1.ISO/IEC 12207:2010: Жизненный цикл ПО. Группы процессов ЖЦ.

ЖЦ ПО - это время от идеи до вывода из эксплуатации. Всего по этому исо процессов 43 (7 связаны с разработкой)

Основные этапы:

- Разработка требований → - Анализ
- Проектирование → - Разработка
- Тестирование → - Внедрение
- Эксплуатация → - Вывод из эксплуатации

Процессы:

- Согласования (2)
- Орг. обеспечения (5)
- Проектов (7)
- Тех. процессов (11)
- Реализации ПС (7)
- Поддержки ПС (8)
- Повт. исп. ПС (3)

2.Модели ЖЦ (последовательная, инкрементная, эволюционная).

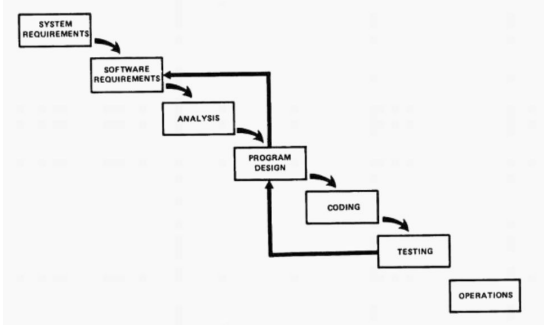
Модель ЖЦ ПО - это структура, определяющая последовательность выполнения и взаимосвязи процессов, действий и задач на протяжении всего ЖЦ.

- последовательная (водопадная, каскадная)
  - стадии выполняются последовательно и один раз
  - требования определены в самом начале
  - поздно получаем инфу о изменениях требований заказчика
- инкрементная
  - продукт разбивается по функциональным требованиям на (одинаковые с арх. т.з.) части
- эволюционная
  - создается прототип, который со временем развивается

- модель формальных преобразований
  - В ней создаются модели ПО, которые последовательно преобразуются друг в друга, а затем в программный код по определенным формальным принципам.

3.Водопадная (каскадная) модель.

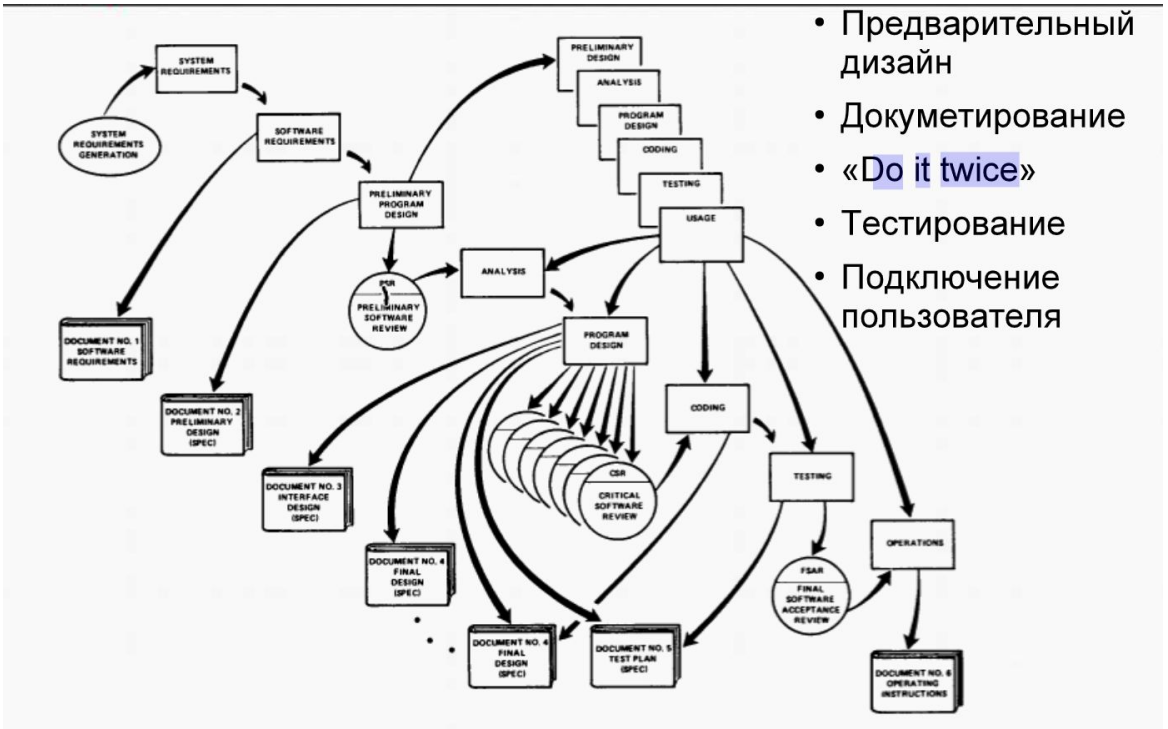
- первая модель
  - состоит из анализа и кодирования
- вторая модель
  - есть понятие итерации
  - есть возможность отката
  - состоит из:
    1. Системные требования.
    2. Требования к ПО.
    3. Анализ требований.
    4. Проектирование программы.
    5. Кодирование.
    6. Тестирование.
    7. Ввод в эксплуатацию.



4.Методология Ройса.

Был развит из водопадной модели.

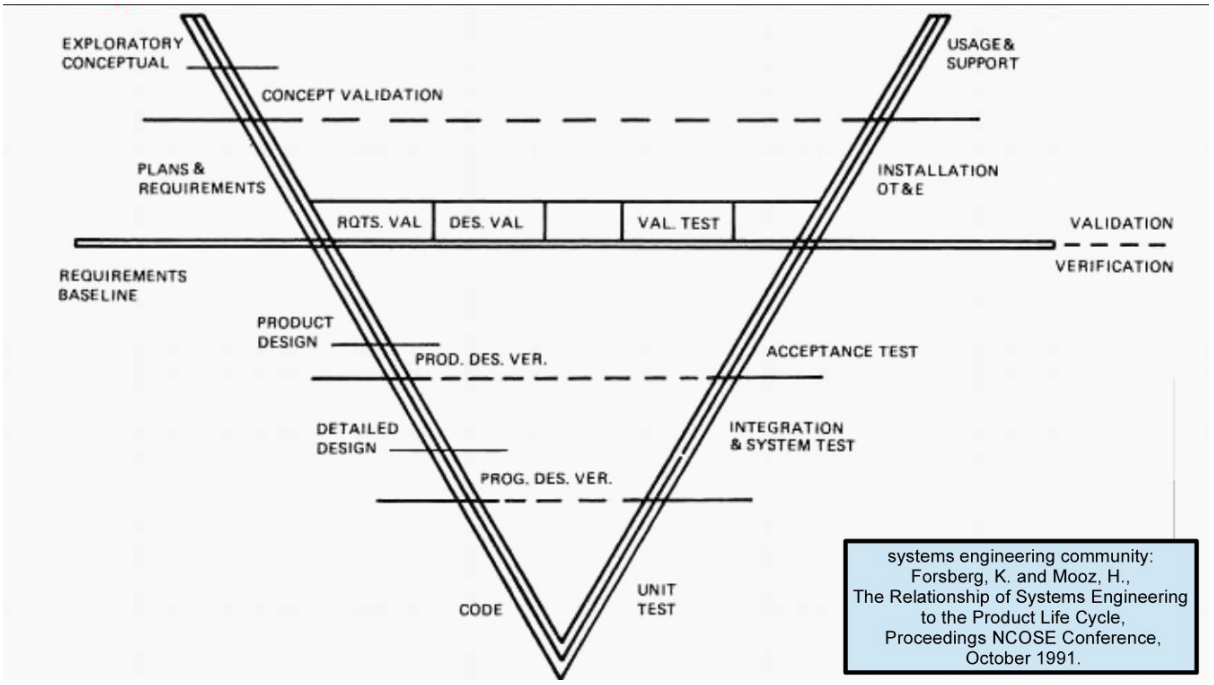
- Предварительный дизайн
  - между требованиями к ПО и анализом
  - спроектировать, определить и создать модели обработки данных
- Документирование
  - требования к системе
  - спецификация предварительного дизайна
  - спецификация дизайна интерфейсов
  - финальные спецификации дизайна системы
  - план тестирования
  - инструкция по использованию
- Do it twice
  - тестовая разработка параллельно основному процессу
- Тестирование
  - планирование, контроль и мониторинг тестирования
  - при планировании исключить дизайнера системы
  - визуальный осмотр -> исправление простых ошибок -> тестирование в окр. ср.
- Подключение пользователей
  - на ранних этапах
  - три точки, где необходим опыт, оценка и подтверждение пользователем:
    1. предварительный просмотр (PSR)
    2. критический просмотр (CSR)
    3. финальный просмотр (FSAR)



5.Традиционная V-chart model J.Munson, B.Boehm.

Те же шаги что и в водопадной модели.

