ПМ3
МЕТОДЫ И СРЕДСТВА ПРОЕКТИРОВАНИЯ ПО

Лекция 2.

Комитеты, непосредственно связанные с разработкой ПО.

Инженерия требований

Разработка требований. Документирование и организация требований.
Организация требований. Способы выявления требований.
Типы документов.
Управление требованиями:
Управление изменениями требований. Причины изменения требований.
Политика управления изменениями. Управление версиями требований.
Управление состояниями требований

Классификация технологических подходов

(модели жизненного цикла)

Комитеты, непосредственно связанные с разработкой ПО

- SEI
- IEEE
- OMG

Комитеты, непосредственно связанные с разработкой ПО (SEI)

- 1984 год – создание SEI (Software Engineering Institute) на базе университета Карнеги-Меллон в г.Питсбурге (США). Инициатор и главный спонсор – министерство обороны США. Основная задача – стандартизация в области программной инженерии, выработка критериев для сертификации надежных и зрелых компаний (что в первую очередь интересует Минобороны США для выполнения его заказов). Самые известные продукты – стандарт CMM, CMMI, разработки в области семейства программных продуктов (product lines). Эти продукты шагнули далеко за пределы военных разработок США, их использование и развитие стало международной деятельностью. Некоторые продукты SEI стандартизованы также ISO. На соответствие CMM/CMMI проводится сертификация.

Комитеты, непосредственно связанные с разработкой ПО (IEEE)

- 1963 год – создание **IEEE** (Institute of Electrical and Electronics Engineers). Ведет историю с конца XIX века, в контексте промышленной стандартизацией в США. Сейчас IEEE международная некоммерческая ассоциация специалистов в области техники, мировой лидер в области разработки стандартов по радиоэлектронике и электротехнике. Штаб-квартира в США, существуют многочисленные подразделения в разных странах, включая Россию. IEEE издаёт третью часть мировой технической литературы, касающейся применения радиоэлектроники, компьютеров, систем управления, электротехники, в том числе (январь 2008) 102 реферируемых научных журнала и 36 отраслевых журналов для специалистов, проводит в год более 300 крупных конференций, принимала участие в разработке около 900 действующих стандартов.

Комитеты, непосредственно связанные с разработкой ПО (OMG)

1989 год – группа американских IT-компаний (в том числе Hewlett Packard, Sun Microsystems, Canon) организовали **OMG** (Object Management Group). Сейчас включает около 800 компаний членов. Основное направление - разработка и продвижение объектно-ориентированных технологий и стандартов, в том числе для создания платформо-независимых программных приложений уровня предприятий. Известные стандарты CORBA, **UML**, MDA.

Все эти комитеты и организации включают программную инженерию в сферу своей деятельности, сотрудничают, выпускают совместные стандарты, используют наработки друг друга и т.д.

Инженерия требований

Инженерия требований.

Разработка требований

- Выявление требований
  - Исследование
  - Интервью
  - Семинар
  - Создание прототипа
  - Use Case
- Анализ требований
  - Уточнение требований
  - Структуризация
  - Установка приоритетов

Документирование и организация требований

- Состав и распределение работ
- Спецификация требований
- Концепция эксплуатации
- Начальный план разработки ПО
- Критерии принятия работ

Управление требованиями

- Изменение требований
  - Предложение изменений
  - Анализ изменений
  - Принятие решений
  - Обновление требований
  - Обновление планов
- Контроль версий требований
  - Управление состояниями требований
  - Прослеживаемость требований

Выводы по разработке требований

- Требования необходимо
  - Собрать
  - Организовать
  - Документировать
  - Изменить
  - Проверить
  - Добавить
  - Уничтожить
  - И т.д.
- Они имеют свой жизненный цикл

Инженерия требований.

Разработка требований

- Выявление требований
  - Исследование
  - Интервью
  - Семинар
  - Создание прототипа
  - Use Case
- Анализ требований
  - Уточнение требований
  - Структуризация
  - Установка приоритетов

Документирование и организация требований

- Состав и распределение работ
- Спецификация требований
- Концепция эксплуатации
- Начальный план разработки ПО
- Критерии принятия работ

Управление требованиями

- Изменение требований
  - Предложение изменений
  - Анализ изменений
  - Принятие решений
  - Обновление требований
  - Обновление планов
- Контроль версий требований
  - Управление состояниями требований
  - Прослеживаемость требований

Документирование и организация требований

Как документировать разные требования ?

- Бизнес-требования

  документ о представлении/границах проекта
- Требования пользователей

  варианты использования
- Функциональные требования (результат должен быть один)

  спецификации (соглашение между исполнителем и заказчиком) требований к ПО

Организация требований

- Группирование требований

  требования объединяются в родственные группы (общих правил нет)

- Иерархическая структуризация требований

  подчинение

  уточнение

Способы документирования

- Документ на естественном языке (понятном заказчику и исполнителю)
- Графические модели
  - Диаграммы
  - Графы (временные…)
  - Схемы
  - Потоки
- Формальные спецификации

  (помогают сгенерировать если не код, то тесты;

  позволяют проверить на полноту, непротиворечивость)

Интенсивно развивается контрактное программирование в языке программирования Eiffel (объектно-ориентированный язык программирования с алголоподобным синтаксисом, разработанный Бертраном Мейером).

Типы документов

Создаются все или некоторые из документов (в зависимости от размера проекта и используемой методологии)

- Состав и распределение работ (кто какие артефакты когда и зачем создает)
- **Спецификация требований** (создается всегда и является основным соглашением между разработчиком и заказчиком)
- Концепция эксплуатации (описывает то, как система будет использоваться какие функции будет выполнять, какие существуют роли, возможные режимы)
- Начальный план разработки ПО (укрупненный план работы над проектом)
- Критерии принятия работ (показывают как и по каким правилам будет сделана работа и приниматься заказчиком)

Спецификация требований

Фундамент всего последующего планирования, проектирования, реализации проекта

Основание для тестирования проекта

Основание для документирования проекта

НО: не должна содержать деталей проектирования, реализации, тестирования и управления проектом

Является исходным соглашением между заказчиком и исполнителем

Состав и распределение работ

Распределяет ответственности между заинтересованными сторонами проекта (задает правила игры):

Кто создает, что и когда

Кто тестирует, что, как и когда

Кто платит, за что и как

Кто докладывает кому

Кто принимает/утверждает завершение работ или этапов

Кто, как и когда санкционирует изменения

И т.п.

Концепция эксплуатации

Описание того, как система должна работать или будет использоваться

- Какие функции будут использоваться, как и кем, в каких условиях
- Как будет происходить ввод/вывод данных
- Как система взаимодействует с другими системами

Этот документ задает основу для разработки вариантов использования

ПО

Высокоуровневый и приблизительный план разработки задает:

- **Основные документы**

- **Точки принятия решений** между заказчиком и исполнителем: например, о продолжении или завершении работ между этапами и как это связано с финансовыми обязательствами

- **Поставляемые артефакты** (исходный код, объектные модули, бинарные модули, дистрибутивные пакеты и т.д.)

- **Этапы работ и контрольные точки** (когда заказчик может контролировать работу (зависит от выбранной методологии)). Этапы показывают, когда происходит сдача определенных этапов, и позволяют сделать вывод об успешности или неуспешности проекта. Точки принятия решений более крупные точки и необязательно связаны с контрольными точками

- **Графики платежей**

Артефакт

- - это любой искусственно созданный элемент программной системы. Например, исполняемые файлы, исходные тексты, веб-страницы, справочные файлы, сопроводительные документы, файлы с данными, модели и многое другое, являющееся физическим носителем информации. Другими словами, артефактами являются те **информационные элементы**, которые тем или иным способом **используются при работе программной системы и входят в ее состав**.

