ВВЕДЕНИЕ В ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ

Технология программирования — способ организации **процесса** создания программы.

— Почему вы пилите тупой пилой, ведь это очень долго и трудно?
— Некогда точить, пилить надо!!!

Литература

1. С. Макконнелл. Совершенный код. — СПб: «Питер», 2005. — 896 с.
2. К. Бек. Экстремальное программирование. — СПб: «Питер», 2002.
3. К. Ауэр, Р. Миллер. Экстремальное программирование. — СПб: «Питер», 2003. — 368 с.
4. А. Якобсон, Г. Буч, Д. Рамбо. Унифицированный процесс разработки программного обеспечения. — СПб: «Питер», 2002. — 496 с.
5. Э. Брауде. Технология разработки программного обеспечения. — СПб: «Питер», 2004. — 655 с.
6. С. Орлов. Технологии разработки программного обеспечения. — СПб: «Питер», 2003. — 480 с.
7. Д. Бентли. Жемчужины программирования. — СПб: «Питер», 2002. — 272 с.
8. Р. Мартин. Чистый код: создание, анализ и рефакторинг. — СПб: «Питер», 2010. — 464 с.
9. http://www.agiledev.ru/

Основные критерии качества ПО

- **Качество** – степень соответствия требованиям (потребностям и ожиданиям пользователя).

- надежность
- возможность точно планировать производство и сопровождение

Критерии качества ПО

Внешние характеристики

- корректность
  - наличие/отсутствие дефектов в спецификации проекте и реализации
- практичность
  - легкость изучения и использования
- эффективность
  - степень использования системных ресурсов
- надежность
  - способность системы выполнять необходимые функции; интервал между отказами
- целостность
  - способность предотвращать неавторизованный или некорректный доступ
- адаптируемость
  - возможность использования в других областях и средах
- правильность
  - степень безошибочности данных, выдаваемых системой
- живучесть
  - способность продолжать работу при недопустимых данных или в напряженных условиях

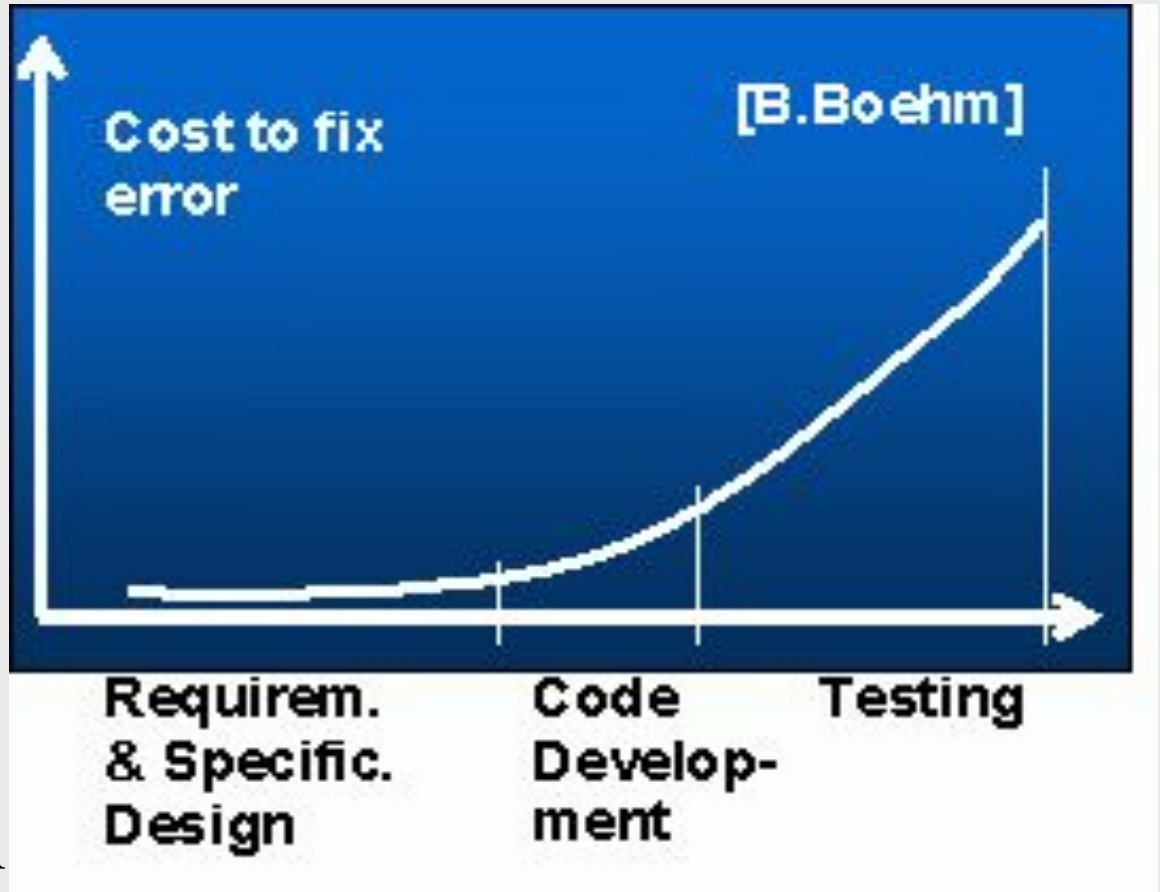
Внутренние характеристики

- удобство сопровождения
- тестируемость
- удобочитаемость
- гибкость
- портируемость
- возможность повторного использования
- понятность