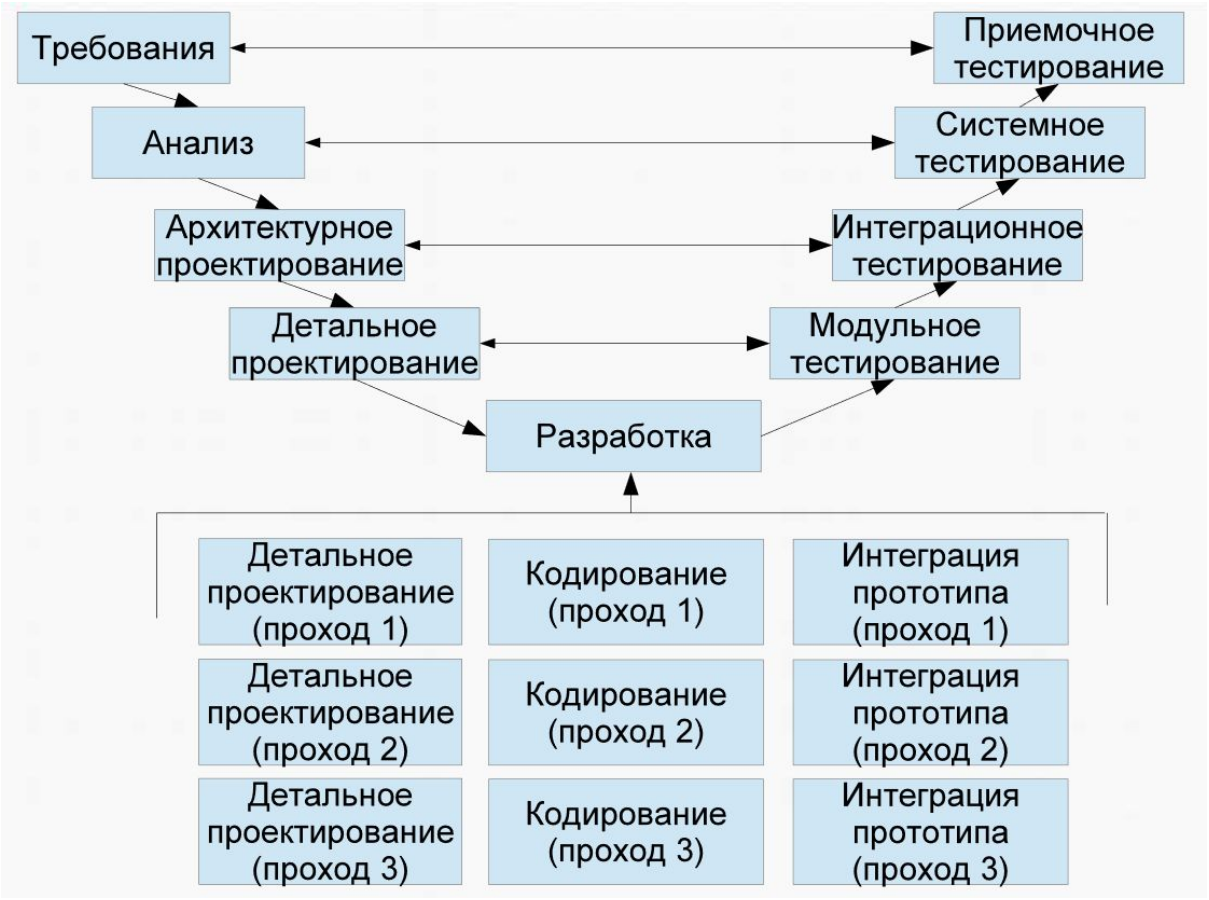
- Каждому уровню разработки соответствует свой уровень тестирования:
    - кодирование - модульное тестирование (на основе критериев верификации)
    - дизайн - интеграционное и системное тестирования (на основе критериев вер.)
    - функц. требования - приемочное тестирование
- Правая часть V, динамическое тестирование, включает программное исполнение тестов
- В левой части могут быть статические тесты (проверка спецификаций, дизайна и т.д.)



6.Многопроходная модель (Incremental model).

Продукт разбивается по отдельным функциональным и техническим требованиям, а впоследствии проектировать, реализовывать и интегрировать воедино в несколько проходов разработки В виде отдельных сборок, которые стали называть инкрементами функционала или просто инкрементами.

- Снижается стоимость изменений требований заказчик
- Можно использовать частично разработанную систему
- Сложность при устаревании архитектуры, требующей рефакторинка
- Сложность ведения актуальной документации
- Сложность предсказания необходимых человеко-часов



7.Модель прототипирования (80-е).

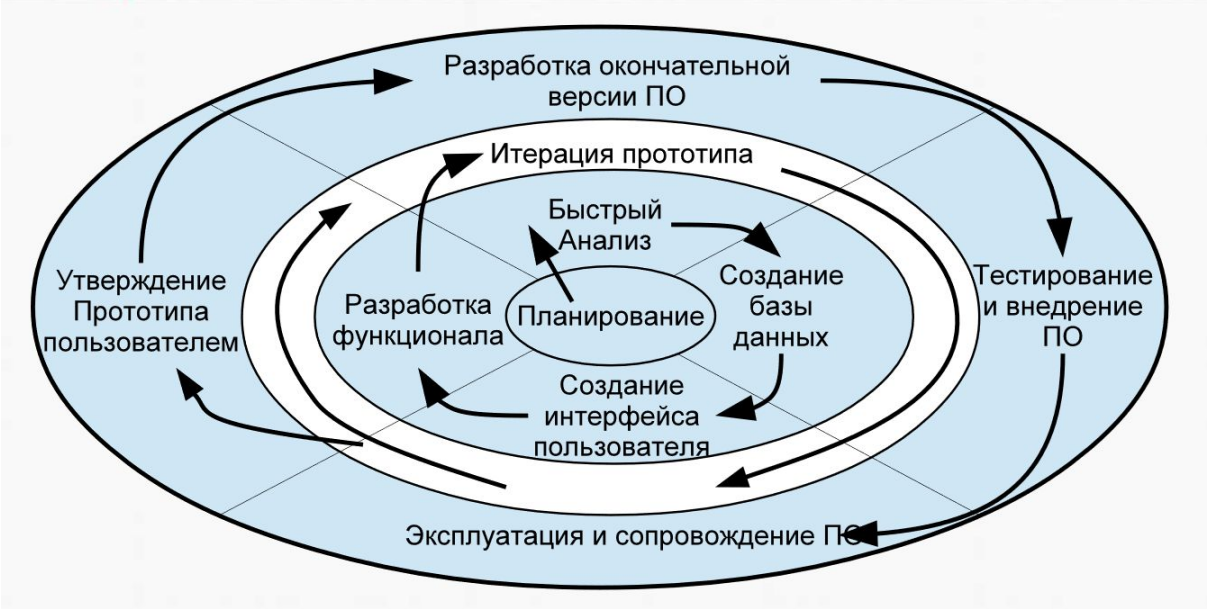
Это эволюционная модель. Прототип не выполняет всех функций, но показывает пользователю то, как принципиально будет выглядеть работа продукта.

- планируется вся итерация
- проводится быстрый анализ

Итерация состоит из:

- создается база данных и интерфейс пользователя разрабатывается функционал
- проверяется совместно с пользователями системы

- если пользователь доволен выполняются оставшиеся три этапа, иначе итерация повторяется



8.RAD методология.

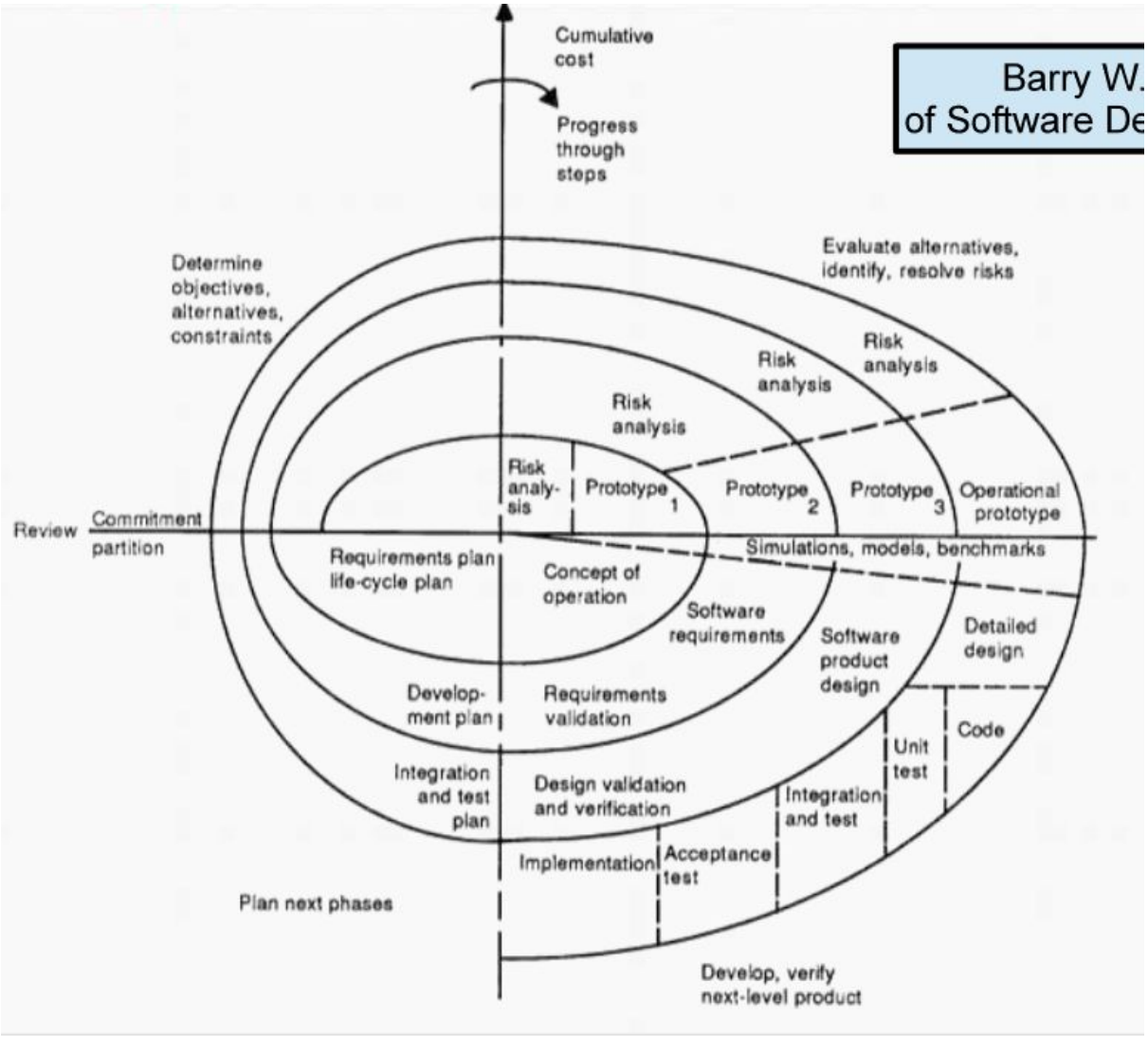
Сначала проводится проектирование, а затем, при помощи средств автоматизации, как из кубиков, собирается ПО.

9.Спиральная модель.

