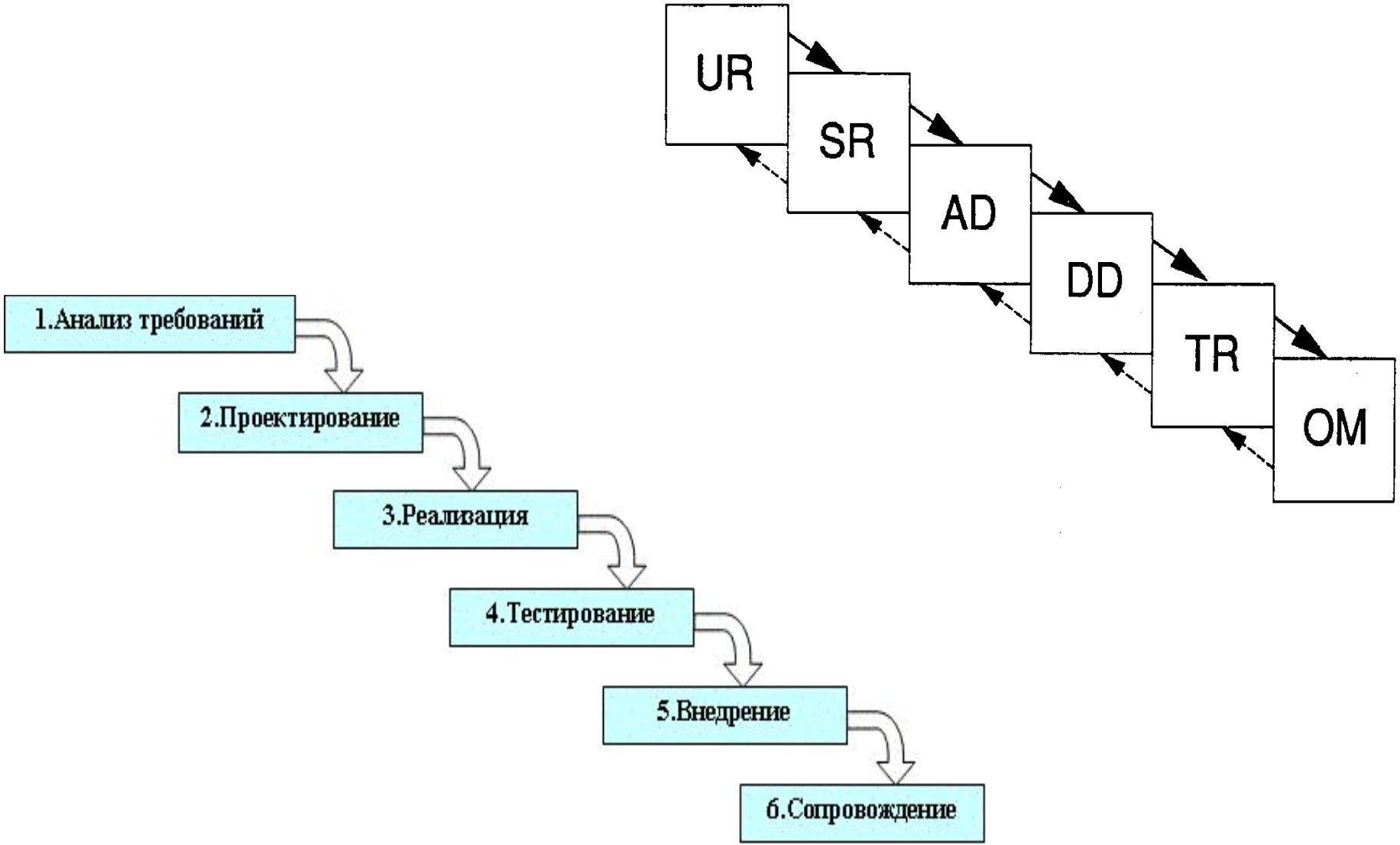
Модели жизненного цикла ПО



Определения

- **Модель жизненного цикла** разбивает процессы проектирования на **фазы** и определяет, какие процессы в какой фазе происходят.
- **Вариант модели жизненного цикла** – это комбинация основных фаз модели жизненного цикла.
- Все проекты ПО **должны** иметь вариант жизненного цикла, включающий в себя основные фазы.

Каскадная модель



Описание каскадной модели

- Это простая модель жизненного цикла ПО, состоит из 6 фаз, выполняющихся последовательно.
- Преимущество: проста и удобна в применении, процесс разработки выполняется поэтапно.
- Недостатки: Ошибка на некотором этапе приводит к возврату назад, что увеличивает затраты. Весь программный продукт разрабатывается за один раз. Нет возможности разбить систему на части.