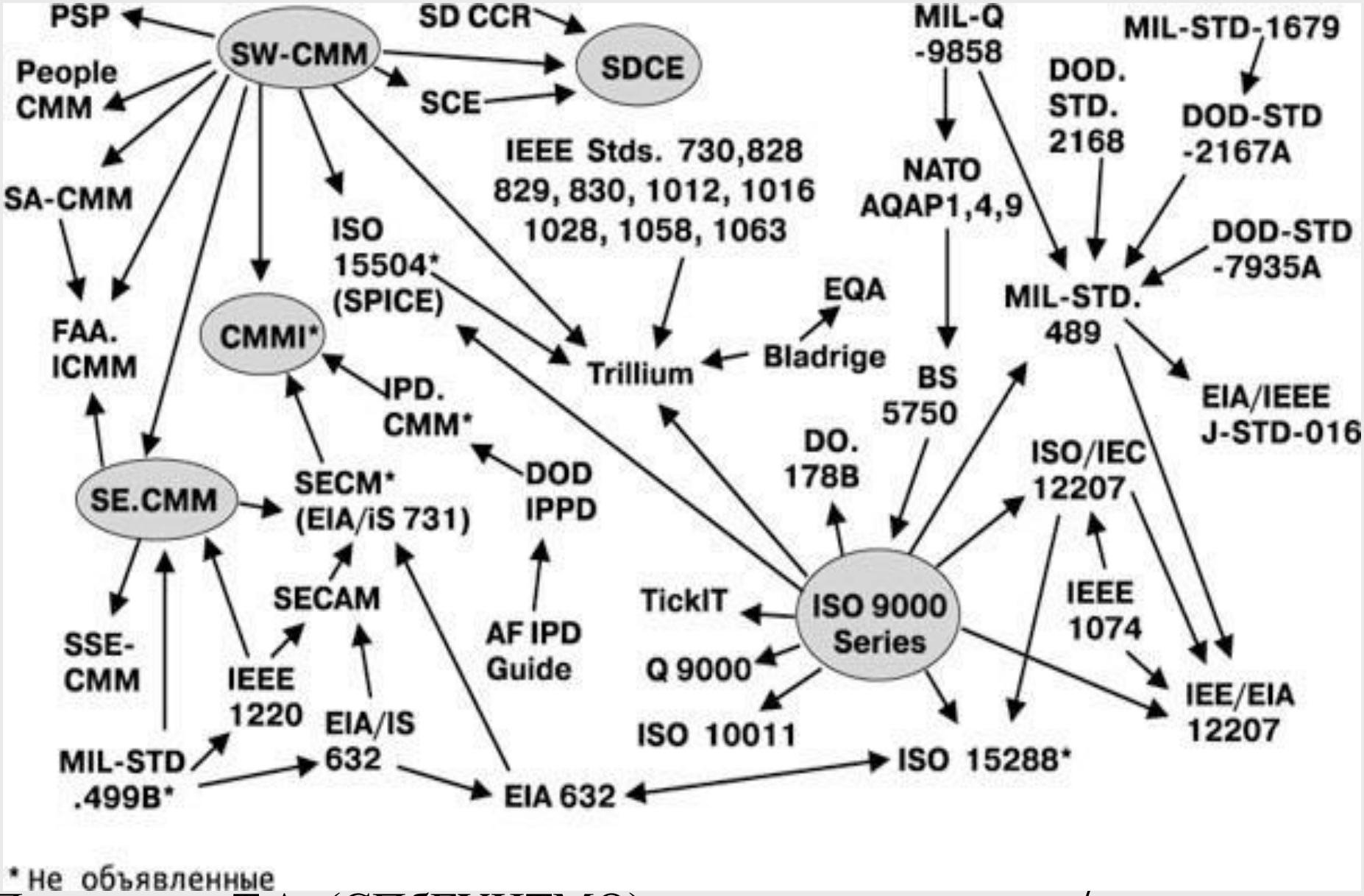
	корректность	практичность	эффективность	надежность	Целостность	адаптируемость	правильность	живучесть
корректность	↑		↑	↑			↑	↓
практичность		↑				↑	↑	
эффективность	↓		↑	↓	↓	↓	↓	
надежность	↑			↑	↑		↑	↓
целостность			↓	↑	↑			
адаптируемость					↓	↑		↑
правильность	↑		↓	↑		↓	↑	↓
живучесть	↓	↑	↓	↓	↓	↑	↓	↑

Главный закон контроля качества ПО

Повышение качества системы снижает расходы на ее разработку



Взаимосвязь наиболее признанных и применяемых в мире стандартов в области разработки программного обеспечения



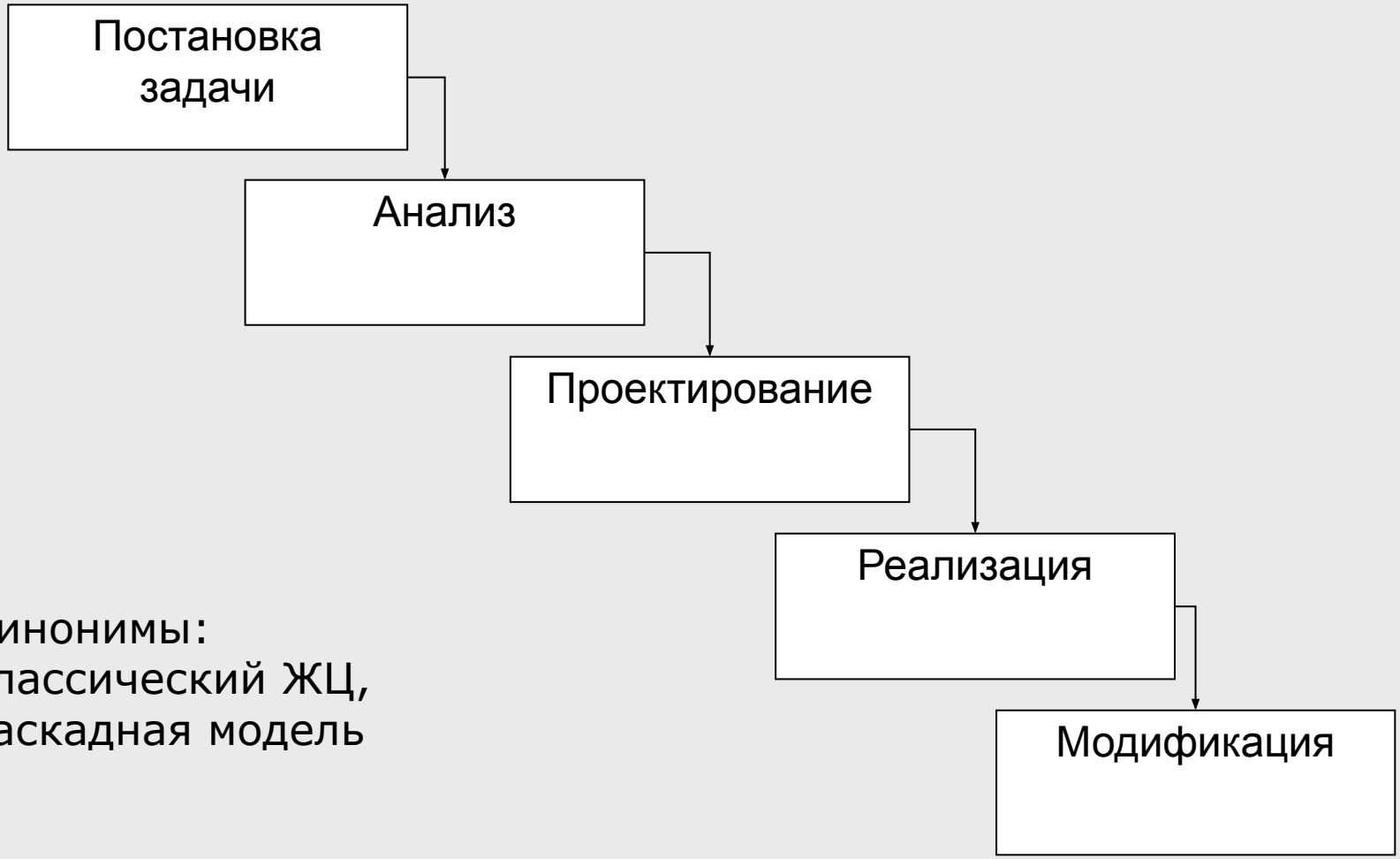
Модели жизненного цикла ПО

- Каскадная (водопадная, нисходящая)
- Макетирование (прототипирование)
- Инкрементная
- Спиральная (итерационная)