Каждый виток спирали - фаза разработки продукта

Витки состоят из:

- Постановка целей
- Выявление альтернативных решений и ограничений (для текущей фазы)
- Анализ рисков (перед созданием прототипа)
- Разработка прототипа и его валидация:
    - основные концепции ПО
    - функциональные требования
    - дизайн продукта
    - дизайн компонентов
    - кодирование
- В конце последней итерации повторяется часть V-образной модели (часть с тестами)
- На последней части витка производится планирование следующей итерации
- Изменения, вносимые в проект, - неотъемлемая часть разработки и каждое изменение может быть принято, отклонено, или проигнорировано (в зависимости от рисков)



10.UML Диаграммы: Структурные и поведенческие.

- структурные диаграммы
  - диаграмма классов/объектов
    - на уровне анализа

      доменная модель для описания предметной области, без деталей реализации
    - на уровне проектирования

      доменная модель для иллюстрации архитектурных механизмов с деталями реализации
  - диаграмма развертывания (размещения)

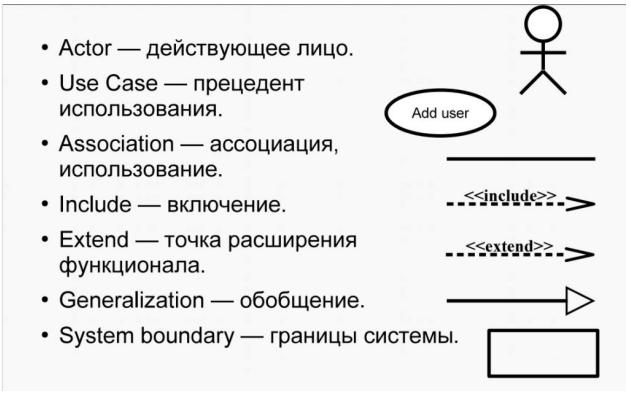
    описывает архитектуру системы с деталями установки компонентов, их точных версий и структуры взаимной вложенности.
  - диаграмма пакетов
  - диаграмма компонентов

- поведенческие диаграммы
  - диаграмма деятельности

    определяют логику и последовательность алгоритма

- диаграмму состояний
  - определяют логику и последовательность внутренних состояний
- диаграмму вариантов использования (use-case)
  - описывает высокоуровневые требования к системе
- диаграмму последовательности
  - определяют логику и последовательность взаимодействия

11.UML: Use-Case модель.



Use-Case Model определена как инструмент графического отображения требований.

Актор - взаимодействует с системой

Use Case - глагол или глагольная фраза, которая указывает что должна сделать система

Association - связь, указывающая на то, что актор использует систему и выполняет use case

Include - включение деятельности в несколько use case

Extend - точка расширения функционала

Generalization - обобщение - показывает, что данный объект является потомком более высокого класса

System boundary - границы системы

12.UML: Диаграмма классов.

Основным преимуществом является наглядное изображение предметной области или архитектурных механизмов. Диаграмма по мере дальнейшей работы может быть уточнена и расширена.

- на уровне анализа
  - доменная модель для описания предметной области, без деталей реализации
- на уровне проектирования
  - доменная модель для иллюстрации архитектурных механизмов с деталями реализации

13.UML: Диаграмма последовательностей

диаграмма, на которой для некоторого набора объектов на единой временной оси показан жизненный цикл какого-либо определенного объекта (создание-деятельность-уничтожение некой сущности) и взаимодействие актеров (действующих лиц) ИС в рамках какого-либо определённого прецедента (отправка запросов и получение ответов).

14.UML: Диаграмма размещения

Диаграмма размещения показывает физическую архитектуру размещения частей приложения на серверах. Она описывает архитектуру системы с деталями установки компонентов, их точных версий и структуры взаимной вложенности.

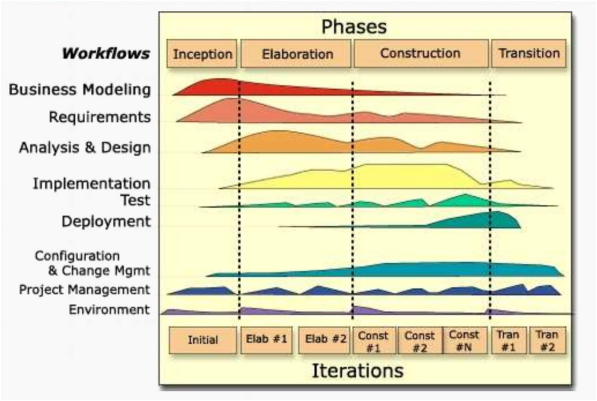
15.*UP методологии (90-е). RUP: основы процесса.

Процесс разбивается на фазы (которые в свою очередь могут быть разбиты на итерации при итеративном подходе). Дисциплина - набор правил и указаний для решения определенной задачи.

В RUP четко определен процесс разработки, описано около 30 ролей.
Роль - группа обязанностей, который берет на себя участвующий в разработке человек. У каждой роли есть свой набор деятельностей. На входе и выходе деятельности используются, создаются или модифицируются артефакты (любой результат труда роли).
Деятельность, которая производится ролью, всегда производится согласно установленным правилам.

Любая фаза заканчивается вехами.
Веха — момент принятия решения о дальнейших действиях: переход на следующую фазу, либо проведение дополнительных работ на данной фазе.

- Есть фазы:
  - Inception (начало)
  - Elaboration (проектирование)
  - Construction (создание)
  - Transition (внедрение)
- Есть дисциплины:
  - бизнес моделирование
  - требования
  - анализ и дизайн
  - имплементация
  - тесты
  - развертывание
  - менеджмент изменений
  - проектный менеджмент
  - окружение



16.RUP: Фаза «Начало».

Основное направление работ в фазе начало - оценить проект, понять сколько нужно ресурсов и времени на его реализацию. Формируется артефакт Концепция (Vision).

- Начало:
    - Определить границы проекта. Определить решаемые и нерешаемые задачи.
    - Разработка и описание основных основных сценариев использования системы.
    - Предложение возможных технических решений (API)
    - Подсчет стоимости и разработка графиков работ.
    - Оценка рисков.

- Веха “Lifecycle Objects”:
    - заинтересованные лица пришли к согласию сроков, стоимости, требованиях, приоритетах и технологиях
    - оценены риски и выбраны стратегии смягчения последствий

17.RUP: Фаза «Проектирование».

Основная задача - разработка и тестирование стабильной и неизменной архитектуры системы, создание одного или нескольких прототипов системы, которые определяют исполняемую архитектуру. Тестирование проводится для нефункциональных требований к системе.

Исполняемая архитектура — это полностью законченные на базе выбранных технологий несколько характерных функций разрабатываемой системы.

Веха “Lifecycle Architecture”: веха стабильности архитектуры.

- Стоимость и сроки соблюдены или приемлемы.
- концепция, требования и архитектура стабильны
- сформулированы критерии тестирования прототипов
- Тестирование прототипов показало отсутствие основных рисков

18.RUP: Фаза «Построение».

Основная задача - экономически выгодно и надлежащим образом разработать ПО.

Проводится создание необходимых версий продукта и результат работы показывается заказчику. В конце фазы производится подготовка продукта, место установки и пользователей к передаче заказчику.

Кардинально менять архитектуру нельзя.

- Веха “Initial Operational Capability”:
    - внедрения ПО к заказчику возможно
    - затраты приемлемы
    - стороны готовы

19.RUP: Фаза «Внедрение».

Предназначена для запуска продукта в пользование и финальное подтверждение готовности продукта.

- Цели:
    - Маркетинг
    - перенос данных
    - обучение
    - отладка процессов устранения сбоев
    - проверить соответствие ПО изначальной концепции
    - бета-тестирование

- Веха “Product Release”:
    - пользователи удовлетворены
    - сделать выводы из реальных и спрогнозированных затрат

20.Манифест Agile (2001).

Agile - подход к разработке ПО, позволяющий быстро и гибко реагировать на изменения технологий и общества. Agile-процессы позволяют использовать изменения для предоставления заказчику конкурентного преимущества. Во главе Agile-подхода ставятся требования заказчика и реакция на них. Однако данный подход невозможен, если на проект выделяется фиксированный бюджет. Agile-методологии хорошо работают для разработок, где заказчик платить за рабочие часы работников. Разработчики со своей стороны получают деньги и разрабатывают новый функционал.

Также важным принципов является сокращение расходов на деятельность не связанную с разработкой продукта.

21.Scrum.

В Scrum процесс разработки сильно упрощен. Единственным служебным артефактом является backlog - список заданий, упорядоченный трудоемкости. Другим артефактом является инкремент продукта. Спринт - период времени от 2 до 4 недель, в который разработчики реализуют выбранный набор требований из бэклога. Каждый спринт заканчивается демо-версией и демонстрируется заказчику.
В команде Scrum от 3 до 10 человек, также выделяется особая роль Product Owner - человек, определяющий порядок выполнения задач из бэклога.
Достоинства Scrum - простота и концентрация на коде, однако лучше всего подходит для небольших проектов.

22.Disciplined Agile 2.X (2013).

Disciplined Agile 2.X или DAD - по делению на фазы напоминает RUP, однако основной цикл разработки (Construction) построен на базе гибких методов (таких как Scrum). Помимо деления на фазы (Начало, Построение, Внедрение) DAD рассматривает процессы, выходящие за рамки разработки. К ним относят: управление архитектурой, повторное использование кода, управление персоналом и. т. д. Кроме того такой подход
к разработке подходит для больших команд и проектов.

23.Требования. Иерархия требований.

Требование - условия или возможности, которым должна соответствовать система. Обычно это подробное описание того, что должно быть реализовано.Оно должно быть однозначно сформулировано и описывать то, что должна выполнять система. Обычно требования описываются с помощью SRS (Software Requirement Specification) или Use-case Model.

Иерархия требований состоит из 3 категорий:

- Потребности - представлены в виде пожеланий к системе
- Функции - представлены в виде четких инструкций, понятных заказчику
- Требования к ПО - представлены в виде более четких инструкций, которые понятны разработчику.

24.Свойства и типы требований (FURPS+).