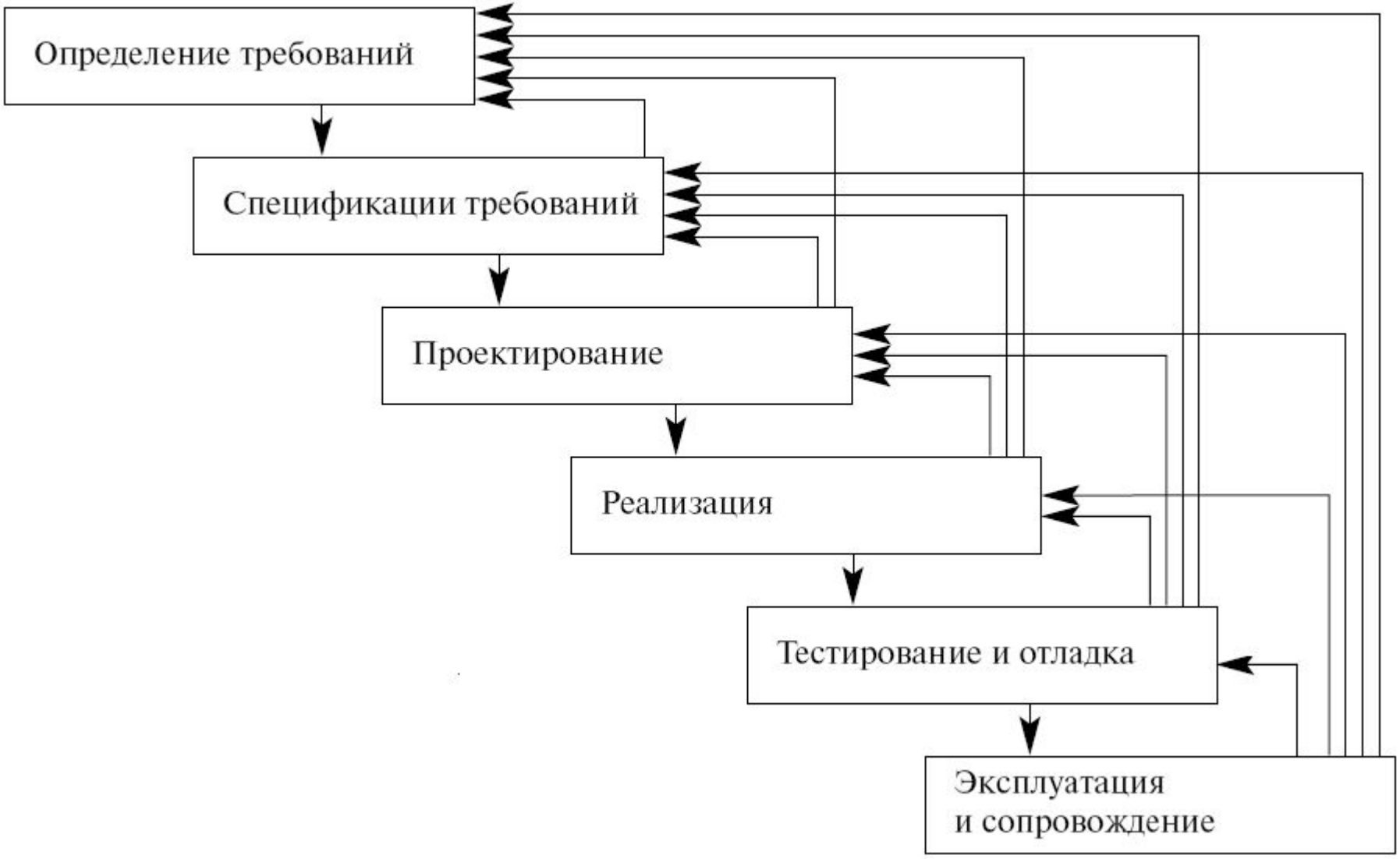
V-модель


Планирование проектов и требований
Производство, эксплуатация и сопровождение
Анализ требований продукта и спецификаций
Системное и приемочное тестирование
Разработка архитектурного проекта на высшем уровне
Интеграция и тестирование
Детализированная разработка проекта
Модульное тестирование
Кодирование

Описание V-модели

- V-модель унаследовала структурную последовательность от каскадной модели. В этой модели особое значение придается действиям, направленным на верификацию и валидацию продукта.
- Преимущества:
  - Основное преимущество это возможность валидации и верификации;
  - Модель проста в использовании.
- Недостатки:
  - В модели не предусмотрено внесение требования динамических изменений на разных этапах жизненного цикла, что считается главным недостатком данной модели;
  - Невозможно внести изменения, не повлияв при этом на график выполнения проекта;