- В терминах RUP участники проектной команды создают так называемые артефакты (work products), выполняя задачи (tasks) в рамках определенных ролей (roles). Артефактами являются **спецификации, модели, исходный код** и т.п.

Критерии принятия работ

Содержит

Критерии принятия работ (каким образом сделанный ПП будет проходить окончательное приемочное, промежуточное испытания, или испытания отдельных этапов)

Содержит критерии, показывающие, что работа будет принята

Сдача-приемка – это конечный набор тестов, по которым делается вывод, что проект соответствует или нет спецификации.

Проверяется ли полностью спецификация?

Спецификация отвечает на вопрос: что должно быть сделано.

Критерии – как проверить, что это совпало?

Критерии принятия работ (Методика испытаний. Программа испытаний)

- Критерии должны быть приняты всеми заинтересованными лицами
- Критерии должны быть четкими (очевидными) и недвусмысленными
- Критерии должны быть количественными, а не качественными (быстрый, удобный и т.д.)

Инженерия требований.

Разработка требований

- Выявление требований
  - Исследование
  - Интервью
  - Семинар
  - Создание прототипа
  - Use Case
- Анализ требований
  - Уточнение требований
  - Структуризация
  - Установка приоритетов

Документирование и организация требований

- Состав и распределение работ
- Спецификация требований
- Концепция эксплуатации
- Начальный план разработки ПО
- Критерии принятия работ

Управление требованиями

- Изменение требований
  - Предложение изменений
  - Анализ изменений
  - Принятие решений
  - Обновление требований
  - Обновление планов
- Контроль версий требований
  - Управление состояниями требований
  - Прослеживаемость требований

Управление требованиями

- Цели:
  - Изменение требований
  - Контроль версий требований
  - Контроль состояний требований
  - Прослеживаемость
  - Совершенствование процессов управления

Управление требованиями

- Управление изменениями
  - Предложение изменений
  - Анализ изменений
  - Принятие решений
  - Обновление требований
  - Обновление планов
- Контроль версий
  - Определение схемы идентификации версий
  - Определение версий спецификаций требований
  - Определение версий отдельных
  - Контроль состояния требований
  - Регистрация состояния требований
- Прослеживание требований
  - Определение связей с другими требованиями
  - Определение связей с другими элементами системы

Управление изменениями требований

Причины изменения требований

Условия возможности изменений

Политика управления изменениями

Анализ влияния изменения

Принятие/непринятие изменений

Причины изменения требований

- Заказчик
  - Не понравилось после просмотра
  - Передумал
  - Забыл
- Рынок
  - Такой продукт уже не продать
  - Нужно выйти на рынок прямо сейчас, иначе этот продукт не продать
- Разработчики
  - Требования неправильно поняты
  - Требования плохо определены
  - Требования не были поняты
  - Сработали архитектурные риски

Условия возможности изменений требований для разных стратегий

- Водопадные стратегии – не возможно
- Инкрементные стратегии – возможно с некоторыми ограничениями
- Эволюционные стратегии – возможно

Политика управления изменениями

- Должен быть принят процесс контроля за изменениями
- Все изменения должны следовать процессу или не рассматриваться
- Для неучтенных требований не выполняется никаких действий, кроме исследования осуществимости
- Все запросы за изменениями должны быть одобрены советом (один человек или группа) по управлению изменениями
- Содержание запроса на изменение должно быть доступно всем заинтересованным лицам проекта
- Начальный текст запроса должен быть неизменным (каждое изменение – отдельная транзакция
- Анализ воздействия должен проводиться для каждого изменения
- Каждое одобренное изменение (добавление требования) должно *прослеживаться* до реализации
- Обоснование каждого одобрения/отказа на изменение должно быть задокументировано

Анализ влияния изменения

- Выявление последствий внесения изменений
- Определение всех сущностей (файлы, модели, артефакты, документы), которые нуждаются в модификации, если изменение будет принято
- Определение задач, необходимых для реализации изменения
- Оценка усилий для завершения этих задач
- Оценка этих задач на критическом пути проекта
- Оценка влияния на график работ
- Оценка влияния на стоимость
- Оценка приоритета изменения, учитывая:
  - Достоинства
  - Недостатки
  - Затраты
  - Риски

Варианты решения на запрос об изменении требований

- Отложить низкоприоритетные требования
- Привлечь дополнительных сотрудников
- Организовать краткосрочную сверхурочнцую работу
- Изменить график работ
- Пожертвовать качеством продукта за счет реализованных возможностей
- ОТКАЗАТЬ!

Управление версиями требований

- Требования могут устаревать
- Требования могут быть противоречивыми
- Контроль версий документов
  - С помощью любой системы контроля версий
- Контроль версий требований
  - Создание начальных версий требований
  - Ведение истории изменений
  - Авторизированный доступ к изменению требований

Управление состояниями требований

Состояние	Определение
Предложено	Требование было выставлено авторизированным источником
Одобрено	Требование было проанализировано и одобрено для определенной версии
Реализовано	Код, реализующий требования, написан
Проверено	Корректная функциональность требования подтверждена версией продукта. Требование может быть прослежено до варианта тестирования
Удалено	Ранее одобренное требование было исключено из базисного списка. Причина удаления – задокументирована.
Отклонено	Предложенное требование – отклонено. Причина отклонения – задокументирована.

Отслеживание состояний требований

(Узнать, в каком состоянии находится требование)

- Показатель прогресса проекта
- Используется при анализе изменений
- Обосновывает некоторые решения, принятые во время разработки
- Обычно измеряется в процентах завершенности работ (делается примерно на глаз)
  - Часто может вводить в заблуждение

Прослеживание требований (трассировка)

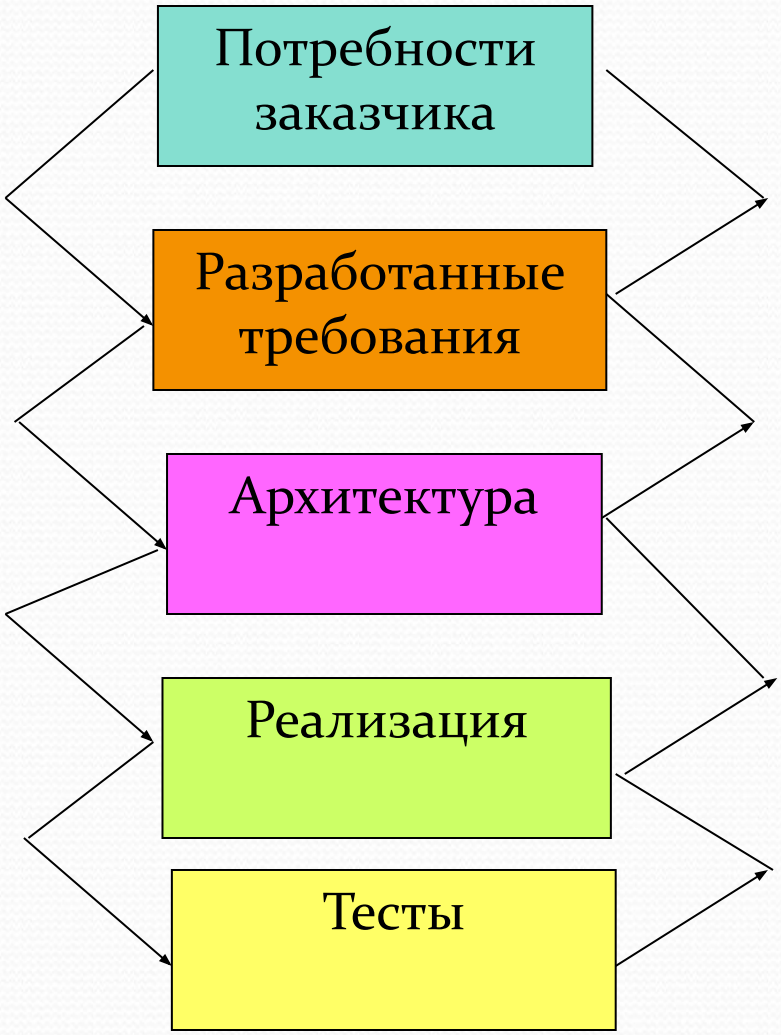
Цели:

- Получить подтверждение, что цели были реализованы
- Убедиться, что требования были оттестированы
- Иметь трассы всех требований от заказчика до тестовых случаев

Если система управления проектами позволяет увидеть всю цепочку от формулировки требования до его проверки, то легко анализировать систему, исследовать влияние изменения требований.