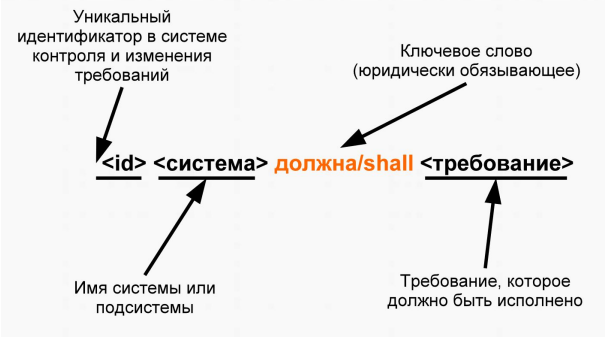
Свойства требований:

- Корректность
- Однозначность
- Полнота
- Непротиворечивость
- Приорезитация
- Проверяемость
- Модифицируемость
- Отслеживаемость

Кроме того, для описания требований используется FURPS+ (типы требований):

- Functionality - Функциональные требования
- Нефункциональные требования, включающие в себя:
    - Usability - требования к удобству использования
    - Reliability - требования к надежности
    - Performance - требования к производительности
    - Supportability - требования к поддержке
- +: дополнительные требования
    - Ограничения проектирования / архитектуры
    - Требования к реализации, интерфейсам
    - Физические требования

25.Формулирование требований. Функциональные требования



- Определяют:
  - Feature sets — наборы функциональных требований.
  - Capabilities — возможности ПО.
  - Security — Требования к безопасности.

FR0 «Система должна обеспечивать ввод, модификацию и удаление данных о клиенте».

SEC0 «Система должна обеспечивать двухфакторную аутентификацию пользователей с помощью имени пользователя и пароля и подтверждения с помощью СМС».

26.Требования к удобству использования и надежности.

- Требования к удобству (Usability) включают в себя следующие пункты:
    - Human Factors - учет особенностей пользователя
    - Aesthetics - эстетические требования
    - UI - пользовательский интерфейс
    - Help system - требования к справочной подсистеме
    - Wizards and agents - мастера и ПО, повышающие продуктивность
    - User Documentation - требования к пользовательской документации

    - Training Materials - требования к учебным материалам

- Требования к надежности (Reliability) включают в себя следующие пункты:
    - Frequency and Severity of failure - частота и обработка отказов
    - Recoverability - способность системы восстанавливать работоспособное состояние (в течении определенного времени)
    - Predictability - предсказуемость поведения
    - Accuracy - точность
    - MTBF - Mean Time Between Failures - среднее время между отказами

27.Требования к производительности и поддерживаемости.

- Требования к производительности (Performance):
    - Speed - скорость решения задач
    - Efficiency - эффективность
    - Availability - способность системы к решению задач
    - Throughput - пропускная способность
    - Response Time - время отклика
    - Recovery Time - время восстановления
    - Resource Usage - использование ресурсов

- Требования к поддерживаемости (Supportability):
    - Extensibility - расширяемость
    - Adaptability - адаптируемость под конкретные задачи
    - Maintainability - поддерживаемость
    - Compatibility - совместимость
    - Configurability - способность задавать конкретную модификацию
    - Serviceability - возможность к обслуживанию
    - Installability - требования к установке
    - Localizability - требования к локализации

28.Атрибуты требований.

Атрибут “приоритет” требований задается либо в численном значении, либо согласно MoSCoW:

- Must have - фундаментальные для системы требования
- Should have - важные
- Could have - потенциально возможные
- Won’t have - могут быть реализованы позже

Кроме того, требования могут иметь и другие атрибуты:

- Статус
    - предложенные, одобренные, отклоненные, включенные.
- Трудоемкость
    - человеко-часы, функциональные точки, use-case points, «попугеи».
- Риск
- Стабильность - частота изменения требования
- Целевая версия - версия ПО в которой будет реализовано требование

29.Описание прецедента.

В описание прецедента должны быть следующие параметры, характеризующие его:

- Название (Name)
- ID
- Краткое описание
- Эктор
- Предусловия
- Основной поток
- Постусловия
- Альтернативный поток

30.Риски. Типы Рисков.

Риск - потенциально опасный для проекта фактор, который может привести к срыву сроков реализации проекта или привести к невозможности реализации.
Риски делятся на:

1. Прямые - можем управлять
2. Непрямые - не можем управлять

1. Ресурсные - связаны с недостатком у компании времени, людей, денег, оборудования
2. Бизнес-риски - связаны с другими компаниями
3. Технические риски - связаны с разработчиками
4. Политические риски - связаны с компанией
5. Форс-мажор

31.Управления рисками. Деятельности, связанные с оценкой.

Управление рисками как дисциплина разбивается на две больших сферы деятельности:

оценка риска и контроль и управление рисками.
Внутри оценки риска производится его идентификация, анализ и назначение ему приоритета.
После его можно поместить под управление системы контроля риска. Если система контроля определит его наступление, будет предприняты корректирующие действия.

Деятельности связанные с оценкой:

- Идентификация риска
    - В модели Карра риски делятся на классы, классы на элементы, элементы на аттрибуты. Первый уровень идентификации рисков - классы, разделяется на 3: Product Engineering, Development Engineering, Program Constraints
    - Также риски делятся на известные, неизвестные и непознаваемые.

- Анализ риска
    - использование разных моделей и методов анализа: стоимостной, сетевой и. т. д.
    - Два основных параметра риска - вероятность его наступления и масштаб. возможных потерь. Эти параметры могут оцениваться по пятибалльной системе, математической формуле или шкале.
- Приоритизация риска
    - Экспозиция риска (Risk Exposure) - произведение вероятности риска и величиной потерь
    - После подсчета данных рисков, создается документ Топ-10 рисков, которые наиболее убыточны по данной величине

32.Управления рисками. Деятельности, связанные контролем и управлением.

Управление рисками как дисциплина разбивается на две больших сферы деятельности:

оценка риска и контроль и управление рисками.

Внутри оценки риска производится его идентификация, анализ и назначение ему приоритета.

После его можно поместить под управление системы контроля риска. Если система контроля определит его наступление, будет предприняты корректирующие действия.

Деятельности связанные с с контролем и управлением:

- Планирование реакции на риск
    - Избегание риска - комплекс мероприятий, направленный на отсрочку или исключение вероятности наступления риска.
    - Перенос риска - комплекс мероприятий, направленных на перенос риска 3-им лицам.
    - Сокращение риска
    - Прием риска
- Мониторинг рисков
    - Контролируются Топ-10 рисков
    - Переоценивание рисков
    - Автоматизированные системы (Jira)
    - Все это происходит во время разработки
- Разрешение неопределенностей,связанных с риском.
  Достигается за счет:
    - Контролируемого и журналируемого обновления системы
    - У каждого изменения есть ID, дата, ответственный и описание

- Управление изменений

33.Изменение. Общая модель управления изменениями.

В разработке ПО не только изменения программного кода могут попадать под контроль отдельному управлению изменениями обычно подлежат сами требования к ПО, выявленные дефекты, артефакты архитектуры, анализа и дизайна.

- Запрос и анализ**:**

  Заказчик (Customer) требует новую функциональность (Require new functionality) или исправление выявленных им проблем (Encounter Problem). Каждая из этих деятельностей приводит к созданию документа(артефакта): требования к системе (Requirements) или отчет о проблеме (Problem Report) соответственно. Оба типа формируют Change Request, который добавляется в Change Log Entry.

  Далее Project Manager определяет техническую необходимость и осуществимость изменений (Technical Feasibility) и определяет costs and benefits. Далее комитет по изменениям ставит статус для изменения: подтверждено, отклонено или отложено.

- Подтверждение и реализация:

  Далее анализируется возможное влияние на другие системы и возможные последствия для пользователей (Analyze Change Impact), после создается план действий для внесения изменений (Create Planning). В результате этих деятельностей появляется Change Impact Analysis и Change Planning. Затем изменения передаются к реализации (Роль Change Builder), создаются объекты (Items), которые изменяются. После проведения изменений, создаются Test Report, Documentation, System Release.

- Проверка и закрытие реализации.

  В конце процесса изменения передаются менеджеру проекта, проверяются (создается артефакт Change Verification), и формально утверждаются.

34.Системы контроля версий. Одновременная модификация файлов.

Система контроля версий выполняют следующие функции:

- Управляют изменениями в программном коде
- Поддерживают групповую работу

Существуют три типа основных систем контроля версий

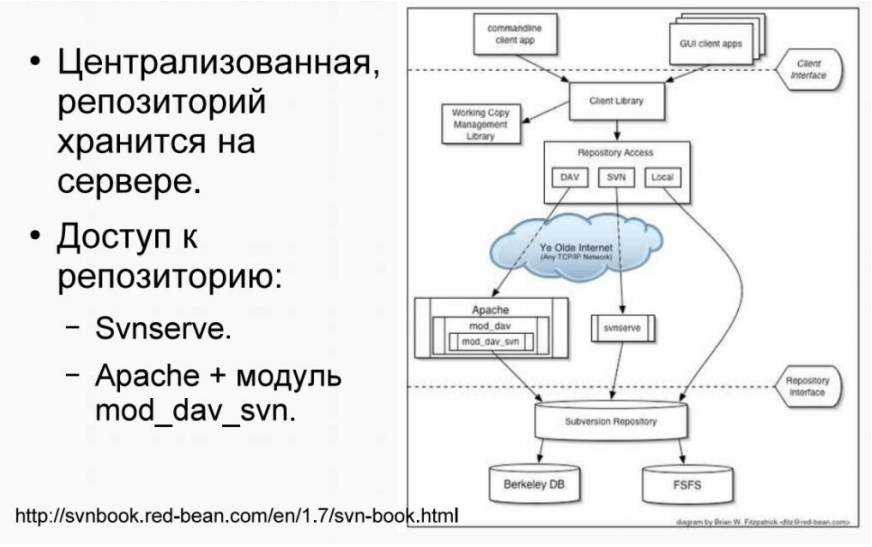
1. На основе файловой системы (центральный сервер для общего доступа к файлам) - устарела
2. Централизованная - с единым репозиторием (SVN)
3. Распределенная - есть центральный репозиторий, который пользователи скачивают в локальные репозитории. Работают с ними. Результат работы обратно в центральный репозиторий. (Git)

Существует 2 подхода к одновременной модификации файлов:

- Lock-modify-unlock. При работе над файлом, он блокируется для других. Это замедляет работу.
- Copy-modify-merge. Каждый пользователь копирует себе репозиторий. Делает изменения. Далее они сливаются в общий репозиторий. Главная проблема - слияние рабочих копий.