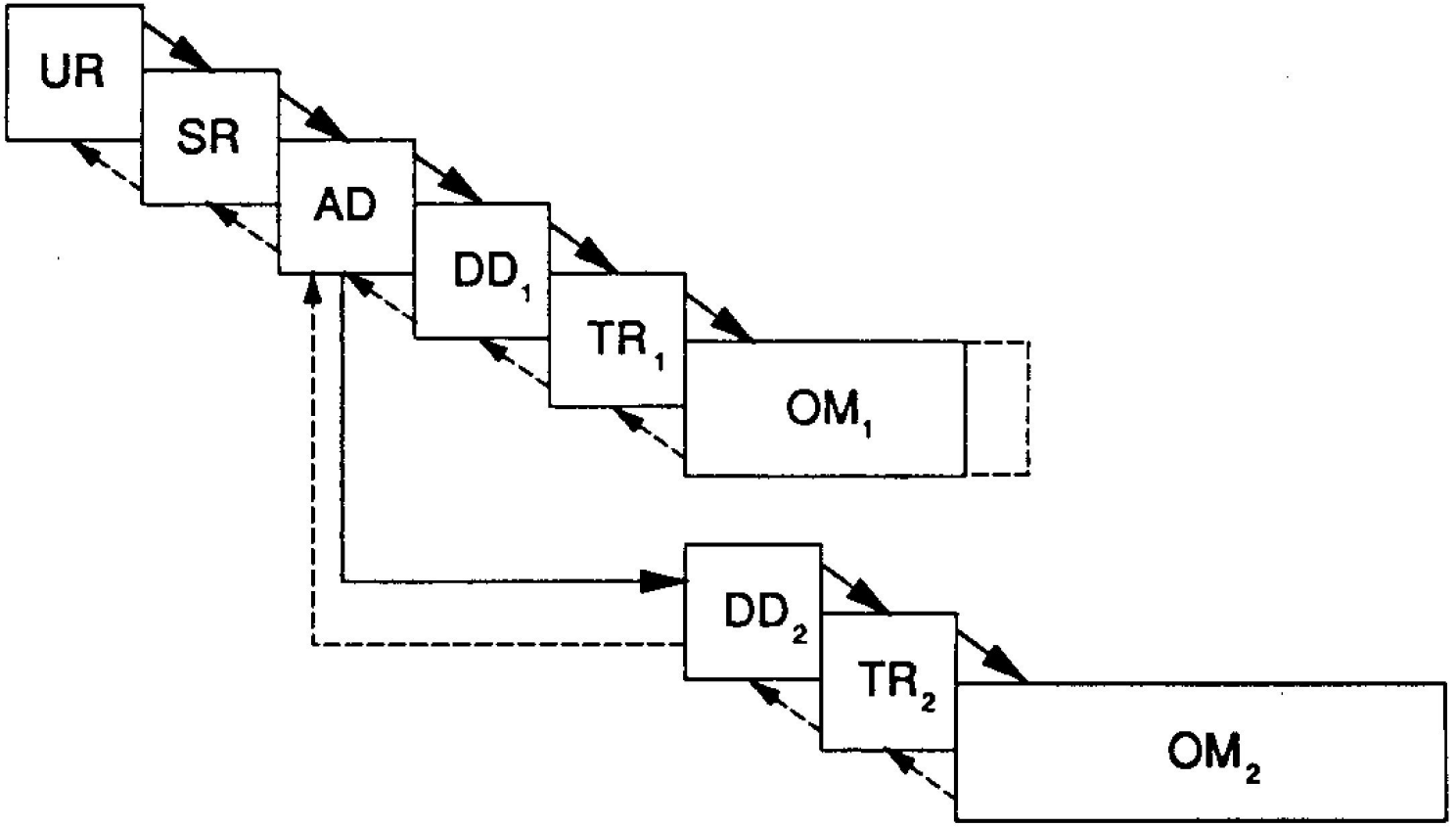
Итерационная модель



Описание итерационной модели

- Также эту модель называют итеративной моделью и инкрементальной моделью. Модель IID предполагает разбиение жизненного цикла проекта на последовательность итераций, каждая из которых напоминает «мини-проект».
- Цель каждой итерации — получение работающей версии программной системы, включающей функциональность, определённую интегрированным содержанием всех предыдущих и текущей итерации.
- Результат финальной итерации содержит всю требуемую функциональность продукта. Таким образом, с завершением каждой итерации продукт получает приращение — инкремент — к его возможностям, которые, следовательно, развиваются эволюционно.

Инкрементная модель



Итерационная модель

Достоинства:

- Главное достоинство данной модели, то что не требуется заранее тратить средства на разработку всего проекта.
- существует возможность поддерживать постоянный прогресс в ходе выполнения проекта, за счет итераций;
- снижаются затраты на первоначальную поставку программного продукта.

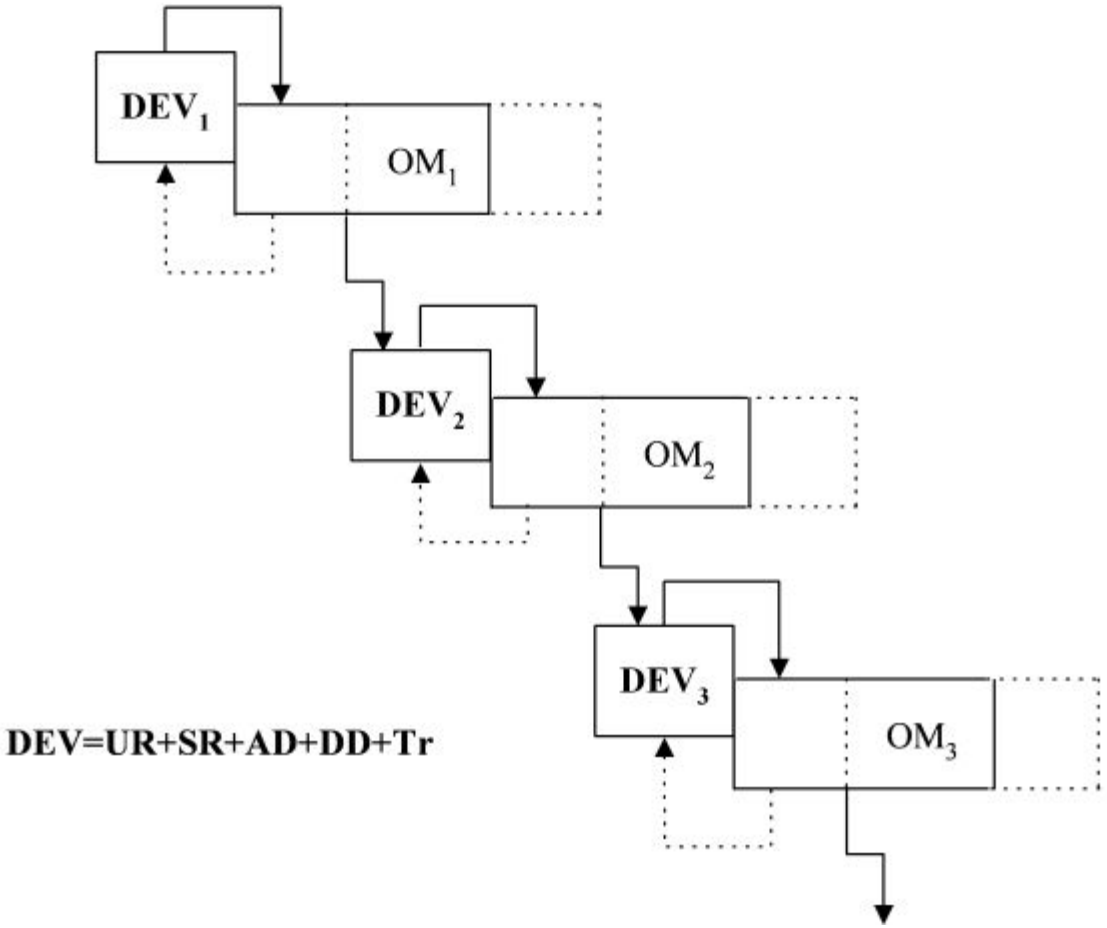
Недостатки.