Стратегии создания ПО

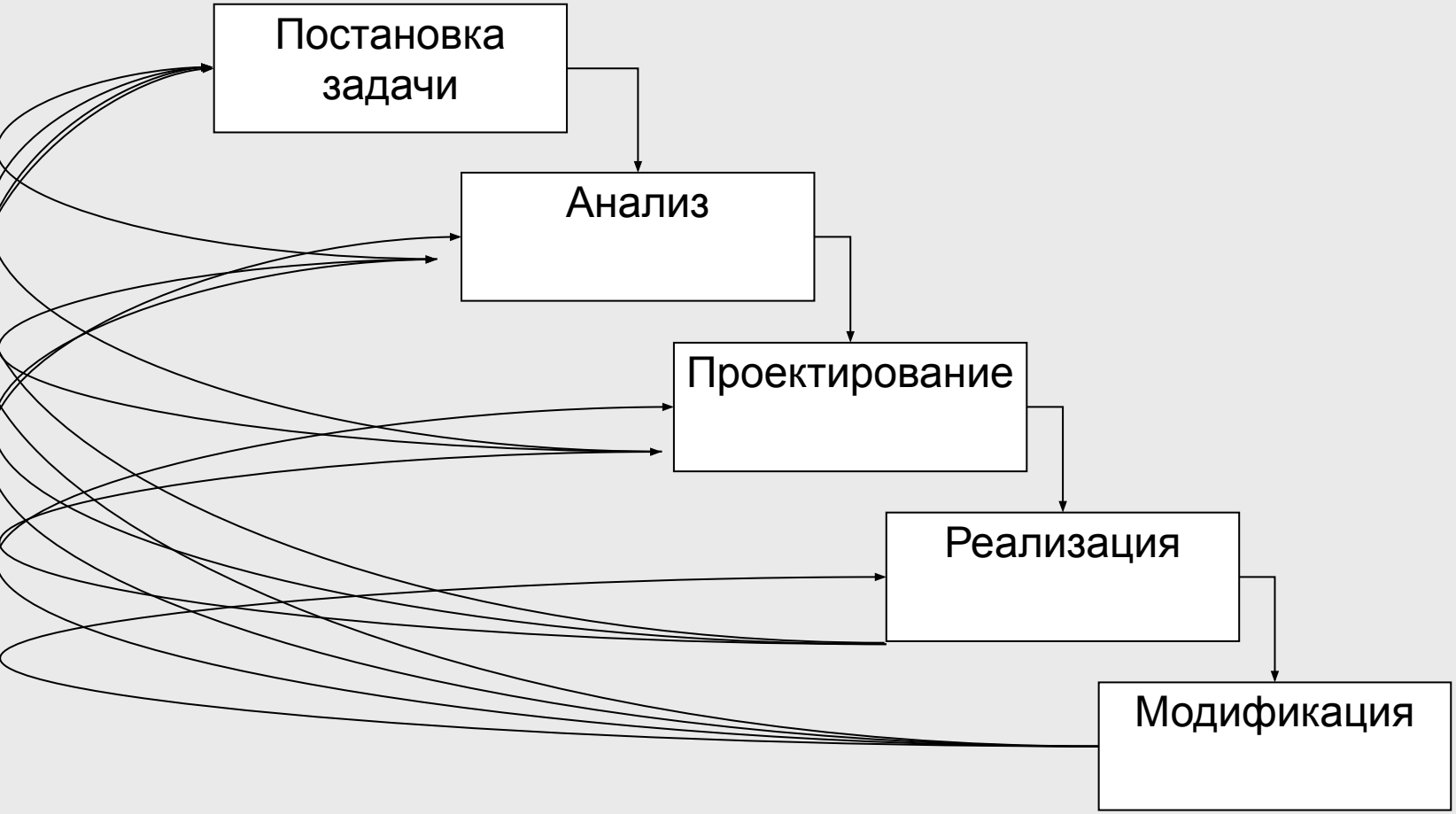
	Водопад-ная	*Итеративные*	
		Инкремен-тная	***Эволюци-онная***
В начале определены все требования?	+	+	-
Циклов конструирования	1	>1	>1
Промежуточное ПО распространяется?	-	**±**	+

Водопадная модель жизненного цикла ПО:



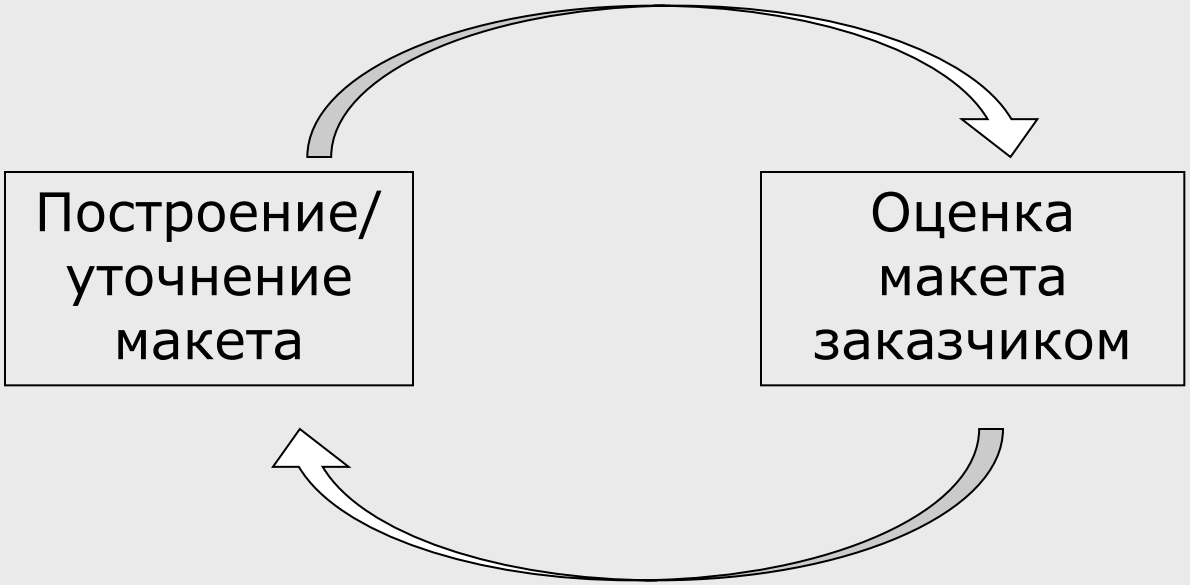
Синонимы:
классический ЖЦ,
каскадная модель

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.