35.Subversion. Архитектура системы и репозиторий.

Subversion (SVN) - централизованная система контроля версий, где уровнем хранения может иметь 2 технические реализации - файловая система (FSFS) или Berkley DB. Доступом к репозиторию управляет демон svnserve, который получает команды от пользователя и выполняет соответствующие изменения в репозиторий. Кроме того, есть решение доступа к репозиторию через сервер Apache (+modules). Удаленный доступ к репозиторию может осуществляться по протоколам svn, https+svn, ssh+svn. Также SVN управляет локальной копией файлов, которые модифицирует разработчик.
trunk - основная ветвь разработки
branches - хранение модификаций продукта
tags - функционально целостные малые изменения.
При создании новой ветки - увеличивается номер ревизии и копируются файлы.



36.Subversion: Основной цикл разработчика. Команды.

Основной цикл разработчика:

- svn checkout - первоначальное создание рабочей копии
- svn update - обновить рабочую копию
- svn add/delete/copy/move/mkdir
- svn status/diff - просмотр изменений

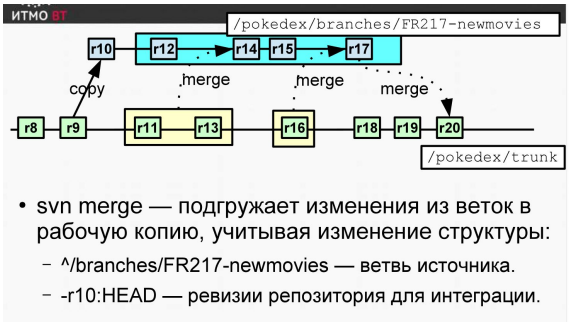
- svn revert - откат изменений
- svn update - для загрузки изменений других
  -- Разрешаем конфликты --
- svn commit - фиксация изменений

37.Subversion: Конфликты. Слияние изменений.

Основная проблема фиксации изменений - конфликты содержимого файлов.
SVN о них бессовестно сообщает и предлагает выбрать один из вариантов:

- (e) - edit - изменить файл в редакторе
- (df) - diff full - показать несоответствия
- (r) - resolve - подтвердить, что конфликт решен
- (dc) - display conflict - показать все конфликты
- (mc) - mine conflict / (tc) - their conflict - подтвердить мои (репозитория) версии для всех конфликтов
- (mf) - mine full / (tf) - their full - подтвердить версии полностью для файла
- (p) - postpone - отложить на потом (создается file.mine, file.rOLDREV, file.rNEWREV) - обычно выбирают его
- (l) - lauch - запустить внешнее средство

Конфликты структуры происходят, когда в репозитории файлы перемещаются, а в рабочей копии они же изменяются



38.GIT: Архитектура и команды.

GIT - распределенная система контроля версий. В ней существует удаленный репозиторий origin. Центральный репозиторий любого проекта может быть получен с помощью fork - организации ветви от другого проекта. У каждого разработчика есть свой локальный репозиторий, который является клоном удаленного репозитория. Разработчик выбирает необходимую ветвь в локальном репозитории, получая

представление на файловой системе файлов с содержимым ветви, которые он может изменять.
Во время работы разработчик указывает изменения каких файлов необходимо фиксировать. Эти изменения помещаются в Stage Area. Далее с помощью команды push изменения фиксируются в центральном репозитории.
Другой способ фиксации изменений - pull request - запрос на на фиксацию изменений, которые необходимо подтвердить владельцу ветви.

Основные команды:

- git init - создание репозитория
- git clone - клонирование репозитория
- git checkout - переход в необходимую ветвь
- git branch - создание новой ветви
- git add - добавить файл в stage area
- git commit - фиксация изменений
- git push - отправка результатов в удаленный репозиторий
- git pull - обновление локального репозитория
- git merge - слияние ветвей в одну
- git status - статус текущих состояний файлов
- git log - журнал коммитов
- git diff - показывает изменения

39.GIT: Организация ветвей репозитория.

Список ветвей:

- Master - основная ветвь, в которой находятся основные версии продукта, готовые к поставке пользователю
- Develop - ветвь разработки. Берет начало от master. Весь код разработчики сохраняют именно сюда. Обычно реализуются функциональные требования, причем каждое из них в отдельной ветке feature.
- Release. Когда определенный функционал готов к выпуску новой версии, создается ветвь release, и исправляются ошибки, которые препятствуют попаданию программы в master. После того, как эти ошибки были исправлены, результат идет в master.
- Featur - новая функциональность
- Hotfix - ветвь исправления критических ошибок в версии пользователя. После исправления результат попадает в master и develop.

40.GIT: Плагин git-flow.

Из-за обилия команд в git и сложности синтаксиса был создал плагин для git - git flow. Он позволяет работать с версиями в терминах версий, а не операций.

41.Системы автоматической сборки: предпосылки появления

Процесс сборки, выполняемый разработчиками - не самая приятная для них работа из-за трех вещей:

- Рутинный процесс сборки. При сборке необходимо повторять одинаковую последовательность команд. Естественным решением будет создать скрипт сборки, однако из-за того, что структура проекта может сильно меняться - это может плохо сказаться на производительности
- Отличия архитектуры систем. (Аппаратное обеспечение, ОС)
- Медленная сборка.

Эти 3 пункта привели к созданию средств для автоматической сборки. Они:

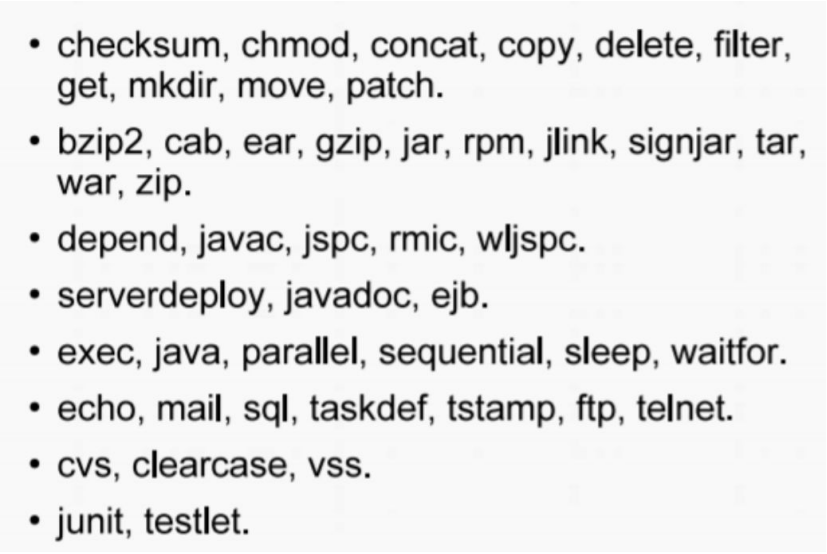
- Используют специальные языки для организации компиляции
- Позволяют задать конфигурацию системы
- Могут организовывать декларативную сборку
- Поддерживают параллельный режим работы
- Интегрируются с build-серверами и автоматическими тестами

42.Системы сборки: Make и Makefile.

Make - низкоуровневая утилита для сборки программного кода. В файле сборки (makefile) указаны опции компилятора и необходимые зависимости. В makefile существуют цели и файлы, от которых они зависят, далее последовательность действий, которые нужно выполнить для сборки.
Сборка начинается с цели all, после чего собирает программу с учетом зависимостей и временем последней модификации файла. Одним из существенных недостатков является неудобство использования, так как нужно следить за большим количеством макропеременных.

43.Системы сборки: Ant. Команды Ant.

Императивная система сборки, которая управляется файлом build.xml. Описание проекта состоит из целей (target) с явным указанием зависимостей других целей. Кроме этого внутри каждой цели указаны действия, которые необходимо выполнить в процессе сборки. Цели можно вызвать явно или из другого файла, что позволяет собирать систему из модулей.
В Ant есть свойства, которые можно менять, чтобы собирать проект в разных условиях. Их можно задать явно значением, переменной окружения или отдельным файлом.
В ant существует большое число команд, необходимых на все случае жизни при сборке Java-приложения. Это позволяет сделать сборку платформонезависимой.



44.Системы сборки: Ant-ivy.

Ivy - менеджер зависимостей для ant позволяющий работать с репозиториями Maven. Зависимости задаются в специальном файле ivy.xml. Ранее зависимости в Ant нужно было скачивать и подключать вручную, однако с помощью ivy становится легче использовать ant.

45.Системы сборки: Maven. POM. Репозитории и зависимости.

Maven - декларативная система сборки, управляющийся на базе специального файла pom.xml (Project Object Model). В POM указаны имя, версия и тип программы, местонахождение исходников, зависимости, плагины и альтернативные конфигурации проекта (профили).
Цель по умолчанию у maven - install. Назначение цели в том, чтобы в локальном репозитории пользователя была собрана версия продукта.

mvn install работает в несколько шагов:

1. Maven читает файл pom.xml с информацией о проекте и определяет необходимые зависимости
2. Maven скачивает необходимые для сборки файлы в локальный репозиторий
3. Maven зависимости сохраняет в локальный репозиторий
4. Осуществляется сборка
5. Результат сборки помещается в target и локальный репозиторий

Также есть возможность помещения продукта в удаленный репозиторий.

46.Maven: Структура проекта. GAV.

В Maven существует система каталогов по умолчанию.

- target - целевая директория
- src/main - основные исходные файлы
    - src/main/java - java файлы для компиляции
    - src/main/webapp - web-файлы

    - src/main/resources - файлы ресурсов, которые нет необходимости компилировать
- src/test - исходные файлы для тестов
    - src/test/java - java файлы тестов
    - src/test/resources - файлы ресурсов

GAV-синтаксис - groupId:artifactId:version
Система наименований в Maven строится по принципу GAV. Наименование программы по этому синтаксису описывается в pom.xml и используется при подключении сторонних библиотек.