- Одним из главных минусов является то что сразу надо определить полную функциональность;
- Модель не проста и необходимо хорошее планирование.

Эволюционная модель

- Эволюционная модель жизненного цикла не требует для начала полной спецификации требований. Вместо этого, создание начинается с реализации части функционала, становящейся базой для определения дальнейших требований. Этот процесс повторяется. Версия может быть неидеальна, главное, чтобы она работала. Понимая конечную цель, мы стремимся к ней так, чтобы каждый шаг был результативен, а каждая версия — работоспособна.

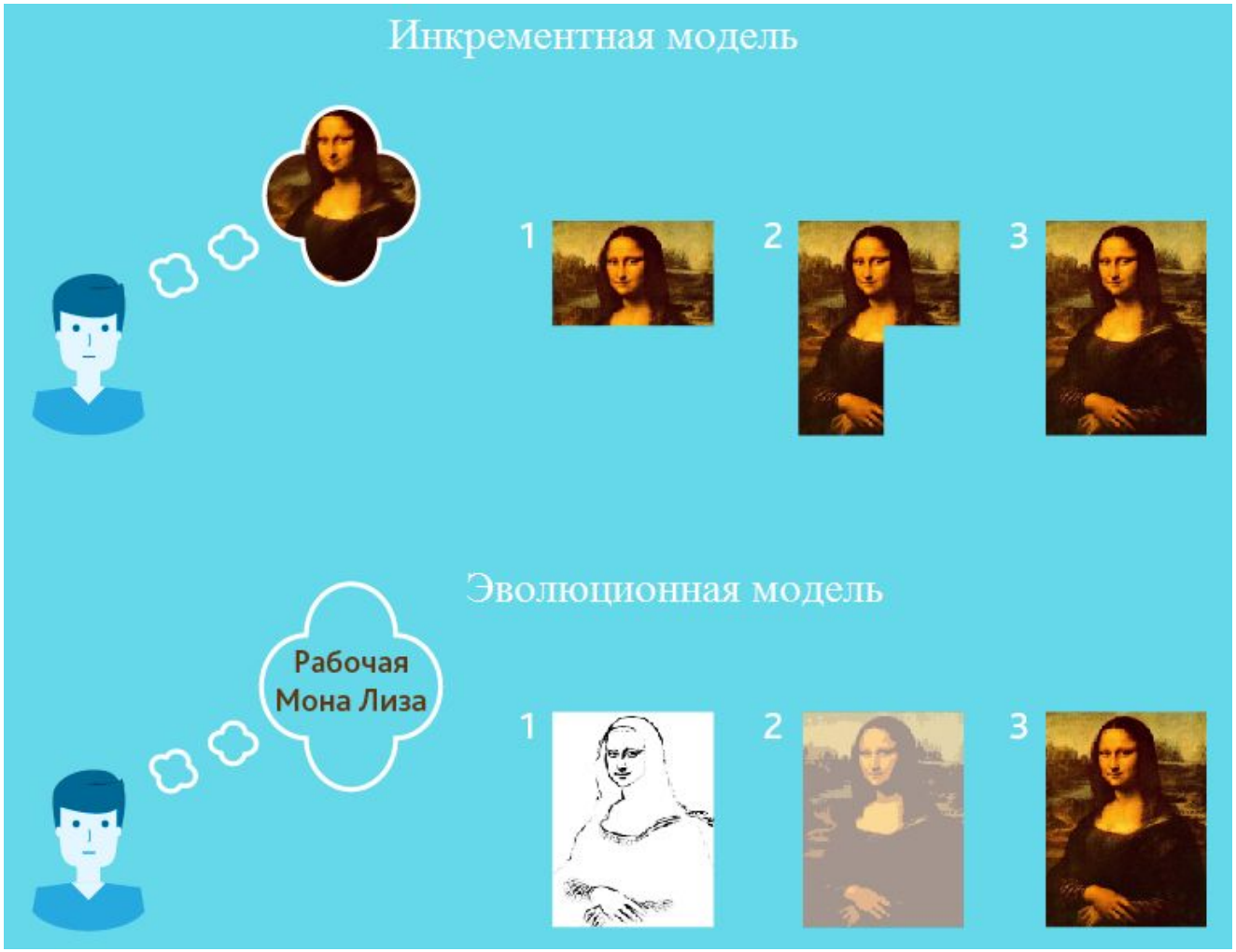
Эволюционная модель



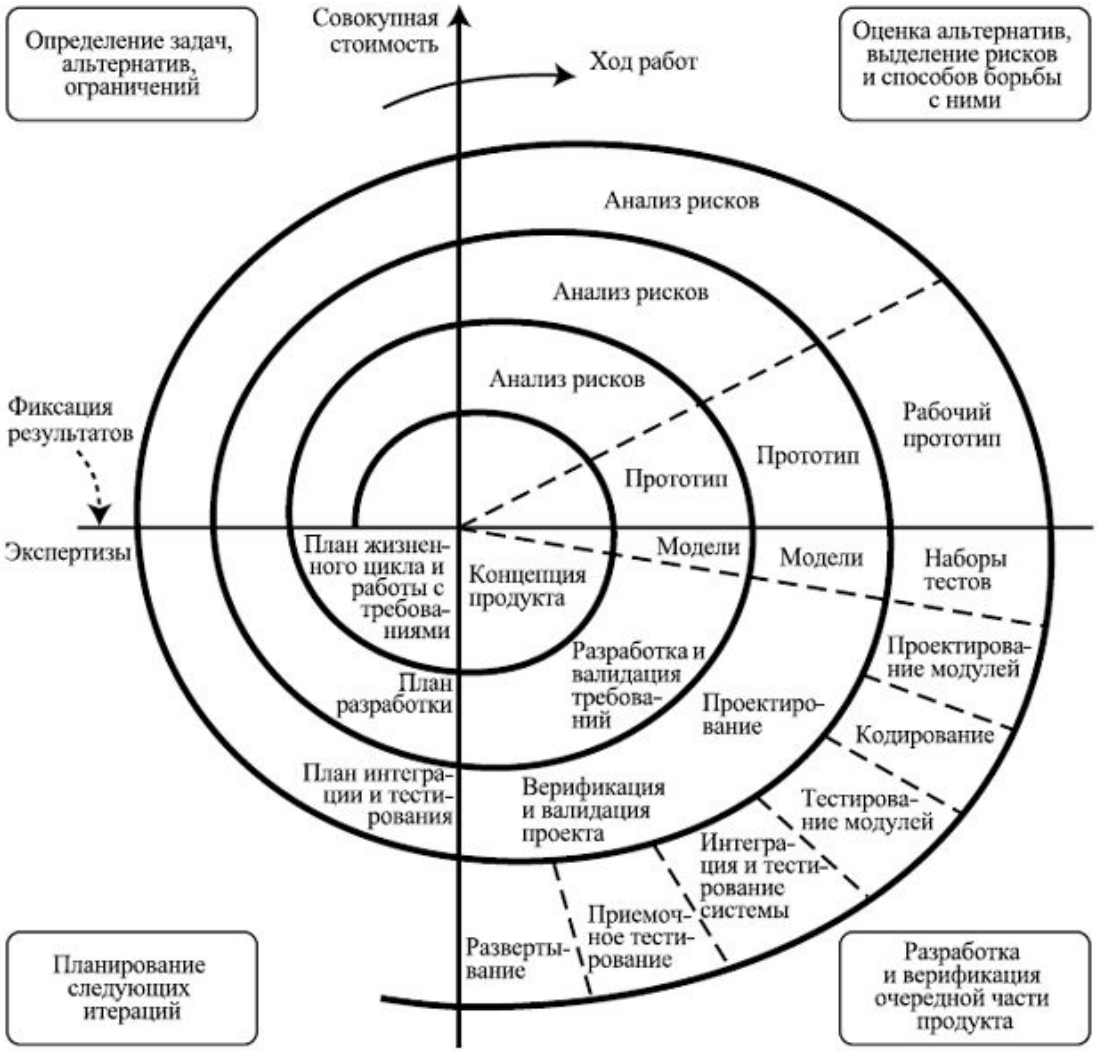
Когда использовать эволюционную модель

- Требования к конечной системе заранее четко определены и понятны.
- Проект большой или очень большой.
- Основная задача должна быть определена, но детали реализации могут эволюционировать с течением времени.

Сравнение инкрементной и эволюционной моделей



Спиральная модель



Описание спиральной модели

- При использовании этой модели ПО создается в несколько итераций (витков спирали).Каждая итерация соответствует созданию фрагмента или версии ПО.
- На выходе из очередного витка мы должны получить готовый протестированный прототип, который дополняет существующий билд.
- Главная особенность спиральной модели – концентрация на возможных рисках. Для их оценки даже выделена соответствующая стадия.

Основные типы рисков, которые могут возникнуть в процессе разработки ПО

- Нереалистичный бюджет и сроки;
- Дефицит специалистов;
- Частые изменения требований;
- Чрезмерная оптимизация;
- Низкая производительность системы;
- Несоответствие уровня квалификации специалистов разных отделов

Плюсы и минусы спиральной модели

- + улучшенный анализ рисков;
- + хорошая документация процесса разработки;
- + гибкость – возможность внесения изменений и добавления новой функциональности даже на относительно поздних этапах;
- + раннее создание рабочих прототипов.
- - может быть достаточно дорогой в использовании;
- - управление рисками требует привлечения высококлассных специалистов;
- - успех процесса в большой степени зависит от стадии анализа рисков;
- - не подходит для небольших проектов.