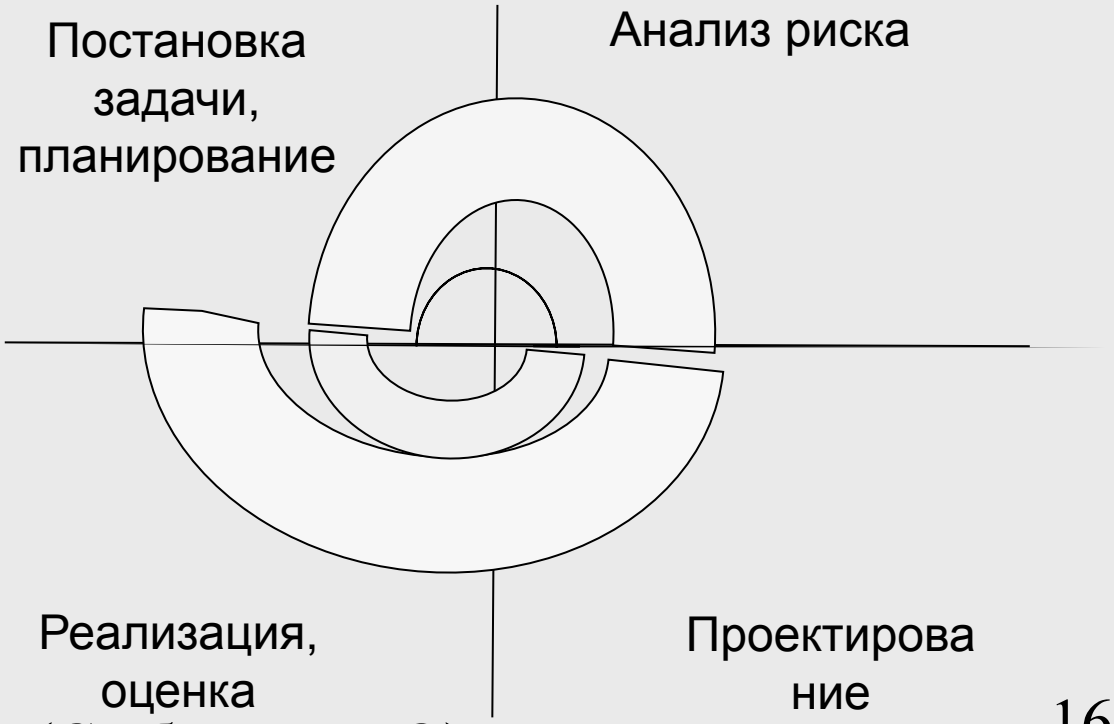
- RAD хорошо зарекомендовал себя для относительно небольших стандартных проектов, разрабатываемых для конкретного заказчика.
- RAD ориентирован на разработку информационных систем, которые можно разбить на отдельные модули и где производительность не критична.

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 называют действующий программный продукт, реализующий отдельные функции и внешние интерфейсы разрабатываемого программного обеспечения.



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.

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

Распространенные технологии

- RUP
- Гибкие технологии (Agile):
  - Экстремальное программирование (XP)
  - Разработка через тестирование (TDD)
  - …

Выбор технологии зависит от сложности программного обеспечения, наличия готовых компонентов, имеющихся ресурсов и т. д.

Экстремальное программирование

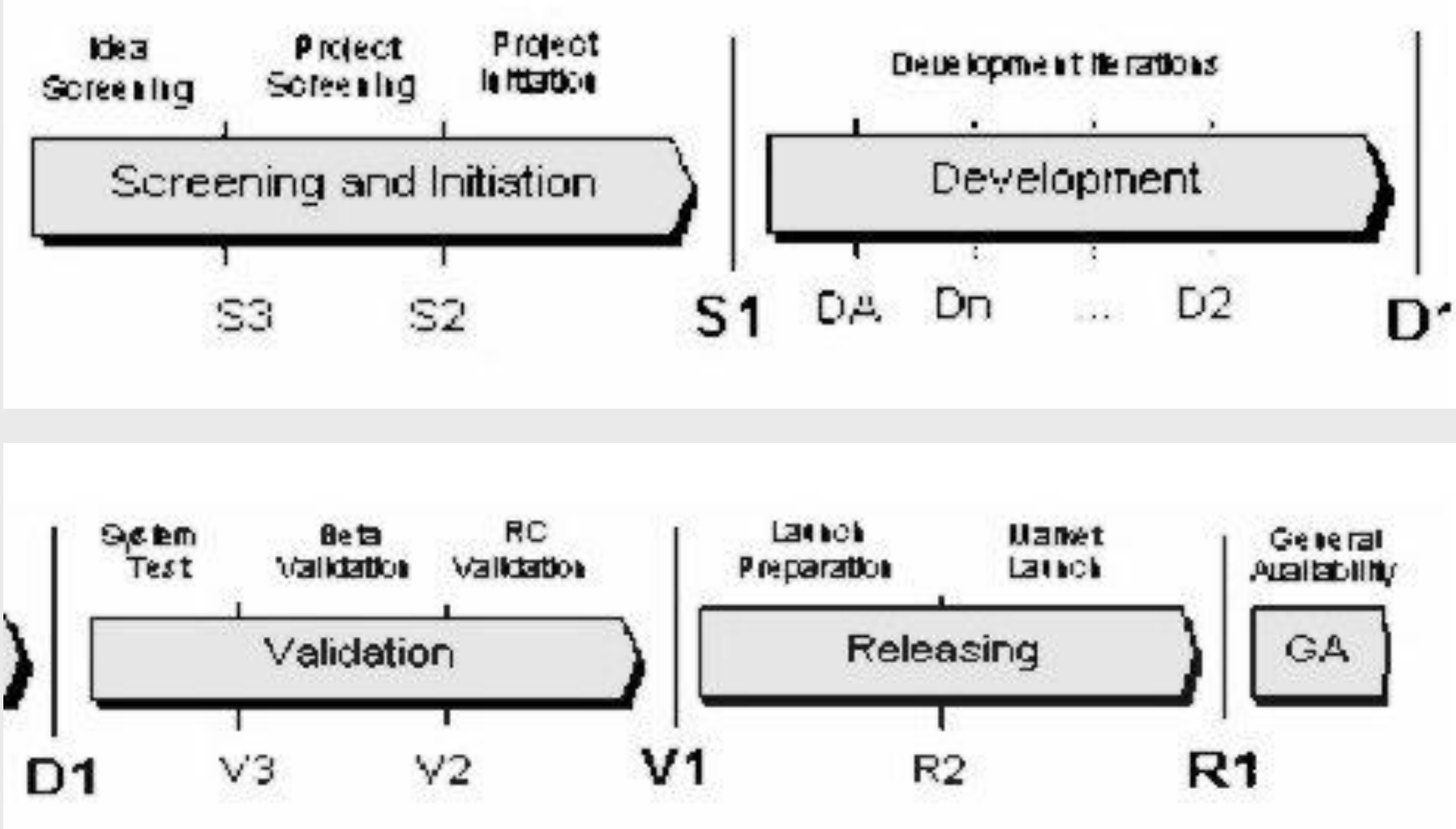
- Основная идея экстремального программирования (XP) — устранить высокую стоимость изменений, вносимых в ПО в процессе как разработки, так и эксплуатации.
- Цикл разработки в XP состоит из очень коротких итераций. Четырьмя базовыми действиями в цикле являются:
  - выслушивание заказчика
  - проектирование
  - 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.

Основные принципы XP

1. Планирование
2. Частая смена версий
3. Метафора
4. Простой проект
5. Тесты
6. Переработка системы
7. Программирование в паре
8. Непрерывная интеграция
9. Коллективное владение
10. Заказчик с постоянным у частием
11. 40-часовая неделя
12. Открытое рабочее пространство
13. Стандарты кодирования
14. Не более чем правила

Область применимости XP: небольшие и средние проекты.