47.Maven: Зависимости. Жизненный цикл сборки. Плагины.

Зависимости Maven - транзитивные, и описываются в специальное блоке dependencies.
Каждая зависимость имеет несколько параметров:

- GAV-синтаксис
- Scope: **compile,** provided, runtime, test, system
- Type: **jar,** war, pom, ear, zip

Жизненный цикл сборки в Maven приложении выполняется последовательно, шаг за шагом. В нем вызываются все цели стадии сборки, а именно:

1. generate-sources / generate-resources - применяется, когда часть приложения автогенерируется
2. compile - исходный код компилируется
3. test-compile - компилируются тесты
4. test - проводятся тесты
5. package - приложение собирается в пакеты
6. integration-test - проводится интеграционное тестирование
7. install - приложение устанавливается в локальном репозитории
8. deploy - происходит загрузка на сервер приложений

Все операции над проектом выполняются плагинами, они используются для управления сборкой. Их надо в явном виде включать в pom. Плагины в качестве точек входа используют цели, связанные с жизненным циклом.

48.Gradle.

.

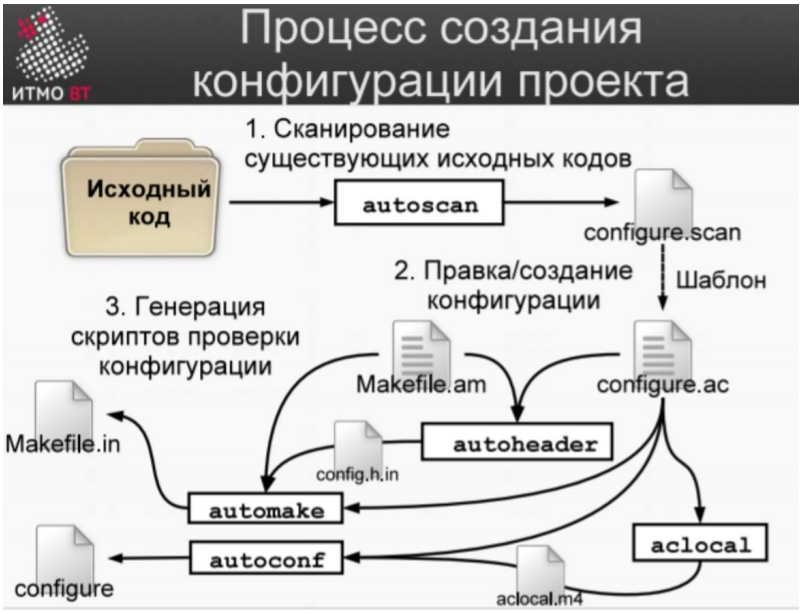
- Нейтрален к языкам программирования
- Базируется на Apache+Ivy
- Подобно Maven использует плагины и жизненный цикл
    - Основные задачи по сборке определены в плагинах
    - Плагины определены для многих типов проектов
- Описание зависимостей от Maven (GAV)
    - Может использовать репозитории Maven и Ivy
- В качестве скрипта сборки использует DSL (Domain Specific Language) на Groovy
- Инкрементальная и параллельная сборка

49.Системы сборки: GNU autotools. Создание конфигурации проекта.

GNU autotools создает makefile используя в качестве системы сборки make на нижнем уровне. На процесс генерации оказывают влияние режимы работы программы, заданная конфигурация, операционная система и фактическое наличие библиотек.

gnu autotools

- использует макропроцессор m4
- используется для программ с открытым исходным кодом
- пользователю нужно знать всего 3 команды - ./configure; make; sudo make install.



50.Системы сборки: GNU autotools. Конфигурация и сборка проекта.

Создание конфигурации проекта осуществляется с помощью 3 шагов:

1. Autoscan - последовательно сканирует существующий код, выделяет участки кода, которые могут зависеть от платформы и ОС. По результатам сканирования получается файл configure.scan, который затем редактируется вручную. В результате получается configure.ac
2. Autoheader - с помощью configure.ac и предварительно созданного файла makefile.am создает шаблок файла config.h.in для файла config.h. В это время aclocal проверяет, что в системе разработчика есть необходимые библиотеки.
3. Далее запускаются утилиты automake и autoconf в результате чего появляются файлы Makefile.in и configure, которые используются позже.

Процесс сборки проекта также включает в себя три шага:

1. ./configure - вызов файла конфигурации. Создает файл config.h и необходимые платформозависимые файлы.
2. make - собирает файлы с помощью config.h и makefile.
3. sudo make install - файлы распределяются в необходимые системные каталоги

51.Сервера сборки/непрерывной интеграции.

Сервера сборки/непрерывной интеграции нужно для многократной сборки проекта в автоматическом режиме. Их основная функция - сборка новой версии продукта при заданных заранее условиях. Кроме этого такие сервера умеют проводить тесты, снимать метрики с программного кода и производить запуск. Кроме этого они предоставляют доступ ко всем собранным версиям продукта.

52.Основные понятия тестирования. Цели тестирования.

Основные понятия тестирования:

- Mistake - ошибка человека
- Fault - дефект в результате ошибки
- Failure - отказ в результате дефекта
- Error - невозможность выполнить задачу из-за отказа
- (BUG) - может означать все вышеперечисленное

Цели тестирования:

- Обнаружение дефектов
- Повышение уверенности в уровне качества
- Предоставление информации для принятия решения
- Предотвращение дефектов

53.Понятие полного тестового покрытия и его достижимости. Пример.

Полное тестовое покрытие - тестирование всех возможных вариантов, подающихся на вход в программу. Полное тестовое покрытие на практике недостижимо, или достижимо путем очень больших затрат. Для функции умножения 2 значений типа int на 3 ГГц процессоре может уйти до 180 лет.

54.Статическое и динамическое тестирование.

Статическое тестирование:

- Не включает выполнение кода
- Может быть как ручным, так и автоматизированным
- Неформальное (различного рода инспекции)

Динамическое тестирование:

- Запуск модулей групп всей системы
- Проводится после появление кода (однако при TDD (Test-Driven Development) - перед)

55.Автоматизация тестов и ручное тестирование.

Автоматизация тестов - положительное явление, но часто ручное тестирование оказывается дешевле и проще.
Отдельный вид тестирования - регрессионное тестирование, которое заключается в запуске старых тестов, чтобы проверить, что при изменении программы ее работоспособность не была нарушена. Чтобы обеспечить регрессионное тестирование - необходимо обеспечить однозначное повторение тестового сценария.

Еще важной проверка одного и того же приложения в разных окружениях то есть тестирование совместимости.

Так же есть приемочное тестирование, чаще оно проводится вручную

56.Источники данных для тестирования. Роли и деятельности в тестировании.

Существует два подхода к тестированию: метод черного и белого ящика.

В методе черного ящика подразумевается, что внутреннее содержимое программы скрыто. Работа при таком тестировании идет на уровне спецификации или на основе опыта тестировщика. Тесты подают на вход программы исходные данные и сравнивают результат работы программы с эталоном.

В методе белого ящика возможно исследовать исходный код. Метод подразумевает построение графа с целью определения тестового покрытия. Определяется цикломатическая сложность программы, которая определяет число путей обхода существующего кода программы, и соответственно максимальное необходимое число тестов.

Роли и деятельности в тестировании делятся на несколько больших категорий:

- Проектирование тестов
- Автоматизация тестов
- Исполнение тестов
- Анализ результатов тестов

57.Понятие тестового случая и сценария.

Тестовый случай состоит из набора входных значений, предусловий выполнения, ожидаемого результата и постусловий.
Набор входных значений - набор данных, на которых разрабатываемое ПО должно вести себя определенным образом с точки зрения спецификация или требований к программе.
Тестовый случай должен быть повторяемым, то есть для одинакового набора значений, должен выводится одинаковый результат.
Тестовый случай должен быть автоматизированным.
Тестовый случай должен учитывать состояния внутри ПО и переходы между ними.

Тестовый сценарий - последовательность тестовых случаев. Обычно описывает типичное пользование системой. В таких сценариях должны обрабатываться как положительные, так и отрицательные реакции системы на действия пользователя. Для неправильных действий пользователя также создаются тестовые сценарии.

58.Выбор тестового покрытия и количества тестов. Анализ эквивалентности.

При планировании тестирования следует соблюдать баланс между качеством и скоростью вывода продукта в эксплуатацию.
Существует 4 метода для выбора тестового покрытия:

- Эквивалентное разбиение - функция разбивается на отрезки, в которых ведет себя одинаково, проверяются точки внутри функции и граничные значения.
- Таблица решений - составляется таблица, отражающая комбинации входных данных с соответствующими выходными данными
- Таблица переходов - выделяются явные состояния внутри системы, определяются переходы между состояниями, которые в дальнейшем покрываются тестами.
- Сценарии использования - учитывает основной и альтернативные пути сценария. При этом используются конкретные значения, вводимые пользователем.

59.Модульное тестирование. Junit 4.

Модульное тестирование - тестирование отдельных компонентов программного обеспечения. Обычно тестируется методы или классы, или их совокупность, которая определяется тестировщиком как отдельный модуль.
Так как модули могут обращаться друг к другу, а корректное их поведение сильно зависит от поведения другого модуля, то в тестировании отдельного модуля может быть применена заглушка, которая выдает заранее известные значения. Это называется **изолированием модулей.** Также, для вызова конкретного модуля, используются драйвера. Их основная функция - последовательный вызов тестируемого метода с различными входными параметрами и условиями.

JUnit - фреймворк для модульного тестирования Java-приложений:

- @Test - аннотация для маркировки метода, как теста
- @Before, @After, @BeforeClass, @AfterClass - аннотации для вызова методов до и после теста
- Методы проверки (Assertions)
- UI, Журнал Тестов

60.Интеграционное тестирование. Стратегии интеграции.

Интеграционное тестирование - проверяет связь между компонентами системы и проводится после модульного тестирования. Смысл тестирования - проверка интерфейсов взаимодействия конкретных модулей. Оно может производится, когда, как минимум, два компонента системы уже разработаны. Остальные

добавляются по готовности. Выбор модели интеграции выбирается на основе разрабатываемого приложения.