Прослеживание требований

Рассмотрим 5 укрупненных артефакта.
Идеально, когда есть 8 типов связей.
Тогда мы сможем, увидев потребности заказчика, выраженные, например, в Use Case диаграмме, найти, как они отображаются в спецификациях требований, а по спецификации понять, каким потребностям заказчика соответствует .
Как эти требования попали в конкретную архитектуру и обратно.
Как элемент архитектуры реализован в коде и какому куску кода соответствует какой элемент архитектуры.
Как каждому кусочку кода соответствует тест и наоборот.
Для этих целей используется матрица (таблица) прослеживания требований.



Матрица прослеживания требований

Пример 1

Требование пользователя 1	Функциональное требование 2	Элемент дизайна 3	Часть кода 4	Тестовый сценарий 5
UC-16	Catalog.item.sort	Class Catalog	Catalog sort ()	Sort 7 Sort 8
UC -20	2.7, 4	Class Order		Order 5
UC-27	SA 2.1	Class Entity		Security 31 Security 32 Security 33

Позволяет посмотреть, как изначальные требования пользователя реализовалось к какой-то элемент реализации или дизайна.

- 1. Требование пользователя в виде Use Case
- 2. Как попало из спецификаций
- 3. В какой элемент архитектуры попало
- 4. Где было представлено в виде кода
- 5. Какие тестовые случаи ему соответствуют

Матрица прослеживания требований

- Пример 2

Функциональное	Сценарий тестирования					
требование	TC1	TC12	TC3	TC4	TC5	TC6
FC1	+					
FC2		+				+
FC3					+	
FC4			+	+		

Инженерия требований. Резюме

Разработка требований

- Выявление требований
  - Исследование
  - Интервью
  - Семинар
  - Создание прототипа
  - Use Case
- Анализ требований
  - Уточнение требований
  - Структуризация
  - Установка приоритетов

Документирование и организация требований

- Состав и распределение работ
- Спецификация требований
- Концепция эксплуатации
- Начальный план разработки ПО
- Критерии принятия работ

Управление требованиями

- Изменение требований
  - Предложение изменений
  - Анализ изменений
  - Принятие решений
  - Обновление требований
  - Обновление планов
- Контроль версий требований
  - Управление состояниями требований
  - Прослеживаемость требований

Программные средства управления требованиями

Программные средства управления требованиями

- Существует более 40 средств управления требованиями.
- Наиболее функциональные:
  - IBM Rational DOORS
  - IBM Rational Requisite Pro
  - Borland
    - Caliber DefineIT
    - CaliberRM

Функции инструментальных средств управления требованиями

- Захват/идентификация требований (на вход подаются структурированные документы, например в Word)
- Выделение структуры и организация требований
- Трассировка требований
- Управление конфигурациями
- Формирование отчетов
- Групповая работа
- Интерфейсы к другим средствам
- Системное окружение
- Пользовательский интерфейс
- Поддержка

Краткая характеристика методологий проектирования ПО

Что влияет на успешность проекта?

- Решаемая задача
- Заказчик
- Со стороны разработчика
  - Команда разработки
  - Инфраструктура
  - ***Выбранная методология проектирования ПО***

Методологии проектирования ПО определяются

- Составом и последовательностью работ
- Ролью участников проекта
- Составом и шаблонами документов
- Организацией и управлением требованиями
- Порядком контроля и проверки качества
- Способом взаимодействия участников

Классическая модель проектирования ПО

- Предложена в 1960-х годах, впервые описана в 1970г. В. Ройсон
- Водопадный (однократный) подход
- Относится к прогнозирующим методологиям
- Предполагает полное описание всех требований на момент старта проекта
- Требования не могут меняться в процессе проектирования
- Программный продукт появляется по окончании проектирования

Классическая модель проектирования ПО

- Анализ и планирование
  - Сбор требований
  - Анализ требований
  - Планирование проекта
- Проектирование
  - Разработка архитектуры
  - Разработка моделей данных
  - Разработка алгоритмов
- Реализация
  - Кодирование
  - Отладка
- Тестирование/верификация
- Сопровождение
  - Внедрение
  - Эксплуатация
  - Внесение изменений

Классическая модель проектирования ПО

- Достоинства:
  - Имеется план и график по всем этапам конструирования
  - Ход конструирования упорядочен
  - Имеется богатый опыт использования
- Недостатки:
  - Не всегда соответствует реальным проектам (отсутствует гибкость)
  - Часто всех требований на начальном этапе нет
  - Результат доступен только в конце