Пример реального процесса разработки ПО



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.

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

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 (use case)
- 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 итеративного процесса

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

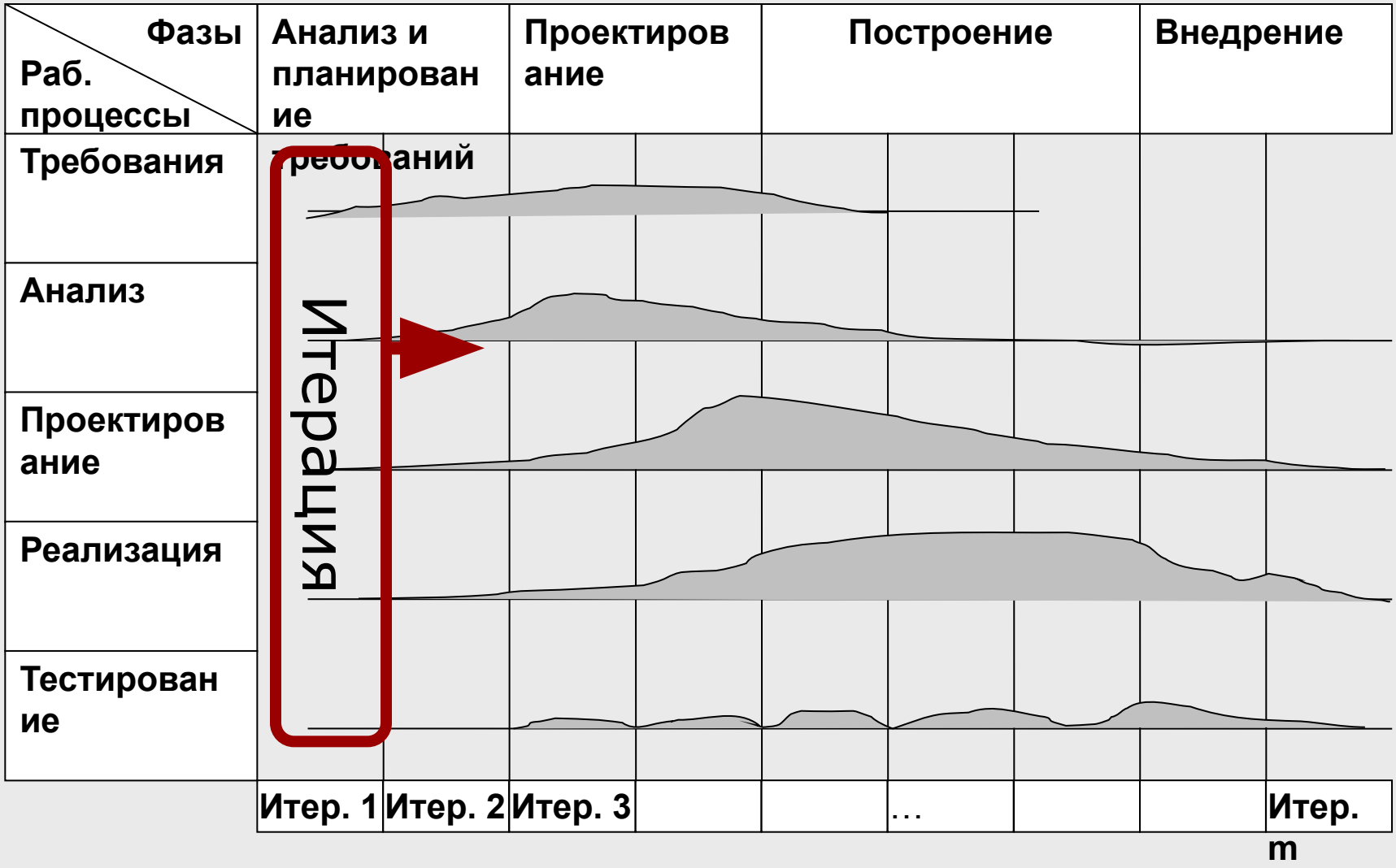
Жизненный цикл УП

- 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

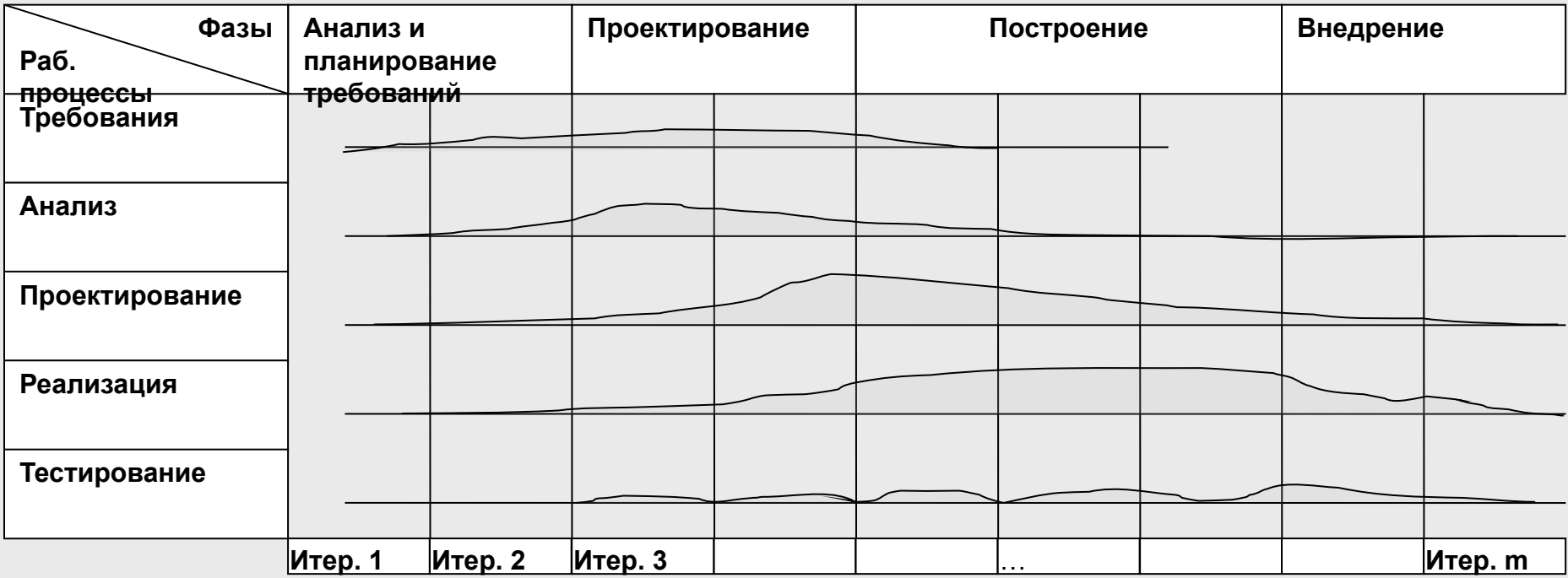
Анализ и планирование требований:

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

Проектирование:

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

Анализ и планирование требований



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

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
- …

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.