Когда использовать спиральную модель

- когда важен анализ рисков и затрат;
- крупные долгосрочные проекты с отсутствием четких требований или вероятностью их динамического изменения;
- при разработке новой линейки продуктов.

Гибкая методология разработки

- Серия подходов к разработке программного обеспечения, ориентированных на использование итеративной разработки, динамическое формирование требований и обеспечение их реализации в результате постоянного взаимодействия внутри рабочих групп, состоящих из специалистов различного профиля.
- Agile — семейство процессов разработки, а не единственный подход в разработке программного обеспечения, и определяется Agile Manifesto. Agile не включает практик, а определяет ценности и принципы, которыми руководствуются успешные команды.

Agile Manifesto

- содержит 4 основные идеи и 12 принципов. Основные идеи:
- люди и взаимодействие важнее процессов и инструментов;
- работающий продукт важнее исчерпывающей документации;
- сотрудничество с заказчиком важнее согласования условий контракта;
- готовность к изменениям важнее следования первоначальному плану.

Принципы, которые разъясняет Agile Manifesto

- удовлетворение клиента за счёт ранней и бесперебойной поставки ПО.
- приветствие изменений требований даже в конце разработки;
- частая поставка рабочего программного обеспечения;
- ежедневное общение заказчика с разработчиками на протяжении всего проекта;
- проектом занимаются мотивированные личности, которые обеспечены нужными условиями работы, поддержкой и доверием;
- рекомендуемый метод передачи информации — личный разговор;
- работающее программное обеспечение — лучший измеритель прогресса;
- спонсоры, разработчики и пользователи должны иметь возможность поддерживать постоянный темп на неопределённый срок;
- постоянное внимание улучшению технического мастерства и удобному дизайну;
- простота — искусство не делать лишней работы;
- лучшие технические требования, дизайн и архитектура получаются у самоорганизованной команды;
- постоянная адаптация к изменяющимся обстоятельствам.

Экстремальное программирование

- Это одна из гибких методологий разработки программного обеспечения.
- Двенадцать основных приёмов экстремального программирования могут быть объединены в четыре группы.

Короткий цикл обратной связи (Fine-scale feedback)

- XP предполагает написание автоматических тестов.
- Быстро сформировать приблизительный план работы и постоянно обновлять его.
- «Заказчик» в XP — это не тот, кто оплачивает счета, а конечный пользователь программного продукта.
- Весь код создается парами программистов, работающих за одним компьютером.

Непрерывный, а не пакетный процесс

- Если выполнять интеграцию разрабатываемой системы достаточно часто, то можно избежать большей части связанных с этим проблем.
- Рефакторинг— это методика улучшения кода без изменения его функциональности.
- Версии продукта должны поступать в эксплуатацию как можно чаще.

Понимание, разделяемое всеми

- Разрабатываемый продукт не следует проектировать заблаговременно целиком и полностью.
- Подбор хорошей метафоры.
- Все члены команды в ходе работы должны соблюдать требования общих стандартов кодирования.
- Каждый член команды несёт ответственность за весь исходный код.

Социальная защищенность программиста

- 40-часовая рабочая неделя.