Методологии и технологии проектирования

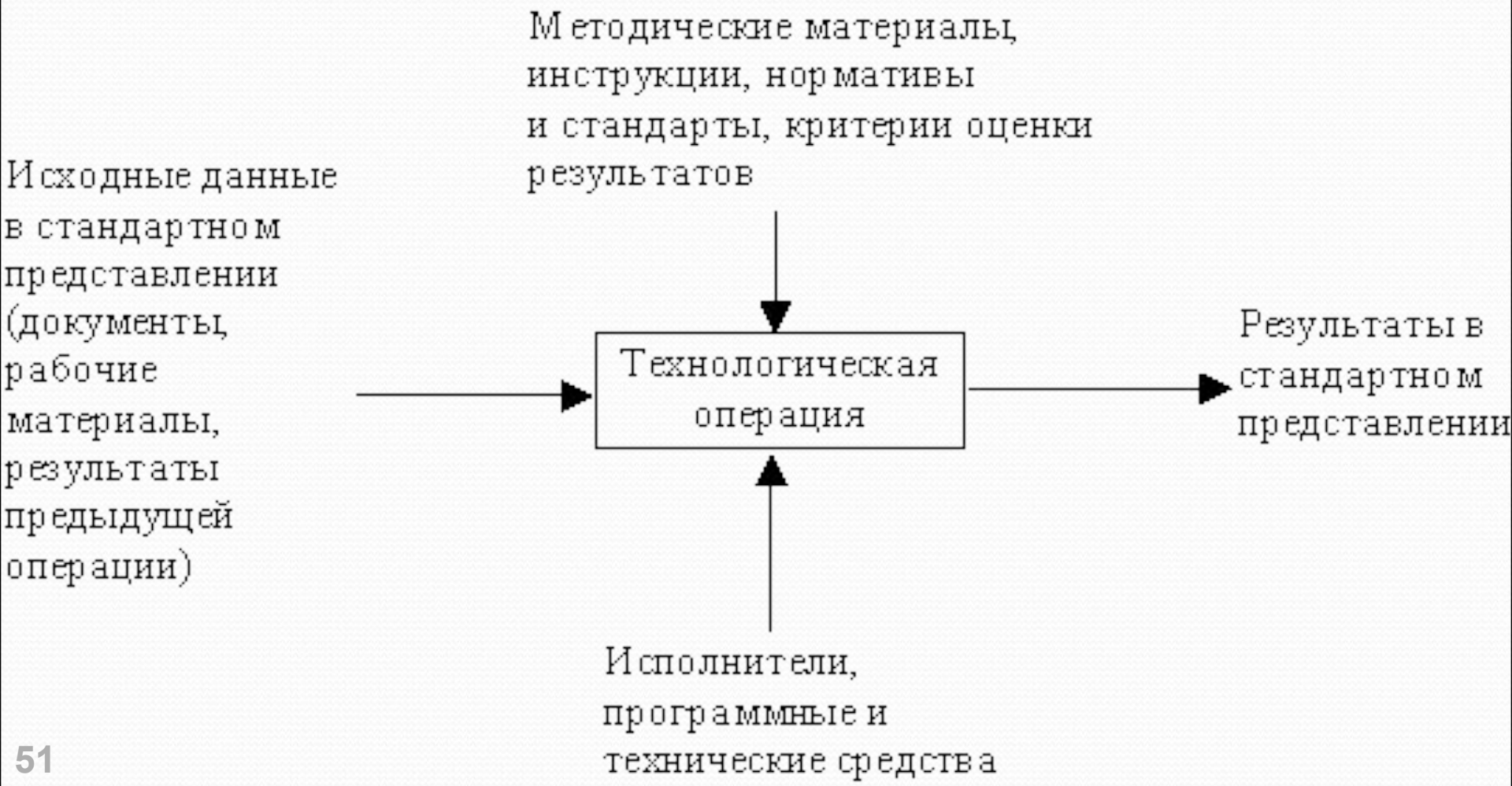
Методологии, технологии и инструментальные средства проектирования (CASE-средства) составляют основу проекта любой ИС. Методология реализуется через конкретные технологии и поддерживающие их стандарты, методики и инструментальные средства, которые обеспечивают выполнение процессов ЖЦ.

Технология проектирования

определяется как совокупность трех составляющих:

- пошаговой процедуры, определяющей последовательность технологических операций проектирования;
- критериев и правил, используемых для оценки результатов выполнения технологических операций;
- нотаций (графических и текстовых средств), используемых для описания проектируемой системы.

Представление технологической операции проектирования



технологии проектирования невозможно без выработки ряда стандартов

- Реальное применение любой технологии проектирования, разработки и сопровождения ИС в конкретной организации и конкретном проекте невозможно без выработки ряда стандартов (правил, соглашений), которые должны соблюдаться всеми участниками проекта. К таким стандартам относятся следующие:
- стандарт проектирования;
- стандарт оформления проектной документации;
- стандарт пользовательского интерфейса.

Стандарт проектирования устанавливает

- набор необходимых моделей (диаграмм) на каждой стадии проектирования и степень их детализации;
- правила фиксации проектных решений на диаграммах, в том числе: правила именования объектов (включая соглашения по терминологии), набор атрибутов для всех объектов и правила их заполнения на каждой стадии, правила оформления диаграмм, включая требования к форме и размерам объектов, и т. д.;
- требования к конфигурации рабочих мест разработчиков, включая настройки операционной системы, настройки CASE-средств, общие настройки проекта и т. д.;
- механизм обеспечения совместной работы над проектом, в том числе: правила интеграции подсистем проекта, правила поддержания проекта в одинаковом для всех разработчиков состоянии (регламент обмена проектной информацией, механизм фиксации общих объектов и т.д.), правила проверки проектных решений на непротиворечивость и т. д.

Стандарт оформления проектной документации устанавливает

- комплектность, состав и структуру документации на каждой стадии проектирования;
- требования к ее оформлению (включая требования к содержанию разделов, подразделов, пунктов, таблиц и т.д.),
- правила подготовки, рассмотрения, согласования и утверждения документации с указанием предельных сроков для каждой стадии;
- требования к настройке издательской системы, используемой в качестве встроенного средства подготовки документации;
- требования к настройке CASE-средств для обеспечения подготовки документации в соответствии с установленными требованиями.

Стандарт интерфейса пользователя устанавливает

- правила оформления экранов (шрифты и цветовая палитра), состав и расположение окон и элементов управления;
- правила использования клавиатуры и мыши;
- правила оформления текстов помощи;
- перечень стандартных сообщений;
- правила обработки реакции пользователя.

Классификация технологических подходов (модели жизненного цикла)

представления жизненного цикла программы

Жизненный цикл программы - это весь период ее разработки и эксплуатации, начиная с момента возникновения замысла и заканчивая прекращением всех видов ее использования.

Технология программирования изучает технологические процессы и порядок их прохождения - стадии (с использованием знаний, методов и средств). Знания, методы и средства могут использоваться в разных процессах и, следовательно, технологиях. Технологии удобно характеризовать в двух измерениях - вертикальном (представляющем процессы) и горизонтальном (представляющем стадии).

Процесс - совокупность взаимосвязанных действий, преобразующих некоторые входные данные в выходные. Процессы состоят из набора действий, а каждое действие из набора задач. Вертикальное измерение отражает статические аспекты процессов и оперирует такими понятиями, как рабочие процессы, действия, задачи, результаты деятельности и исполнители.

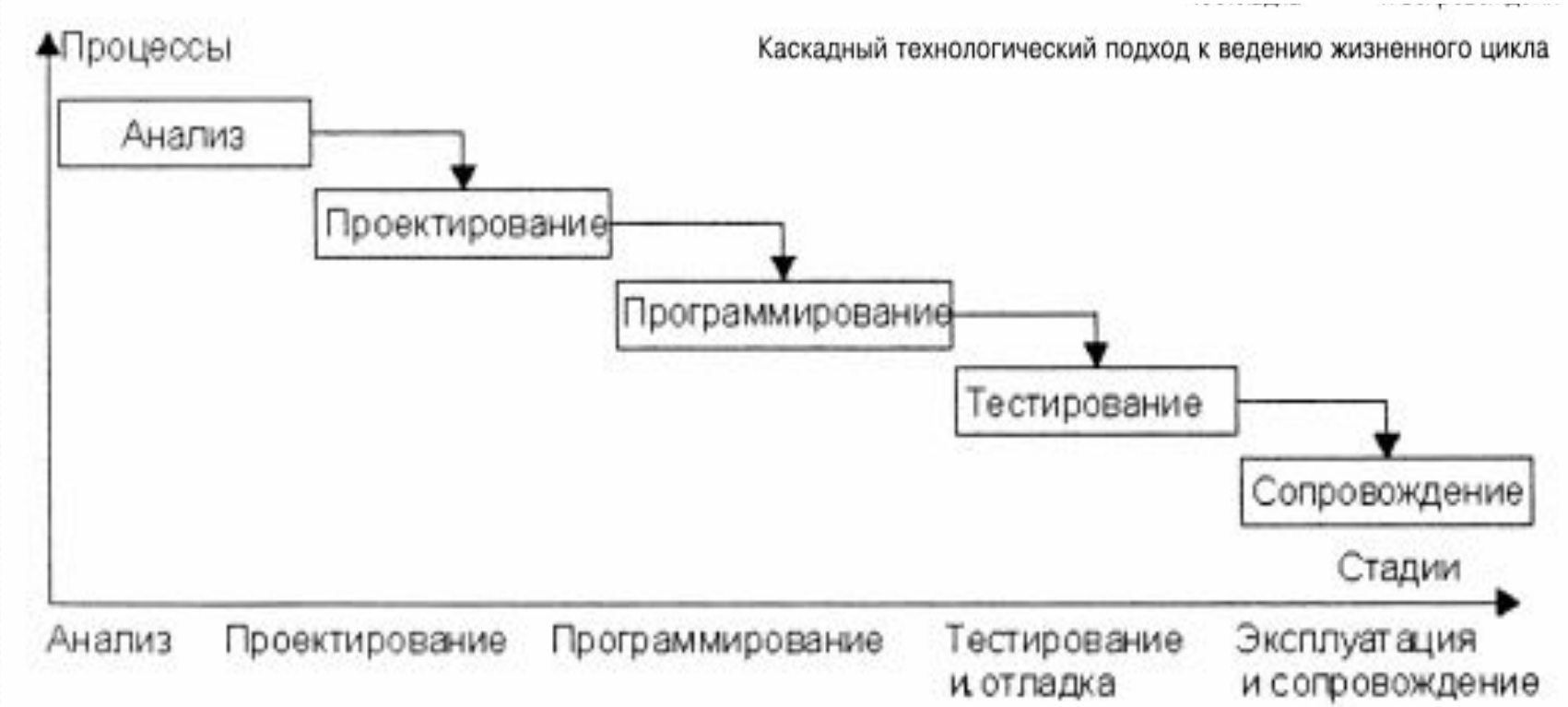
Стадия - часть действий по созданию программного обеспечения, ограниченная некоторыми временными рамками и заканчивающаяся выпуском конкретного продукта, определяемого заданными для данной стадии требованиями. Стадии состоят из этапов, которые обычно имеют итерационный характер. Иногда стадии объединяют в более крупные временные рамки, называемые фазами. Итак, горизонтальное измерение представляет время, отражает динамические аспекты процессов и оперирует такими понятиями, как фазы, стадии, этапы, итерации и контрольные точки.