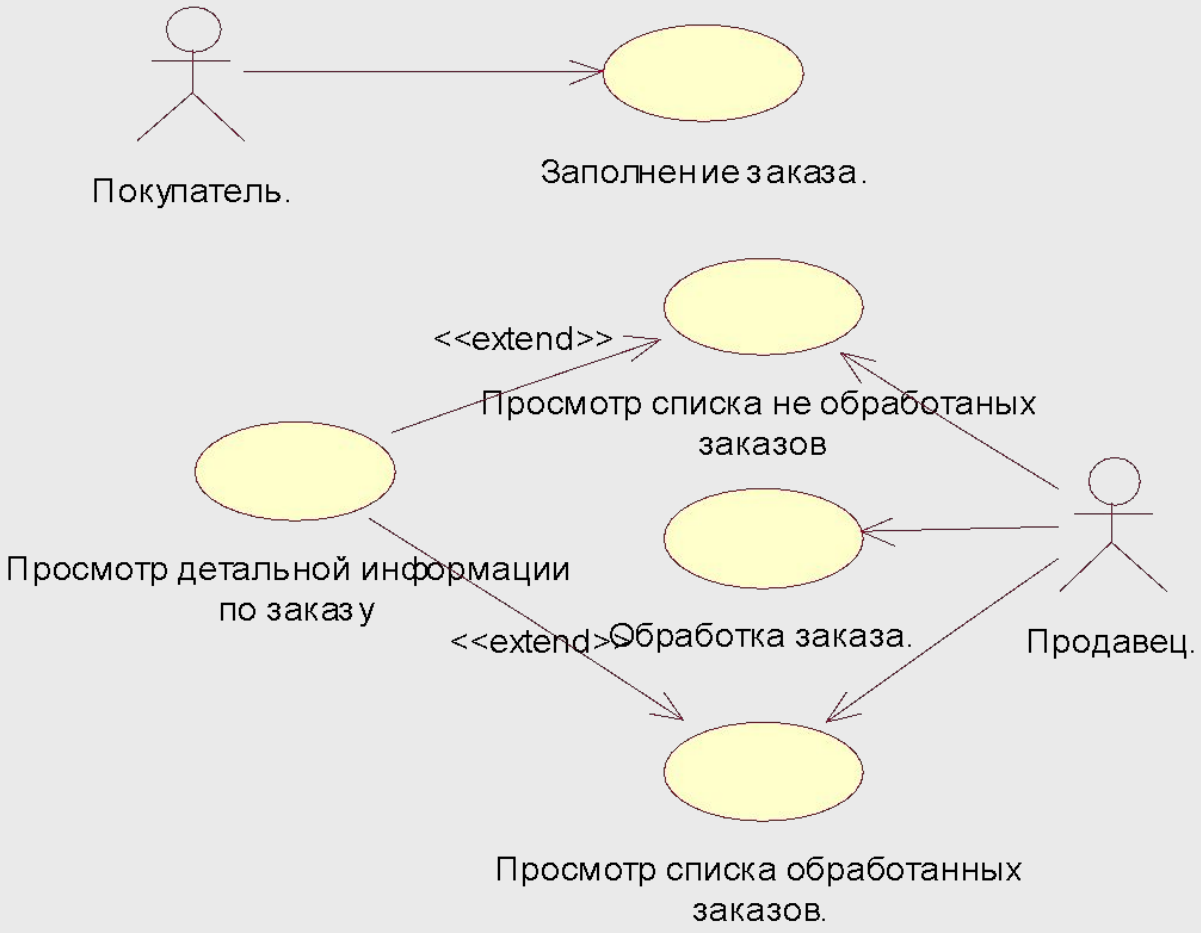
UML

UML - Unified Model Language - является языком для специфицирования, визуализации, конструирования и документирования программных продуктов, а также используется в бизнес-моделировании и моделировании любых иных (не программных) систем.

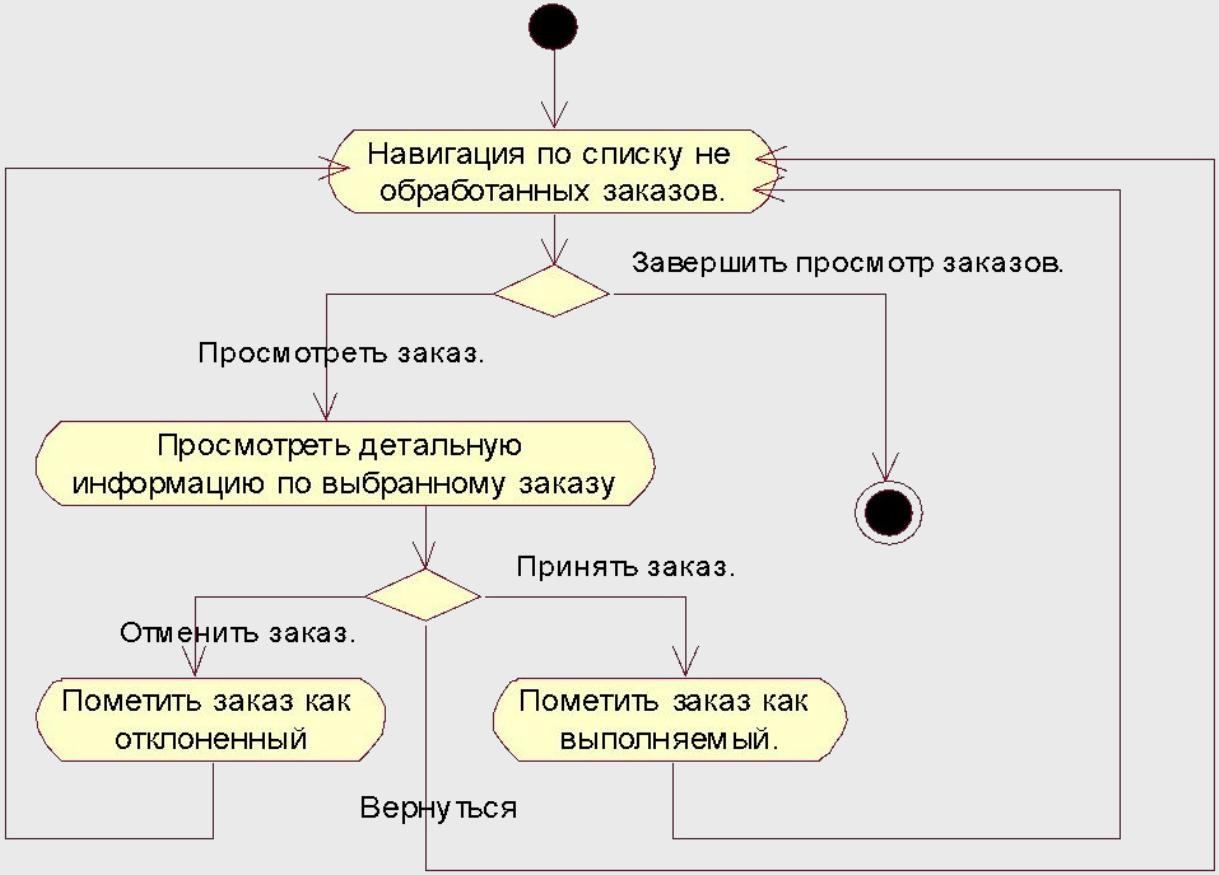
UML позволяет задавать следующие аспекты:

- 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)

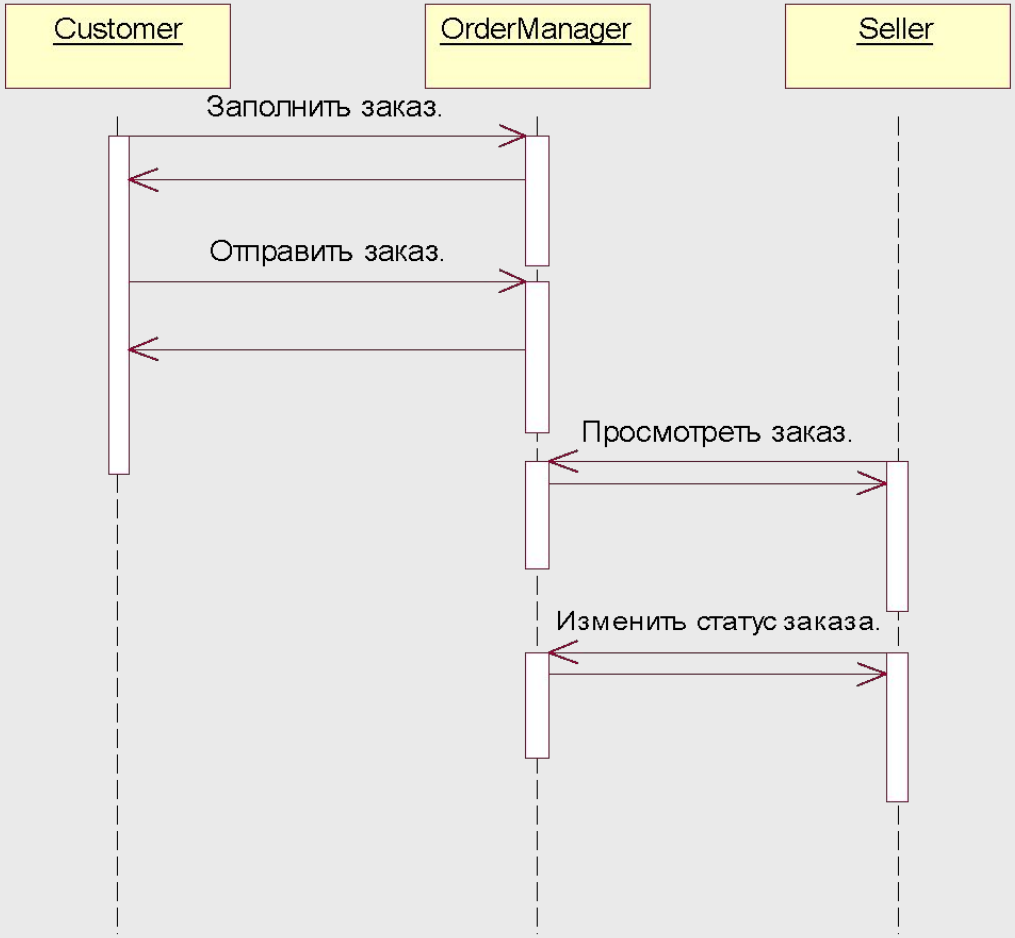
Диаграммы вариантов использования (Use case diagrams)



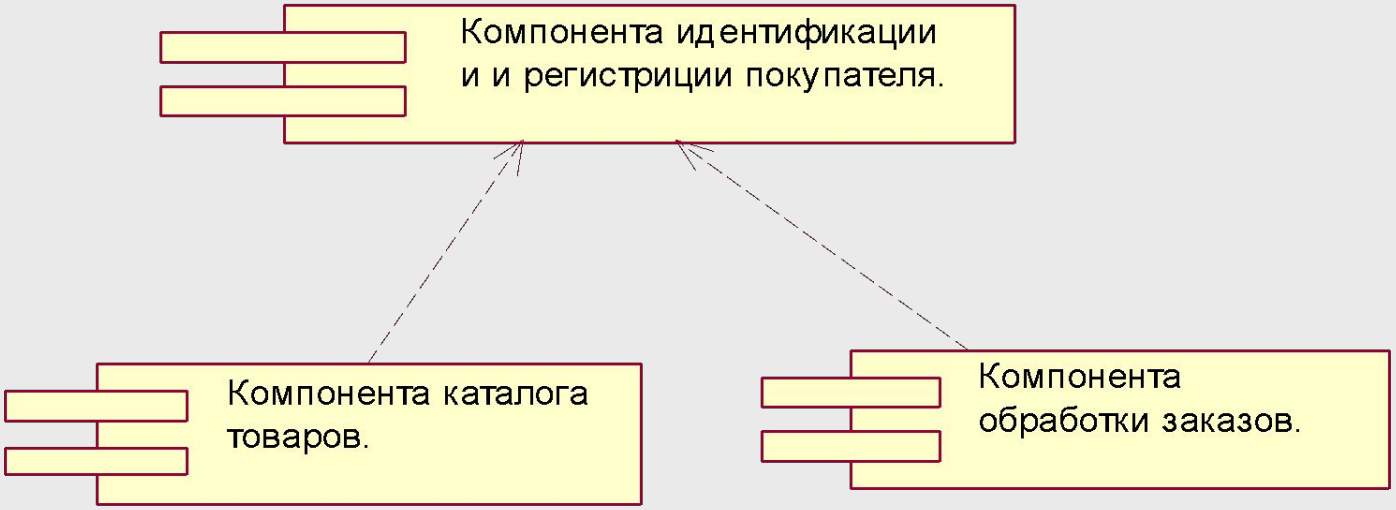
Диаграммы деятельности (Activity diagrams)



Диаграммы последовательностей действий (Sequence diagrams)



Диаграммы компонент (Component diagrams)



Чем чаще программист жалуется на чужой soft, тем хуже он делает свой.

- Более подробно – самостоятельно!

Чем чаще программист жалуется на чужой soft, тем хуже он делает свой.

Capability Maturity Model

Структурное программирование

- В основе структурного подхода лежит идея *декомпозиции*, то есть разбиения системы на части, реализующие отдельные подзадачи. Каждая подзадача, в свою очередь, дробится на более мелкие. Этот процесс называется *нисходящим проектированием*.

- При написании структурной программы применяется фиксированный набор конструкций, называемых *базовыми*, что позволяет уменьшить как количество ошибок в программе, так и их цену.

- Под *ценой ошибки* понимается стоимость ее исправления: она тем выше, чем позже в процессе разработки обнаружена ошибка.

Этапы создания структурной программы

I. Постановка задачи

Изначально задача формулируется в терминах предметной области, и необходимо перевести ее на язык понятий, более близких к программированию.

На этом этапе также определяется среда, в которой будет выполняться программа.