Стратегии интеграции:

- Сверху-вниз - выбор для бизнес-приложений. Сначала появляется Бизнес логика с использованием драйверов и заглушек, затем UI, затем уровень хранения.
  - +: быстро появляется “осязаемое” приложение
  - -: большое количество заглушек
- Снизу-вверх - выбор для ПО, которое сильно зависит от аппаратуры. Сначала разрабатывается слой для работы с аппаратурой, далее логика, и только потом UI. Плюсы и минусы - наоборот.
- Другие:
  - функциональная (UI-Логика-БД).
  - ЯДРО (формируется минимальный работоспособный функционал, а потом добавляются остальные функции).
  - Большой взрыв (собираем все и молимся).

61.Функциональное тестирование. Selenium.

- На базе сценариев использования.
- Ручное/автоматическое.
- На готовой системе, в рамках модульного и интеграционного.
- Проверяются функции системы, начиная с интерфейса пользователя.
- Средства автоматизации:
  - Открытые: Selenium, Sahi, Watir.
  - Коммерческие: от HP, Rational (IBM)…

62.Техники статического тестирования. Статический анализ кода.

Рецензирование может проводиться вручную или с помощью средств, главной составляющей ручного тестирования является исследование и комментирование продукта.

	Сквозной контроль	Технический Анализ	Инспекция
Основное Предназначение	Поиск дефектов	Поиск дефектов	Поиск дефектов
Дополнительная цель	Обмен опытом	Принятие решений	Улучшение процесса
Подготовка	Обычно нет	Популяризация	Формальная подготовка
Ведущий	Автор	В зависимости от обстоятельств	Подготовленный модератор
Рекомендованный размер группы	2-7	>3	3-6
Формальная процедура	Обычно нет	Иногда	Всегда
Объем материалов	небольшой	От среднего до большого	небольшой
Сбор метрик	Обычно нет	Иногда	Всегда
Выходные данные	Неформалный отчет	Формальный отчет	Список дефектов, результаты метрик, формальный отчет

Есть несколько техник:

- друг - Одной из распространенных техник является рецензия коллегой. Глаза человека замыливаются, и мы можем не

замечать очевидных ошибок. Однако коллега может ошибаться или быть субъективным.

- Технический анализ: проводится анализ под руководством лидера проекта.
- Management review: менеджер разработки
- Сквозной контроль: просмотр специальным экспертом, который фиксирует недочеты и дефекты.
- Инспекции: много инспекторов у каждого из которых есть 2 собственные роли.

Преимущества статического тестирования:

- раннее обнаружение и исправление дефектов
- улучшение продуктивности разработки
- уменьшение времени разработки
- уменьшение времени и стоимости тестирования
- уменьшение числа дефектов
- улучшение отношений в команде
- передача опыта

Средства стат анализа проводят проверку кода на неопределенное поведение (не инициализирована переменная), нарушение алгоритмов, разрушение кроссплатформенности и тд. Пример: FindBugs, Lint (строят синтаксическое дерево)

63.Тестирование системы в целом. Системное тестирование. Тестирование производительности.

Тестирование системы в целом начинается после окончания интеграции. На этом этапе нужно провести проверку заявленных характеристик.

Тестирование состоит из нескольких частей:

- Системное тестирование, выполняется внутри организации разработчика
- Альфа и Бета тестирование – выполняется пользователем под контролем разработчика. Альфа выполняется в среде разработчика, а бета в среде пользователя.
- Приемочное тестирование – выполняется пользователем без контроля разработчика.

Системное тестирование обычно производится от простых сценариев к сложным. Первыми тестируются заявленные возможности ПО. Используются те же сценарии, что и в функциональнov тесте, но тестируется вся система на корректность реализаций функций.
Постепенно сценарии усложняются, проверяется стабильность работы системы при нескольких пользователях и нескольких запросах от 1 пользователя. Затем проверяется устойчивость к сбоям, по принципу негативного сценария. Затем проверяется совместимость. Затем находят максимальную нагрузку системы.
CARAT - Capacity, Accuracy, Response time, Availability, Throughput

$$Availability = \frac{MTBF - MTTR}{MTBF}$$

, где

MTBF — mean time before failure — среднее время до отказа,

MTTR — mean time to recover — среднее время до восстановления.

Пример инструмента нагруз. тестирования: Apache Jmeter.

64.Тестирование системы в целом. Альфа- и бета-тестирование.

Тестирование системы в целом начинается после окончания интеграции. На этом этапе нужно провести проверку заявленных характеристик.
Тестирование состоит из нескольких частей:

- Системное тестирование, выполняется внутри организации разработчика
- Альфа и Бета тестирование – выполняется пользователем под контролем разработчика. Альфа выполняется в среде разработчика, а бета в среде пользователя.
- Приемочное тестирование – выполняется пользователем без контроля разработчика.

65.Аспекты быстродействия системы. Влияние средств измерения на результаты.

1. Системный и архитектурный аспект.
   Архитектура может быть: распределенной, где требуется баланс производительности на всех уровнях обработки информации, кластерной, виртуализованной.
2. Низкоуровневый аппаратный аспект
   Техническими характеристиками аппаратуры. Пример: от тактовой частоты зависит линейная скорость выполнения каждого потока программы.
3. Программный аспект
   выбор подходящих алгоритмов для разрабатываемых программ. Зачастую сроки важнее оптимальности решения задач. Пример: Qsort и пузырек.
4. Человеческий фактор

Во время **анализа** сначала выбираются критерии оценки, после этого выбираются средства измерения, бывают неинтрузивные (не оказывают влияние на результаты),

интрузивные(оказывают влияние на результаты) и слабо интрузивные. В вычислительных системах любое измерение влияет на результат. Далее необходимо выбрать нагрузку и нагрузить систему. Нагрузку необходимо приблизить к пользовательской. В конце анализ и изменения основываясь на результатах. Повторять до полного удовлетворения.

66.Ключевые характеристики производительности.

Параметры связаны между собой.

Время отклика системы (latency) – измеряется от выдачи запроса до получения первых результатов.

Полное время обслуживания - до всех результатов

Пропускная способность – показатель максимального числа запросов через канал ввода-вывода, систему, узел.

Утилизация ресурса (%util) – показывает долю времени полезной работы

Занятость (%busy) – время работы 1 процесса.

Ожидание (%wait) – сколько времени не пуста очередь.

Точка насыщения - точка достижения максимальной производительности.

Масштабируемость показывает на сколько еще можно нагрузить систему.

Эффективность - это отношение полезный работы ко всей.

Еще есть **ускорение** и **прирост производительности** - насколько больше полезной работы после изменений.

В JAVA – скорость запуска и memory footprint (стратегии использования памяти)

67.Нисходящий метод поиска узких мест.

Классический нисходящий метод поиска узких мест последовательно рассматривает систему от более общих компонент к более частным.

1. **Исключение ошибок аппаратуры и администратора**.Проверяются системные журналы, файлы конфигурации системы и прикладного ПО.
2. **Общесистемный и межузловой мониторинг.** После исключения ошибок начинается наблюдение за системой. Операционная система обладает достаточно развитыми средствами наблюдения за своими подсистемами. Средства системного межузлового мониторинга помогают наблюдать общую картину работы приложения.
3. **Мониторинг приложения.** Следующим этапом является наблюдение за самим приложением. Фиксируются алгоритмические проблемы, проблемы API, проблемы, связанные с многопоточностью, блокировками и синхронизацией. Используются средства мониторинга и профилирования приложений.
4. **Мониторинг микроархитектуры.** Если полученного в ходе предыдущего испытания прироста недостаточно, происходит переход к мониторингу микроархитектуры. (Выравнивание данных, оптимизация кэшей, пузырьки конвейера, предсказание переходов.)

68.Пирамида памяти и ее влияние на производительность.

Сверху пирамиды находится самая быстрая, дорогая и маленькая память. Короче, чем ниже память, тем дольше обращение к ней. И нужно минимизировать кол-во обращений. Стоит помнить про локальность памяти.

	Объем	Тд	*	Тип	Управл.
CPU	100-1000 б.	<1нс	1с	Регистр	компилятор
L1 Cache	32-128Кб	1-4нс	4с	Ассоц.	аппаратура
L2-L3 Cache	0.5-32Мб	8-20нс	30с	Ассоц.	аппаратура
Основная память	0.5Гб-4ТБ	60-200нс	3-10м	Адресная	malloc,free slab-alloc
SSD	128Гб-1Тб/drive	25-250мкс	5д	Блочн.	open read/write
Жесткие диски	0.5Тб-4Тб/drive	5-20мс	4мес	Блочн.	open read/write
Магнитные ленты	1-6Тб/к	1-240с	200л	Последов.	программно

NB: при 2% cache miss производительность падает ниже 60% от идеальной!

69.Мониторинг производительности: процессы.

Процесс или поток внутри процесса может находится в 3 состояниях:

- Runnable - готов к исполнению
- On CPU User/Sys - находится на исполняющем устройстве
- Wait for smth - ожидать ввода/вывода, освобождения блокировки, при этом не занимая ресурсы процессора

Если процесс отработал свой квант времени и нашелся более приоритетный процесс - то происходит Context Switch (переход к другому процессу, сохраняя окружение предыдущего). Старый процесс переходит в состояние Runnable.
Context Switch может произойти, когда процесс запросил ввод/вывод.

Если процесс находится в ожидании чего-либо, то при окончании ожидания он переходит в состояние Runnable.

Если процесс не отработал свой квант времени, находится в состоянии On CPU и нашелся процесс с более высоким приоритетом, то произойдет вытеснение процесса(или несвободное переключение контекста).

70.Мониторинг производительности: виртуальная память.

Виртуальная память – метод управления память, позволяющий выполнять программы, требующие больше оперативной памяти, чем имеется в компьютере, через автоматическое перемещение частей программы между основной памятью и хранилищем.
В работе виртуальной памяти задействованы физическая память, устройство подкачки и виртуальные страницы, которые могут принадлежать либо физической, либо виртуальной памяти. Каждая страница может быть в памяти, на swap-устройстве или быть зарезервированной. Весь процесс поделен на страницы от 4 кб. Страницы, связанные с памятью, называются *именованной памятью*, а созданные в динамической куче – *анонимные*.
С виртуальной памятью связано несколько параметров. *Scan rate* – число отсканированных страниц за единицу времени. Если эта величина высокая, значит не хватает оперативки.