Методология проектирования (технологический подход) определяется спецификой комбинации стадий и процессов, ориентированной на разные классы программного обеспечения и на особенности коллектива разработчиков.

Технологические подходы

- **1. Подходы со слабой формализацией**
- **2 Строгие (классические, жесткие, предсказуемые) подходы**
  - 2.1. Каскадные технологические подходы.
    - 2.1.1. Классический каскадный подход
    - 2.1.2. Каскадно-возвратный подход
    - 2.1.3. Каскадно-итерационный подход
    - 2.1.4. Каскадный подход с перекрывающимися процессами
    - 2.1.5. Каскадный подход с подпроцессами
    - 2.1.6. Спиральная модель
  - 2.2. Каркасные подходы.
    - 2.2.1. Рациональный унифицированный процесс (RUP)
  - 2.3. Генетические подходы.
    - 2.3.1. Синтезирующее программирование
    - 2.3.2. Сборочное (расширяемое) программирование
    - 2.3.3. Конкретизирующее программирование
  - 2.4. Подходы на основе формальных преобразований.
    - 2.4.1. Технология стерильного цеха
    - 2.4.2. Формальные генетические подходы
- **3 Гибкие (адаптивные, легкие) подходы**
  - 3.1. Ранние технологические подходы быстрой разработки (RAD)
    - 3.1.1. Эволюционное прототипирование
    - 3.1.2.Итеративная разработка
    - 3.1.3. Постадийная разработка
  - 3.2. Адаптивные подходы.
    - 3.2.1. Экстремальное программирование (XP)
    - 3.2.2. Адаптивная разработка
  - 3.3. Подходы исследовательского программирования.
    - 3.3.1. Компьютерный дарвинизм

Простейшее представление жизненного цикла программы



Здесь на каждой стадии выполняется единственный процесс. При разработке и создании больших программ такая схема недостаточно корректна (да и просто нереалистична). Но её можно взять за основу для многих других технологических подходов к ведению жизненного цикла.

Классификация технологических подходов

- Donald Ervin Knuth

Кнут Дональд Эрвин

Американский учёный, эмерит-профессор Стэнфордского университета и нескольких других университетов в разных странах, преподаватель и идеолог программирования, автор 19 монографий и более 160 статей, разработчик нескольких известных программных технологий.

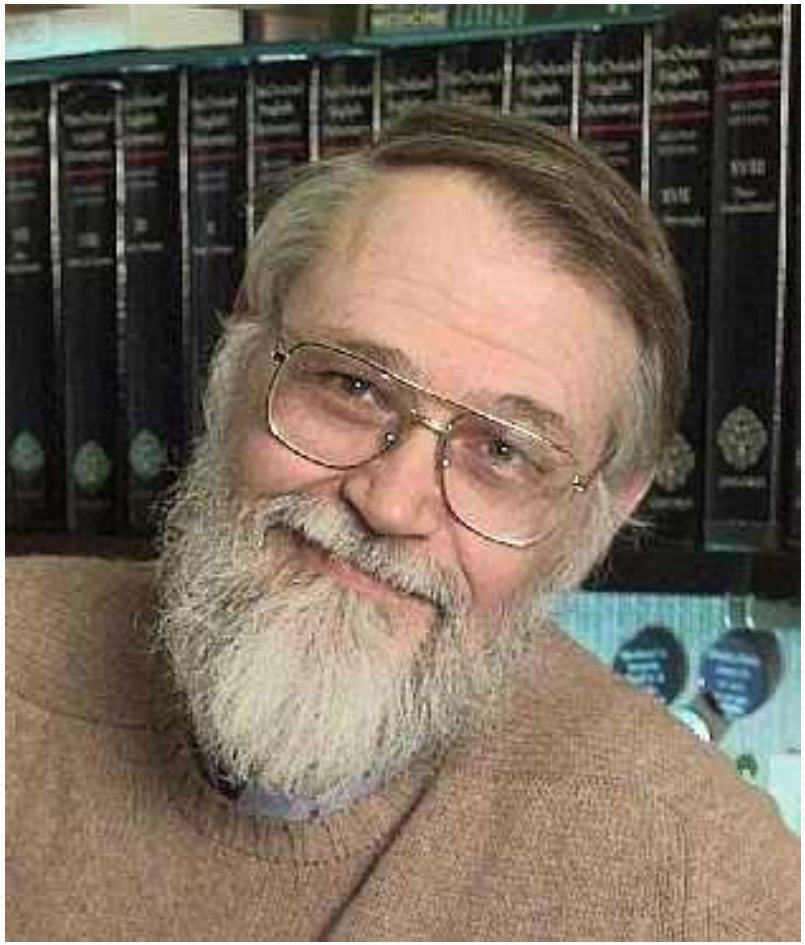
Автор известной книги “Искусство программирования”

На январь 2013 годаНа январь 2013 года Дональд Кнут занимает 37 место в списке самых цитируемых авторов в областикомпьютерных наук согласно проекту CiteSeer

создатель настольных издательских систем TEXсоздатель настольных издательских систем TEX и METAFONT, предназначенных для набора и вёрстки книг, посвящённых технической тематике (в первую очередь — физико-

Классификация технологических подходов

Brian Wilson Kernighan



- Брайан Уилсон Керниган - род. 1942, Торонто, Онтарио, Канада - соавтор знаменитого руководства "Язык программирования Си" совместно с автором языка Деннисом Ритчи. Соавтор языка AWK (Alfred Aho, Peter Weinberger, Brian Kernighan). В соавторстве с Робом Пайком написал также известные книги "Практика программирования" и "UNIX. Программное окружение". Последнюю часто называют своего рода "Библией для UNIX-программистов".