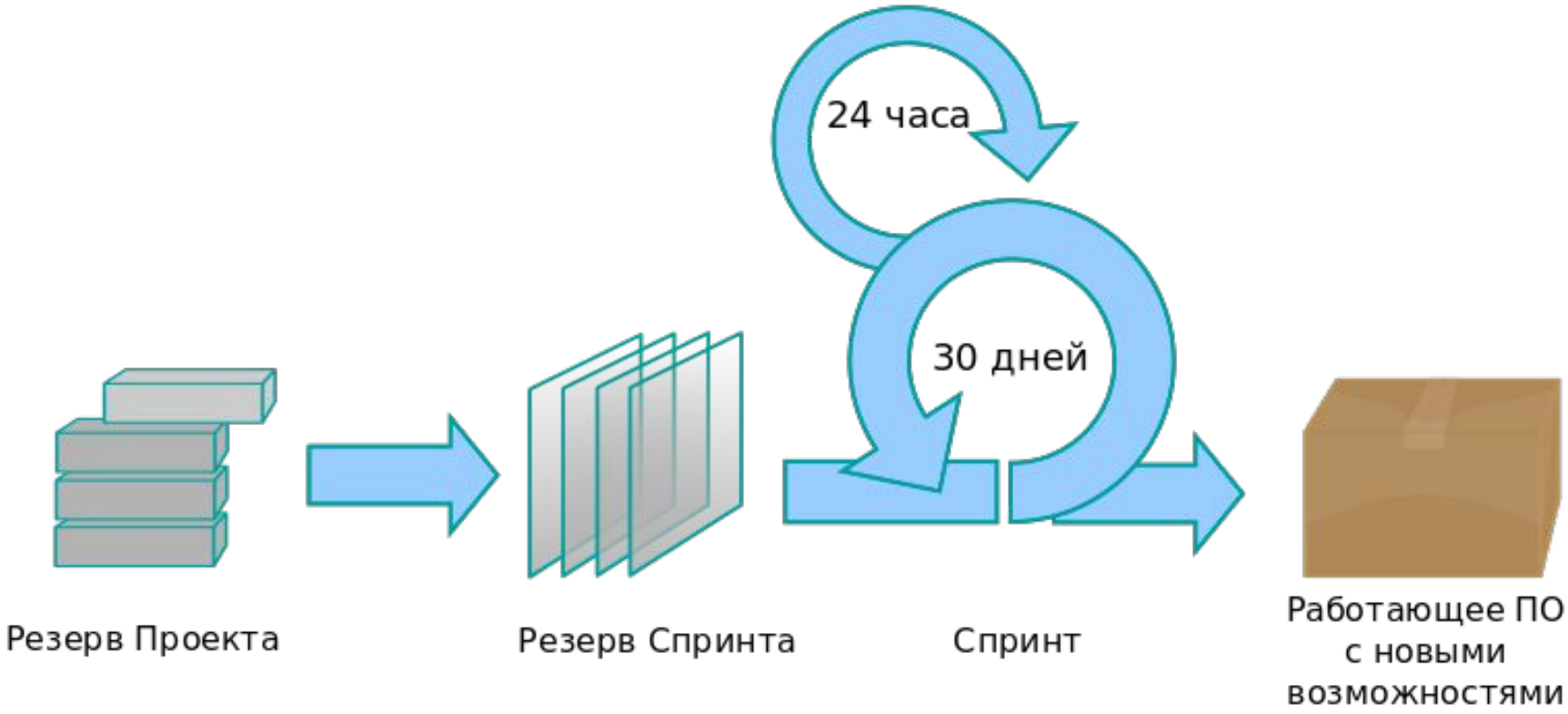
Scrum методология

- Методология управления проектами, активно применяющаяся при разработке информационных систем для гибкой разработки программного обеспечения.
- Scrum чётко делает акцент на качественном контроле процесса разработки.

Scrum

- Скрам (Scrum) — это набор принципов, на которых строится процесс разработки, позволяющий в жёстко фиксированные и небольшие по времени итерации, называемые спринтами (sprints), предоставлять конечному пользователю работающее ПО с новыми возможностями, для которых определён наибольший приоритет.
- Спринт — итерация в скраме, в ходе которой создаётся функциональный рост программного обеспечения. Жёстко фиксирован по времени. Длительность одного спринта от 2 до 4 недель.

Scrum-процессы



Scrum-процессы

- Резерв проекта — это список требований к функциональности, упорядоченный по их степени важности, подлежащих реализации.
- Резерв спринта — содержит функциональность, выбранную владельцем проекта из резерва проекта. Все функции разбиты по задачам, каждая из которых оценивается скрам-командой.

Kanban методология

- Канбан является Agile методологией разработки ПО.
- Канбан еще более “гибкая” методология, чем SCRUM и XP. Это значит, что она не подойдет всем командам и для всех проектов. Это значит, что команда должна быть еще более готовой к гибкой работе.

Kanban

- В Канбан нет таймбоксов ни на что (ни на задачи, ни на спринты).
- В Канбан задачи больше и их меньше.
- В Канбан оценки сроков на задачу опциональные или вообще их нет.
- В Канбан “скорость работы команды” отсутствует и считается только среднее время на полную реализацию задачи.

Канбан-доска

Правила Канбан

- Визуализируйте производство:
- Разделите работу на задачи, каждую задачу напишите на карточке и поместите на стену или доску.
- Используйте названные столбцы, чтобы показать положение задачи в производстве.

- Ограничивайте WIP (work in progress или работу, выполняемую одновременно) на каждом этапе производства.
- Измеряйте время цикла (среднее время на выполнение одной задачи) и оптимизируйте постоянно процесс, чтобы уменьшить это время.

Сравнение Scrum и Kanban
Сходства

- Оба – и Lean, и Agile.
- Оба используют вытягивающие системы планирования.
- Оба ограничивают НЗР.
- Оба используют прозрачность для обеспечения улучшения процесса.
- Оба ориентированы на ранние и частые поставки продукта.
- Оба полагаются на самоорганизующиеся команды.
- Оба требуют деления задач на более мелкие.
- В обоих случаях план релиза постоянно оптимизируется на основе эмпирических данных (производительности/ времени выполнения задачи).

Сравнение Scrum и Kanban Отличия

- Scrum
  - Обязательны ограниченные по времени итерации
  - Команда обязуется выполнить конкретный объём работы за эту итерцию
  - Производительность
- Kanban
  - Ограниченные по времени итерации не обязательны
  - Обязательства опциональны
  - Время выполнения задачи

Сравнение Scrum и Kanban Отличия

- Scrum
  - Кросс-функциональные команды обязательны
  - Задачи должны быть разбиты на более мелкие так, чтобы они были завершены в течение одного спринта
- Kanban
  - Кросс-функциональные команды, опциональны. Допустимы узкопрофильные команды
  - Нет каких-либо определенных размеров задач