Постановка задачи завершается созданием технического задания, а затем внешней спецификации программы, включающей в себя:

- описание *исходных данных и результатов* (типы, форматы, точность, способ передачи, ограничения);
- описание *задачи*, реализуемой программой;
- способ обращения к программе;
- описание возможных *аварийных ситуаций и ошибок* пользователя.

II. Выбор модели и метода решения задачи

Постановка задачи формализуется и на этой основе определяется общий метод ее решения. При наличии нескольких методов выбирается наилучший, исходя из критериев сложности, эффективности, точности и т. д.

III. Разработка внутренних структур данных

При решении вопроса о том, как будут организованы данные в программе, полезно задать себе следующие вопросы.

- Какая точность представления данных необходима?
- В каком диапазоне лежат значения данных?
- Ограничено ли максимальное количество данных?
- Обязательно ли хранить их в программе одновременно?
- Какие действия потребуется выполнять над данными?

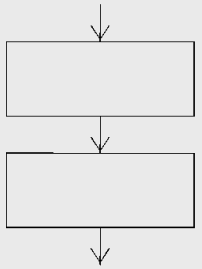
IV. Проектирование

Под проектированием программы понимается определение ее общей структуры и взаимодействия модулей. На этом этапе применяется *технология нисходящего проектирования*.

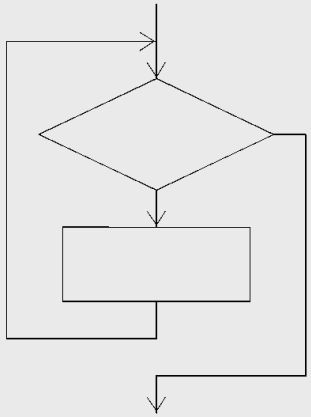
- Для каждой подзадачи составляется *внешняя спецификация*, аналогичная приведенной выше.
- На этом же этапе решаются вопросы *разбиения программы на модули*, главным критерием при этом является минимизация их взаимодействия.
- Очень важной является *спецификация интерфейсов*
- Процесс проектирования является итерационным.
- Алгоритмы записывают в обобщенной форме.

V. Структурное программирование

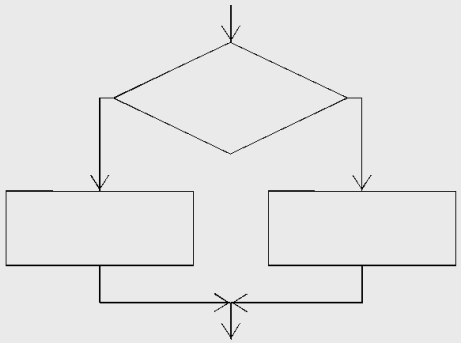
- Кодирование также организуется по принципу «сверху вниз»: вначале кодируются модули самого верхнего уровня и составляются тестовые примеры для их отладки, при этом на месте еще не написанных модулей следующего уровня ставятся так называемые *заглушки*.
- При кодировании применяются только три структуры, называемые *базовыми конструкциями* структурного программирования — *следование, ветвление* и *цикл*.



Следование



Цикл



Ветвление

Правила кодирования

- Программа должна состоять из максимально обособленных частей, связанных друг с другом только через интерфейсы.
- Каждое законченное действие оформляется в виде подпрограммы.
- Все величины, которыми подпрограмма обменивается с вызывающей программой, должны передаваться ей через параметры.
- В подпрограмме полезно предусматривать реакцию на неверные входные параметры и аварийное завершение.
- Величины, используемые только в подпрограмме, следует описывать внутри нее как локальные переменные.
- Имена переменных должны отражать их смысл.
- Следует избегать использования в программе чисел в явном виде.

- Для записи каждого фрагмента алгоритма необходимо использовать наиболее подходящие средства языка.
- Программа должна быть «прозрачна».
- Вложенные блоки должны иметь отступ в 3–4 символа
- Помечайте конец длинного составного оператора
- Более короткую ветвь if лучше поместить сверху
- Сообщение об ошибке должно быть информативным
- Необходимо предусматривать печать сообщений в тех точках программы, куда управление при нормальной работе программы передаваться не должно.

Документирование программы

- Сопровождение программы занимает гораздо больший промежуток времени, чем ее написание, поэтому документирование программы имеет большое значение.
- Основная часть документации должна находиться в тексте программы.
- В идеале комментарии должны создаваться до написания программы — ими служит подробный алгоритм, изложенный на естественном языке.
- Комментарии должны представлять собой правильные предложения без сокращений
- Абзацный отступ комментария должен соответствовать отступу комментируемого блока.
- Программу после написания необходимо тщательно отредактировать

VI. Нисходящее тестирование

- *Тестирование* — процесс, посредством которого проверяется правильность программы. Тестирование носит позитивный характер, его цель — показать, что программа работает правильно и удовлетворяет всем проектным спецификациям.
- *Отладка* — процесс исправления ошибок в программе, при котором цель исправить все ошибки не ставится. Исправляют ошибки, обнаруженные при тестировании.
- Существует две стратегии тестирования: «черный ящик» и «белый ящик». При использовании первой внутренняя структура программы во внимание не принимается и тесты составляются так, чтобы полностью проверить функционирование программы на корректных и некорректных входных воздействиях.
- Стратегия «белого ящика» предполагает проверку всех ветвей алгоритма. Общее число ветвей определяется комбинацией всех альтернатив на каждом этапе.

Этапы тестирования

- Базовый тест
  (простой тестовый пример)
- Инвентаризация
  (определить различные категории данных и создать тесты для каждого элемента категории)
- Комбинированные тесты
  (скомбинировать различные входные данные)
- Граничные оценки
  (оценить поведение программы при граничных значениях данных)
- Ошибочные данные
  (оценить отклик системы на ввод неправильных данных)
- Нагрузочные тесты, создание напряжений
  (попытаться вывести систему из строя)

Средства автоматизированного тестирования

1. инструменты функционального тестирования
   HP (QuickTest Professional, WinRunner), IBM (Robot, Functional Tester), Borland (SilkTest) и AutomatedQA (TestComplete)
2. инструменты нагрузочного тестирования
   HP (LoadRunner), IBM (Robot, Performance Tester)
3. средства поддержки процесса тестирования

Популярные средства:

JUnit - — библиотека для тестирования программного обеспечения на языке Java.

Среда разработки с развитыми средствами тестирования: Ruby on Rails.

Программистские приметы ☺

- Если новая программа с первого раза компилируется без ошибок, значит, она написана принципиально неправильно.
- Если к вам перестали поступать жалобы на вашу программу, значит, ею уже никто не пользуется.
- Чем универсальнее программа, тем меньше мест, где можно ее применить.
- Чем точнее программист выполняет требования заказчика, тем бестолковее получается программа.
- Чем больше заказчик понимает в программировании, тем больше он мешает работе программистов.
- Ошибки легче всего делаются и труднее всего обнаруживаются в самых простых местах программы.
- Нет более живучих программ, чем заплатки, сделанные на скорую руку.
- **Чем чаще программист жалуется на чужой soft, тем хуже он делает свой.**