Классификация технологических подходов (1 группа)

Выделим основные группы технологических подходов и укажем подходы для каждой из них.

- **1. Подходы со слабой формализацией**

  Эти подходы не используют явных технологий и их можно применять только для очень маленьких проектов, как правило, завершающихся созданием демонстрационного прототипа. К подходам со слабой формализацией относятся так называемые ранние технологические подходы, например подход "кодирование и исправление".

  - **1.1. Подход "кодирование и исправление"**

Классификация технологических подходов (2 группа)

- **2 Строгие (классические, жесткие, предсказуемые) подходы**

  Данную группу подходов рекомендуется применять для средних, крупномасштабных и гигантских проектов с фиксированным объемом работ. Одно из основных требований к таким проектам - предсказуемость. В эту группу входят подходы, перечисленные ниже.

  - 2.1. Каскадные технологические подходы.
    - 2.1.1. Классический каскадный подход
    - 2.1.2. Каскадно-возвратный подход
    - 2.1.3. Каскадно-итерационный подход
    - 2.1.4. Каскадный подход с перекрывающимися процессами
    - 2.1.5. Каскадный подход с подпроцессами
    - 2.1.6. Спиральная модель
  - 2.2. Каркасные подходы.
    - 2.2.1. Рациональный унифицированный процесс
  - 2.3. Генетические подходы.
    - 2.3.1. Синтезирующее программирование
    - 2.3.2. Сборочное (расширяемое) программирование
    - 2.3.3. Конкретизирующее программирование
  - 2.4. Подходы на основе формальных преобразований.
    - 2.4.1. Технология стерильного цеха
    - 2.4.2. Формальные генетические подходы

Классификация технологических подходов (3 группа)

- **3 Гибкие (адаптивные, легкие) подходы**

Подходы этой группы рекомендуется применять для небольших или средних проектов в случае неясных или изменяющихся требований к системе. Команда разработчиков должна быть ответственной и квалифицированной, а заказчики должны быть согласны принимать участие в разработке.

- 3.1. Ранние технологические подходы быстрой разработки.
  - 3.1.1. Эволюционное прототипирование
  - 3.1.2. Итеративная разработка
  - 3.1.3. Постадийная разработка
- 3.2. Адаптивные подходы.
  - 3.2.1. Экстремальное программирование
  - 3.2.2. Адаптивная разработка
- 3.3. Подходы исследовательского программирования.
  - 3.3.1. Компьютерный дарвинизм

1. Подходы со слабой формализацией

1.1. Подход "кодирование и исправление"

Подход "кодирование-исправление" (code and fix) упрощенно может быть описан следующим образом. Разработчик начинает кодирование системы с самого первого дня, не занимаясь сколь-либо серьезным проектированием. Все ошибки обнаруживаются, как правило, к концу кодирования и требуют исправления через повторное кодирование.

Фактически каждый из программистов, так или иначе, применял этот подход. Практически все учебные программы пишутся в таком стиле. Следует заметить, что при использовании данного подхода затрачивается время лишь на кодирование и заказчику легко демонстрировать прогресс в разработке в строках кода.

Этот подход может быть рекомендован к использованию в двух случаях.

- Для очень маленьких проектов, которые должны завершиться разработкой демонстрационного прототипа.
- Для доказательства некоторой программной концепции.

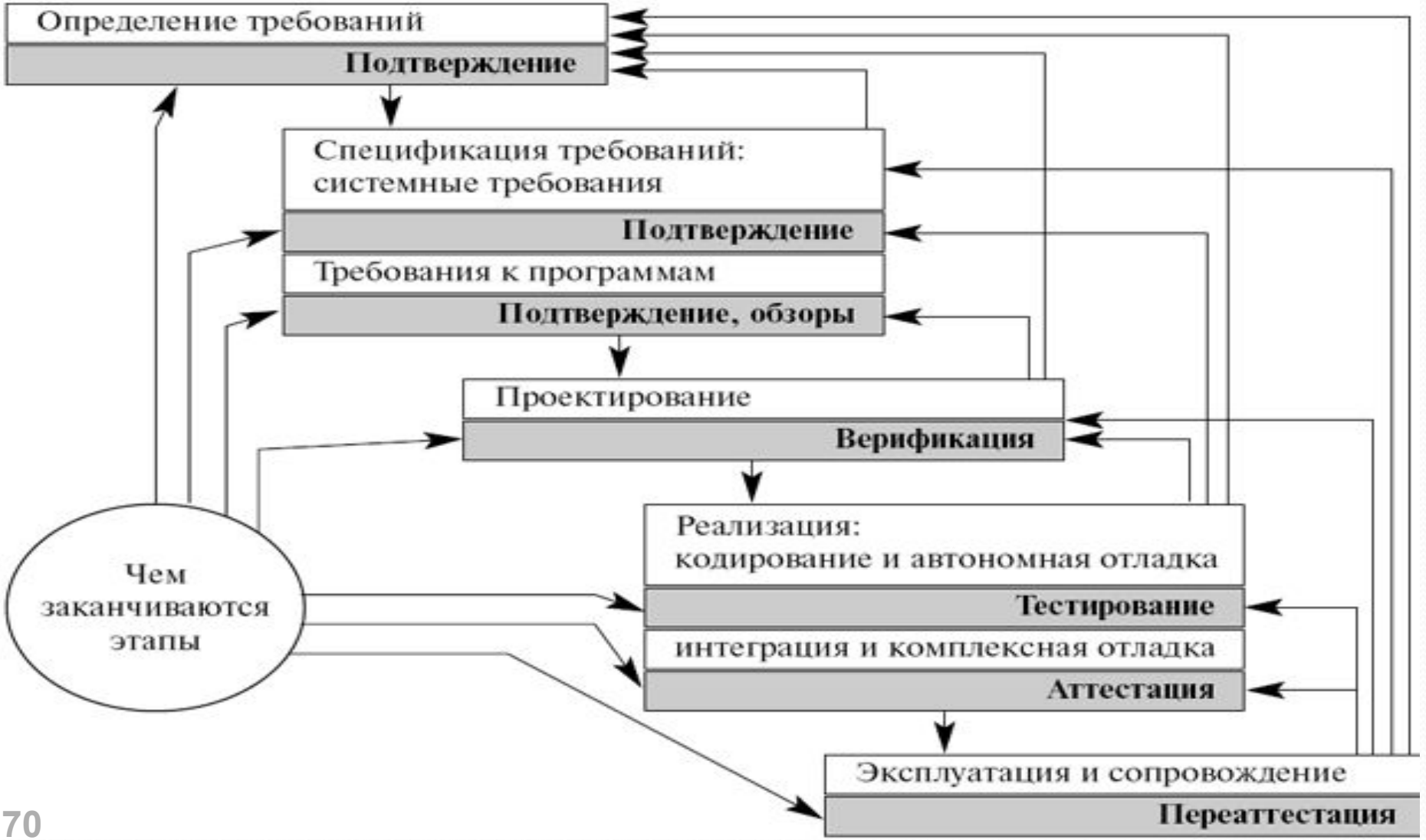
2. Строгие (классические, жесткие, предсказуемые, прогнозирующие, тяжеловесные) подходы

Каскадные технологические подходы (2.1 группа)

Каскадные технологические подходы задают некоторую последовательность выполнения процессов, обычно изображаемую в виде каскада. Эти подходы также иногда называют подходами на основе модели водопада.

Схема общепринятой модели жизненного цикла проекта

Каскадная модель



Верификация и аттестация

- В *каскадной модели* верификация и аттестация приписаны к разным этапам.
- Если рассматривать их как метод проверки проектных результатов, то охарактеризуем их отличие:
- верификация отвечает на вопрос, правильно ли **создана** программная система;
- аттестация отвечает на вопрос, правильно ли **работает** программная система.