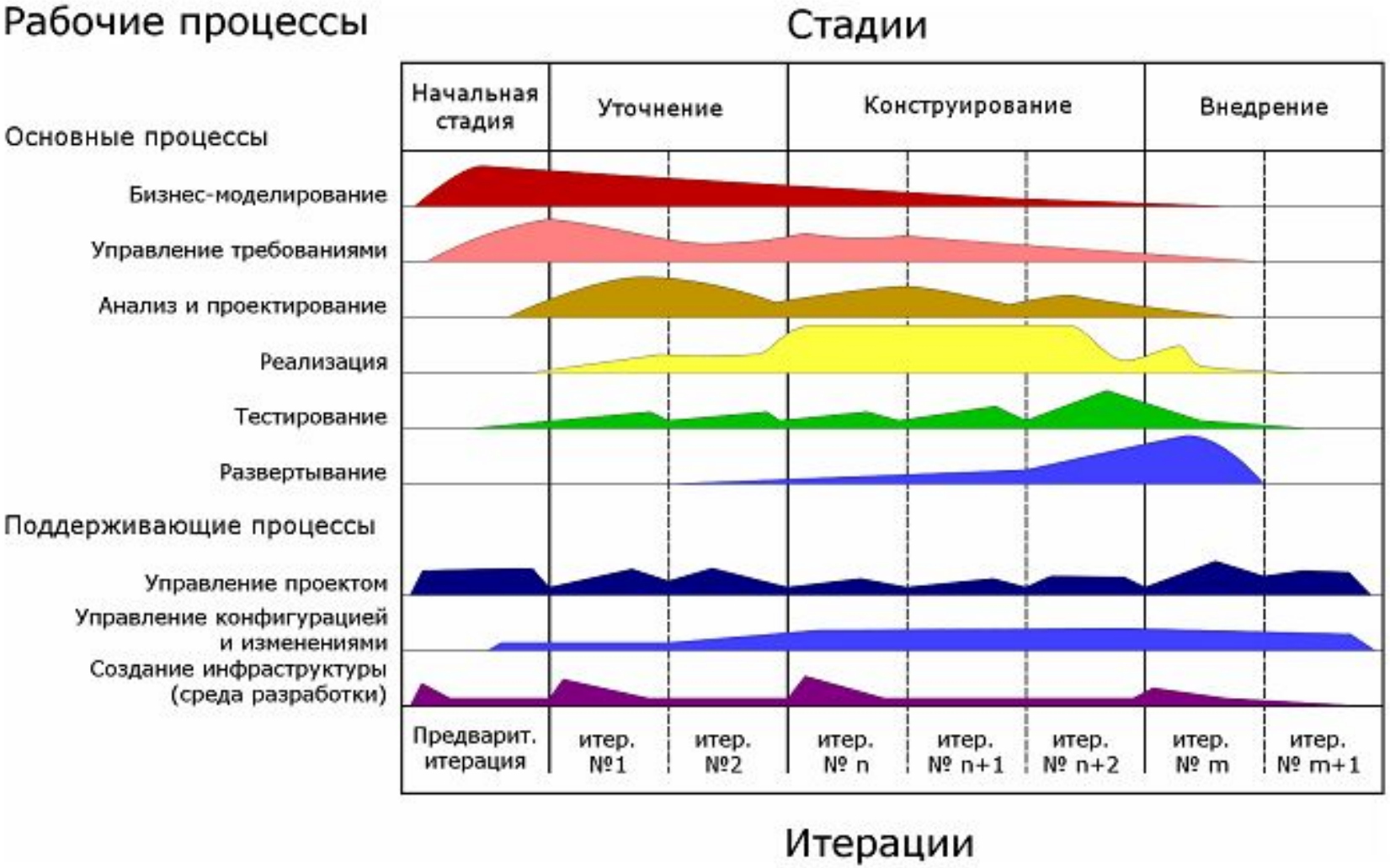
Rational Unified Process (RUP)

- Методология разработки программного обеспечения, созданная компанией Rational Software.
- RUP использует итеративную модель разработки. В конце каждой итерации проектная команда должна достичь запланированных на данную итерацию целей, создать или доработать проектные артефакты

Принципы RUP

- Ранняя идентификация и непрерывное (до окончания проекта) устранение основных рисков.
- Концентрация на выполнении требований заказчиков к исполняемой программе.
- Ожидание изменений в требованиях, проектных решениях и реализации в процессе разработки.
- Компонентная архитектура, реализуемая и тестируемая на ранних стадиях проекта.
- Постоянное обеспечение качества на всех этапах разработки проекта (продукта).
- Работа над проектом в сплочённой команде, ключевая роль в которой принадлежит архитекторам.

Процесс разработки по RUP



Начальная стадия (Inception)

- Формируются видение и границы проекта.
- Создается экономическое обоснование (business case).
- Определяются основные требования, ограничения и ключевая функциональность продукта.
- Создается базовая версия модели прецедентов.
- Оцениваются риски.

Уточнение (Elaboration) включает

- Документирование требований (включая детальное описание для большинства прецедентов).
- Спроектированную, реализованную и оттестированную исполняемую архитектуру.
- Обновленное экономическое обоснование и более точные оценки сроков и стоимости.
- Сниженные основные риски.

Построение (Construction)

- В фазе «Построение» происходит реализация большей части функциональности продукта.
- Фаза Построение завершается первым внешним релизом системы и вехой начальной функциональной готовности (Initial Operational Capability).

Внедрение (Transition)

- В фазе «Внедрение» создается финальная версия продукта и передается от разработчика к заказчику. Это включает в себя программу бета-тестирования, обучение пользователей, а также определение качества продукта.
- В случае, если качество не соответствует ожиданиям пользователей или критериям, установленным в фазе Начало, фаза Внедрение повторяется снова.

Спасибо за внимание!

Вопросы:

1) Какие стандарты, описывающие ЖЦ ПО вы знаете?
2) Какие типы процессов описывает стандарт ISO/IEC 12207?
3) Назовите основные фазы ЖЦ ПО по PSS_05_0.
4) Перечислите классические модели ЖЦ ПО.
5) Перечислите гибкие методологии ЖЦ ПО.