71.Мониторинг производительности: буферизированный файловый ввод-вывод.

При чтении данных сначала указывается количество байт, которые надо прочитать. Интенсивность чтения определяется требованиями программы.

Алгоритм буферного IO:

1. в программе обращение к функции из системной библиотеки
2. формируется запрос к ядру и вызывается соответствующая функция ядра
3. I/O в ядре в подсистеме VFS, запрос попадает на уровень, не связанный с конкретной файловой системой
4. имя файла преобразуется в DNLS в inode файла
5. работа с буферным кешем для экономии ресурсов:
    - данные содержатся в виде блоков данных файла в ОЗУ - промежуточное хранилище
    - каждые (30 с) данные измененные сохраняются на диск
6. ниже VFS - реализация модулей ядра конкретной файловой системы. К точке монтирования подключается устройство с использованием его специального файла.
7. работает драйвер. на уровне аппаратного интерфейса данные перемещаются на диск.

Для вырожденного типа обмена с дисковой подсистемой различают 2 типа доступа: случайный и последовательный. В случайной каждый запрос обращается к новому месту диска, а в последовательном данные записываются/читаются большими групповыми последовательностями. Для повышения скорости в случайном типе — переходить на SSD.

Мониторинг характеристик I/O осуществляется с помощью *iostat*. Кол-во чтений/записей, объем чтения/записи, среднее время чтения/записи, время ожидания,

время обслуживания и процент занятости устройства (util).

72.Мониторинг производительности: Windows и Linux.

В различных операционных системах существуют разные средства мониторинга.

В Windows самым распространённым является стандартное средство системного мониторинга Task Manager, где, помимо всего прочего, выводится статистика использования
процессора, памяти, дисковой подсистемы и др.

Для более детального исследования поставляются также специальные системные оснастки,
такие, как Resource Monitor & Performance Monitor, Reliability Monitor и т.д.

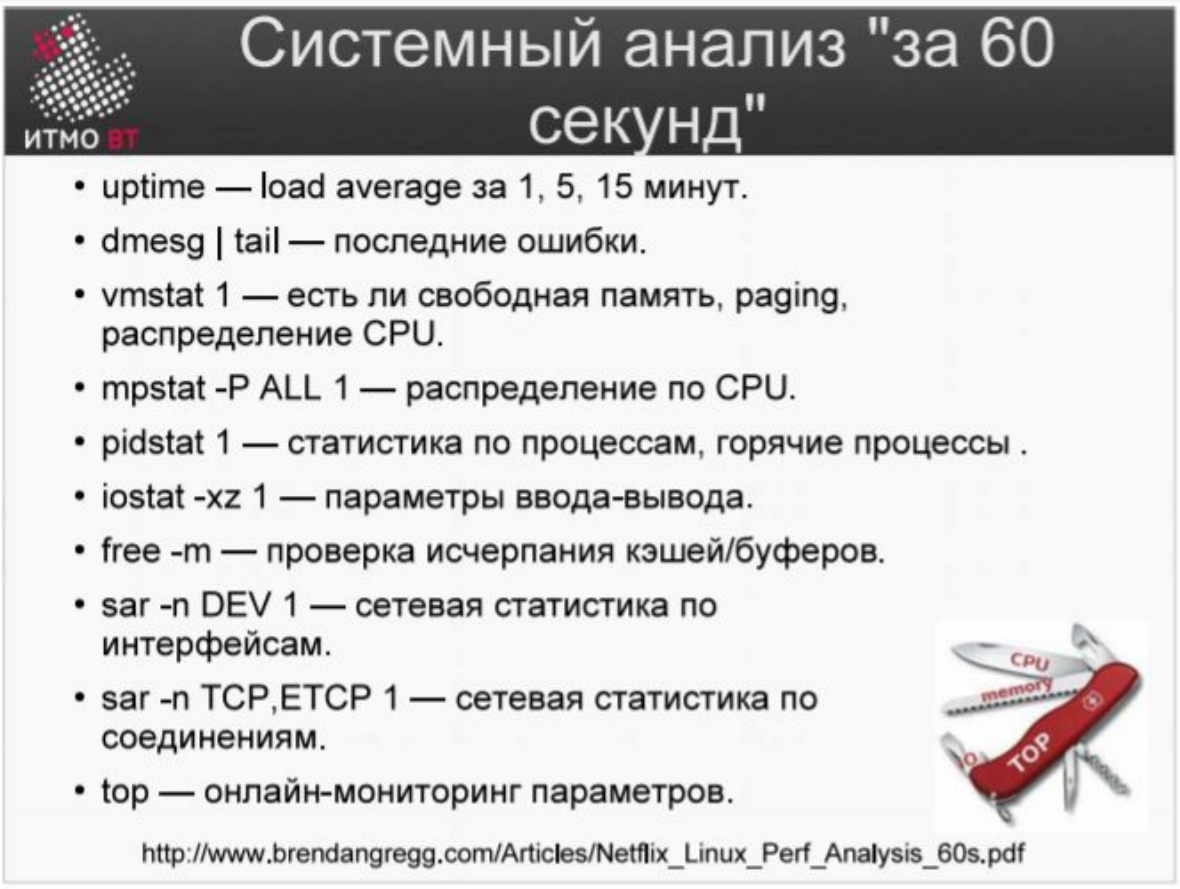
Linux

много средств мониторинга (в основном, неинтрузивные — не влияющие на результат исполнения программы).

- **top**: динамически наблюдать характеристики запущенных процессов (текущий приоритет PR, занимаемая память (VIRT - общий размер адресного пространства процесса, RES - размер в физической памяти, SHR - в shared с другими процессами)
- **sar**: системная информация ядра
- **pref**: характеристики ядра или запущенной под perf программы, может собирать промахи мимо кеша.
- **strace**: наблюдение за процессом, трассировка сист вызовов (процесс выполняет к ядру или сист библиотекам) - интрузивен
- **system tap (stap)**: установка точек сбора информации в ядре (информация о подсистемах ядра), использует своя ЯП, слабо интрузивен.

73.Системный анализ Linux "за 60 секунд".

.



74.Создание тестовой нагрузки и нагрузчики.

Для тестовой системы, которая является полной копией реальной, нужно иметь возможность создавать нагрузку, близкую по характеристикам к реальной пользовательской. Кроме того, в тестовых системах есть возможность использовать интрузивные средства мониторинга.

Обычно, такого рода нагрузку создают с помощью средств создания синтетической нагрузки. Синтетическая нагрузка всегда будет отличаться от реальной, и для ее создания используется большое число параметров для гибкой настройки.

Для Java, наиболее популярным инструментом нагрузочного тестирования является Apache JMeter.

75.Профилирование приложений. Основные подходы.

С помощью профилирования можно узнать время исполнения функции, объем созданных объектов, проследит за потоками приложений и борьбой потоков за захват блокировки.

Существует два основных подхода:

1. диагностические точки можно внедрять в сами функции из указанного набора. Для данного способа для начала следует определить проблемные места.
2. использование прерываний программы с заданной периодичностью, Профилировщик прерывает программу и собирает интересующую программу. Собирается инфа о состоянии стека и кучи. Интервал нужно выбирать оптимально.

76.Компромиссы (trade-offs) в производительности.

Борьба за производительность сопряжена с компромиссами.

- Различные подсистемы ЭВМ взаимозависимы и изменения в одном месте неизбежно приведут к изменениям в другом.
    - Например, чем быстрее мы хотим получить доступ к данным, тем больше памяти нам потребуется: linear search в массиве vs indexing
    - Другой пример — блочный доступ к диску с кэшированием блоков данных в ядре. Больше кэша — быстрее чтение-запись, но меньше памяти для других задач.
- Выбор алгоритма и архитектуры может ускорить работу приложения во много раз.

77.Рецепты повышения производительности при высоком %SYS.

Высокая загруженность CPU задачами уровня ядра

1) высокая нагрузка на подсистему ввода-вывода, нужно реже читать/писать, сжимать данные, буферизация или замена устройств на более быстрые
2) Недостатки в работе планировщика, частая диспетчеризация, нужно проверить не слишком ли много потоков и что делает ОС
3) Избыточная подкачка страниц. Нужно выдать больше памяти системе, меньше процессам. Либо запретить выгрузку важных процессов из памяти.
4) Трата времени процессора в других системных функциях. Нужно найти и исключить лишние функции и настроить параметры ядра.

78.Рецепты повышения производительности при высоком %IO wait.

Высокое время ожидания CPU

1) проблемы, связанные с самими приложениями. Следует оптимизировать запросы к диску. Можно ввести объем большими порциями за 1 операцию.
2) Буфера/Кэши, нужно расширить память для промежуточного использования и настроить кэши.
3) Проблемы с аппаратурой, нужно купить более совершенную дисковую систему. Эффективны ССД - накопители

79.Рецепты повышения производительности при высоком %Idle.

При высоком времени простоя ЦПУ может быть мало процессов на стадии выполнения.

1) распараллелить алгоритмы в приложении
2) проанализировать блокировки, например, когда много потоков пытаются завладеть одним и тем же участком кода
3) держать блокировку как можно меньше и делать ее более легкой
4) добавить потоки в пулы приложений.
5) использовать Lock-Free алгоритмы

Проблемы могут корениться в самой ОС, это связано с дефектами в ОС на системных блокировках. Нужно промониторить их. Также может помочь настройка подсистем ядра.

80.Рецепты повышения производительности при высоком %User.

При высокой загруженности процессора нужно воспользоваться средствами профилирования, они помогут найти самые затратные функции.

1) использовать алгоритмы меньшей сложности
2) использовать большие объекты и структуры повторно, так в линуксе процесс сначала попадает в состояние зомби и помещается в death row и когда нужен новый процесс, то может восстать
3) на уровне микроархитектуры важно избавиться от кэш-промахов и промахов мимо TLB. Кэш промахи можно исправить работой над структурой данных, а мимо TLB - использованием больших страниц памяти.