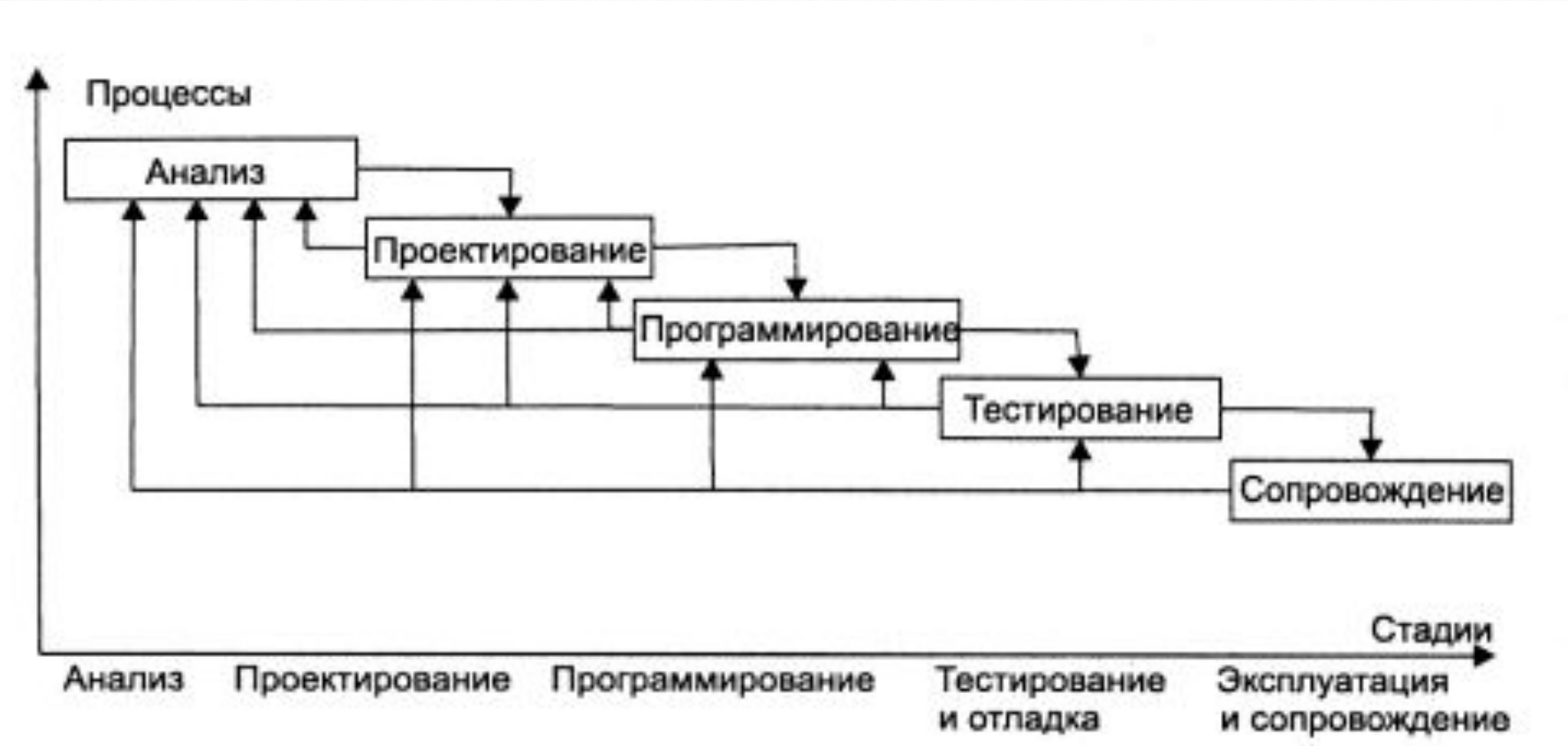
2.1.1. Каскадный подход

Каскадный подход (pure waterfall) считается "дедушкой" технологических подходов к ведению жизненного цикла. Фактически, его можно рассматривать как отправную точку для огромного количества других подходов. Сформировался каскадный подход в период с 1970 по 1985 годы. Специфика "чистого" каскадного подхода такова, что переход к следующему процессу осуществляется только после того, как завершена работа с текущим процессом. Возвраты к уже пройденным процессам не предусмотрены. Данный подход может быть рекомендован к применению в тех проектах, где в самом начале все требования могут быть сформулированы точно и полно. Например, в задачах вычислительного характера. Достаточно легко при таком технологическом подходе вести планирование работ и формирование бюджета.

2.1.2. Каскадно-возвратный подход

Основной недостаток каскадного подхода - отсутствие гибкости. Именно этот недостаток преодолевается каскадно-возвратным подходом, в котором разрешены возвраты к предыдущим стадиям и пересмотр или уточнение ранее принятых решений. Каскадно-возвратный подход отражает итерационный характер разработки программного обеспечения. Этот подход в значительной степени отражает реальный процесс создания программного обеспечения, в том числе и существенное запаздывание с достижением результата. На задержку оказывают существенное влияние корректировки при возвратах.

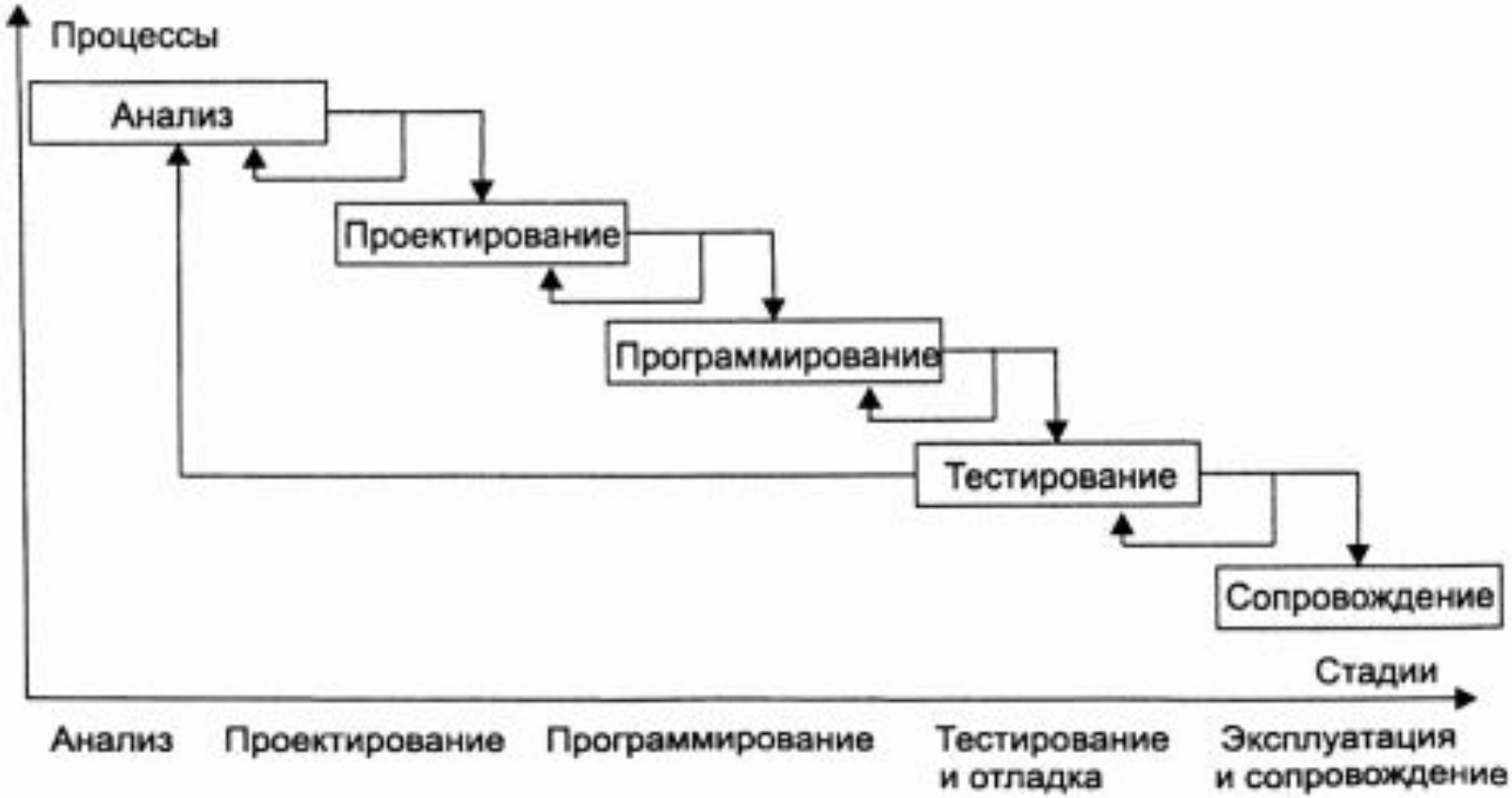
2.1.2. Каскадно-возвратный подход



2.1.3. Каскадно-итерационный подход

- Каскадно-итерационный подход предусматривает последовательные итерации каждого процесса до тех пор, пока не будет достигнут желанный результат. Каждая итерация является завершенным этапом, и ее итогом будет некоторый конкретный результат. Возможно, данный результат будет промежуточным, не реализующим всю ожидаемую функциональность.

2.1.3. Каскадно-итерационный подход



2.1.4.Каскадный подход с перекрывающимися процессами

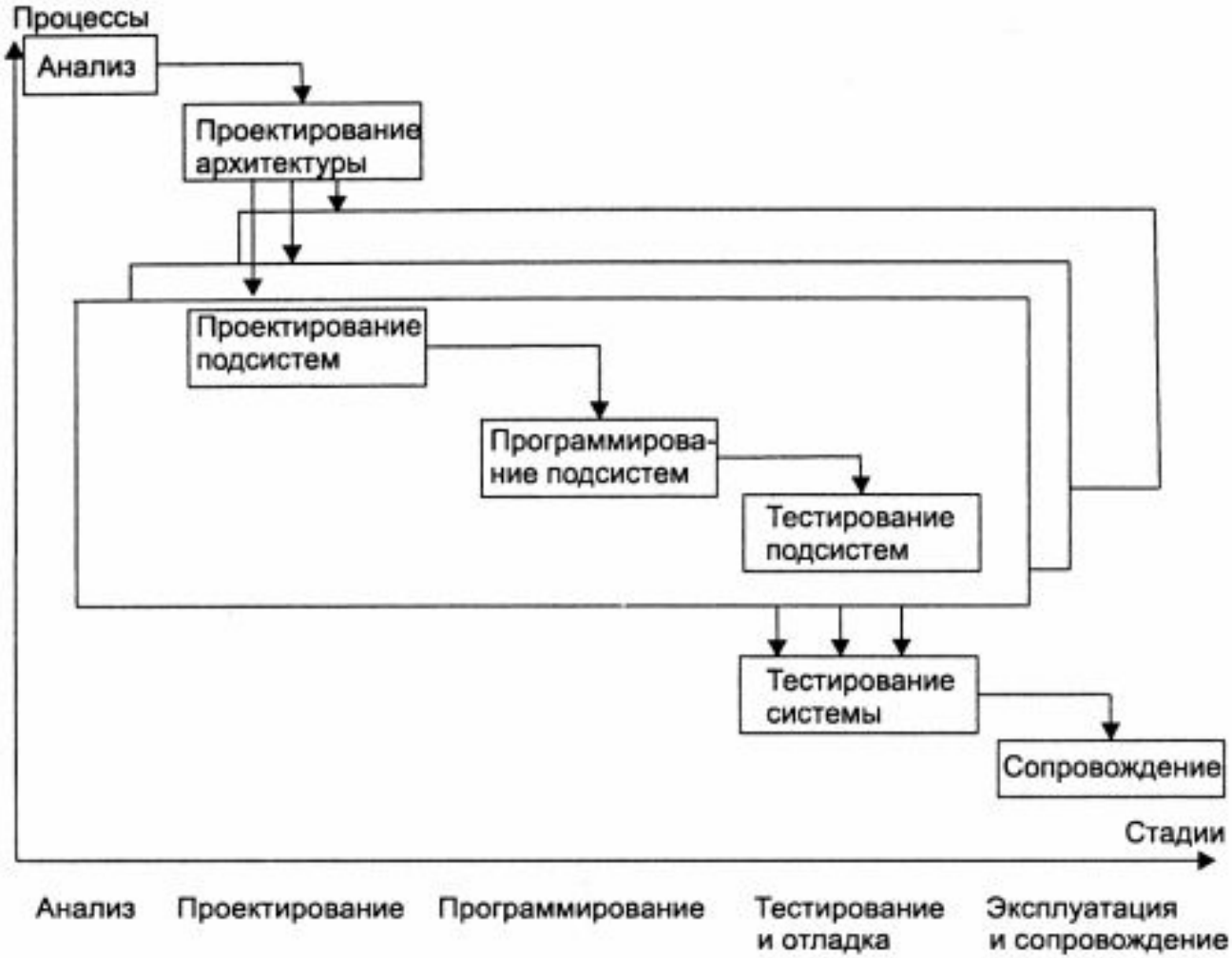
Классический каскадный подход позволяет выполнять каждый процесс отдельной команде. Достаточно разумным является использование команды на том же самом процессе в следующей разработке. Каскадный подход с перекрывающимися процессами (waterfall with overlapping) предполагает наличие таких специализированных команд, позволяющих до определенной степени сократить передаваемую документацию. Следующий процесс начинается до завершения текущего. Более того, несколько процессов могут выполняться параллельно.

2.1.4.Каскадный подход с перекрывающимися процессами

2.1.5. Каскадный подход с подпроцессами

- Каскадный подход с подпроцессами (waterfall with subprocesses) очень близок подходу с перекрывающимися процессами. Особенность его в том, что с архитектурной точки зрения проект достаточно часто может быть разделен на подпроекты, которые могут разрабатываться индивидуально. В данном подходе требуется дополнительная фаза тестирования подсистем до объединения их в единую систему. Следует особое внимание обращать на грамотное деление проекта на подпроекты, которое должно учесть все возможные зависимости между подсистемами.

2.1.5. Каскадный подход с подпроцессами



2.1.6. Спиральная модель

Спиральная модель (spiral model) была предложена Барри Боэмом (Barry Boet) в середине 80-х годов XX века с целью сократить возможный риск разработки. Фактически, это была первая реакция на устаревание каскадной модели. Спиральная модель использует понятие прототипа - программы, реализующей частичную функциональность создаваемого программного продукта. Создание прототипов осуществляется за несколько витков спирали, каждый из которых состоит из "анализа риска", "некоторого процесса" и "верификации". Обращение к каждому процессу предваряет "анализ риска", причем, если риск превышения сроков и стоимости проекта оказывается существенным, то разработка заканчивается. Это позволяет предотвратить более крупные денежные потери в будущем.

2.1.6. Спиральная модель