1 вопрос

Введение в технологии программирования. Парадигмы программирования.
Технология программирования - совокупность методов средств и процедур, используемых в процессе разработки ПО. При этом все существующие средства разработки можно разделить на две основные категории: языки программирования высокого уровня и низкого.
При разработке ПО определяют парадигму программирования.
Парадигма программирования - способ создания программ с помощью определенных принципов и подходящего языка, позволяющего писать ясные программы:
Структурное - методология разработки ПО, в основе лежит представление программы в виде иерархической структуры. Вся программа разбивается на отдельные блоки, элементарными блоками являются: последовательность, ветвление, циклический блок.
Функциональное - в которой любой процесс вычисления трактуется как вычисление значения функции в ее математическом понимании. Вычисление ведется от исходных данных, которые в свою очередь могут являться результатами работы предыдущих функций. Функция будет запущена тогда и только тогда, когда на ее входе будет иметься полный набор исходных данных. Порядок выполнения функции определяется порядком прихода аргументов. Представитель: lisp.
Логическое программирование - парадигма, основанная на автоматическом доказательстве теорем на базе заданных фактов и правил вывода. Оно основано на аппарате математической логики. Язык- пролог.
Автоматное программирование - при использовании которой программа или ее фрагмент определяется как модель формального автомата. В зависимости от сложности задачи в автоматном программирование могут использоваться конечные автоматы или автоматы более сложной структуры
ООП - основные компоненты :объект и класс, а программа представляет собой процесс взаимодействия объектов, функционирующих на основе имеющихся у них методов, обладающих поведением и возможностью обмена сообщениями. На базе ООП может быть построено прототипное программирование. В этом случае отсутствует понятие класса, а наследование определяется как механизм клонирования или создания прототипов, то есть полной копии уже существующего в программе объекта. Программирование на основе шаблонов
СОП - при которой выполнение программы определяется срабатыванием некоторых событий, генерируемых пользователем либо другими элементами системы. События представляют собой потоки, организующие взаимодействие между компонентами программы с помощью исполнительных механизмов.
Агентно-ориентированное программирование - парадигма, в которой основная концепция - работа параллельных независимых агентов. Для каждого агента определяется поведение, которое зависит от среды, в которую агент помещается в текущий момент времени. Агент воспринимает среду через набор датчиков и регулирует собственное поведение в зависимости от полученное информации за счет собственных исполнительных организмов.

2 вопрос. Основные понятия и подходы

Под технологией понимают набор методов средств и процедур необходимых для нормального процесса проектирования программного обеспечения.

Под методами понимают совокупность следующих задач:

- Планирование и оценка проекта
- Анализ системных и программных требований

- Проектирование алгоритмов, структур данных и программных структур, кодирование, тестирование и сопровождение

Средства обеспечивают автоматизированную или автоматическую поддержку методов. Процедуры служат интерфейсом с целью конечной реализацией проекта. Процедуры определяют порядок определения методов, контроль обеспечивающих качество и координацию изменений, а так же возможность ведения версий.

Как любой процесс. Процесс разработки ПО представляет собой набор технологических операций. Всякая технологическая операция преобразует входные данные, которыми являются документы или рабочие материалы в требуемую результативную форму. Для этого в качестве управления используются методические материалы, нормативные документы и стандарты в качестве механизма используются исполнительные программные и технические средства

Типы программ

Технология разработки ПО должна охватывать различные типы программ. Выделяют следующие типы:

- Автономное По – устанавливается на одиночный компьютер, не связано с другим программным обеспечение
- Встроеное ПО – часть приложения, которое работает на конкретной аппаратной платформе.
- ПО реального времени – должно работать в реальном времени, т.е. время отклика минимально
- Сетевое ПО – сеть между двумя частями

Технология программирования должна поддерживать ПО на протяжений всего его жизненного цикла. Жизненный цикл ПО это промежуток времени от необходимости создания до момента завершения сопровождения продукта. Состав процессов жизненного цикла регламентируется международным стандартом. Данный стандарт описывает структура и процессы жизненного цикла

3 основных типы:

1) Основные процессы.
    a. Приобретение
    b. Поставка
    c. Разработка
    d. Эксплуатация
    e. Сопровождение
2) Вспомогательные процессы. Обеспечивают инфраструктуры процесса
    a. Документирование
    b. Управление конфигурацией
    c. Обеспечение качества
    d. Верификация
    e. Аттестация
    f. Аудит

g. Разрешение проблем

3) Организационные процессы.
    a. Управление
    b. Усовершенствование
    c. Создание инфраструктуры
    d. Обучение

Процесс разработки отличается в зависимости от выбранной стратегии конструирования.

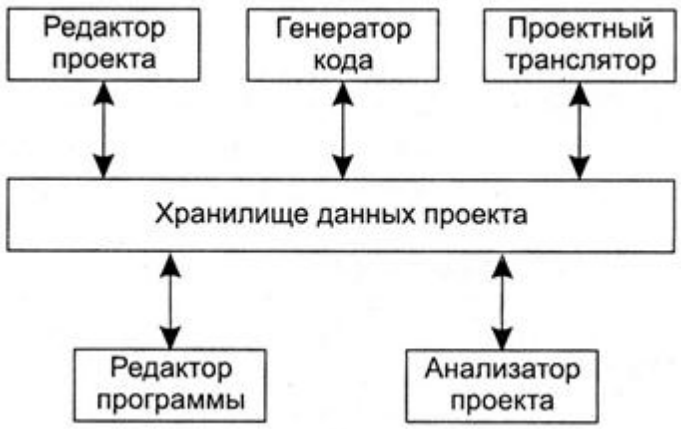
Существуют 3 стратегии:

1) Однократный проход (Водопадная стратегия). Предполагает линейную последовательность этапов конструирования
2) Инкрементная стратегия. Предполагает определение пользовательских требований в целом, разбиение на этапы и реализации в виде последовательных версий
3) Эволюционная стратегия. Система создается как набор версий, однако в каждой версии исправляются предыдущие требования

Модели системного структурирования:

- Модель хранилища данных;

В модели хранилища данных подсистемы разделяют данные, находящиеся в общей памяти. Как правило, данные образуют БД. Предусматривается система управления этой базой.



- Модель клиент-сервер;

Модель клиент-сервер используется для распределенных систем, где данные распределены по серверам. Для передачи данных применяют сетевой протокол, например, TCP/IP.

- Трехуровневая модель;

Трехуровневая модель является развитием модели клиент-сервер.

Уровень графического интерфейса пользователя запускается на машине клиента. Бизнес-логику образуют модули, осуществляющие функциональные обязанности системы. Этот уровень запускается на сервере приложения. Реляционная СУБД хранит данные, требуемые уровню бизнес-логики. Этот уровень запускается на втором сервере – сервере базы данных.

Преимущества трехуровневой модели:

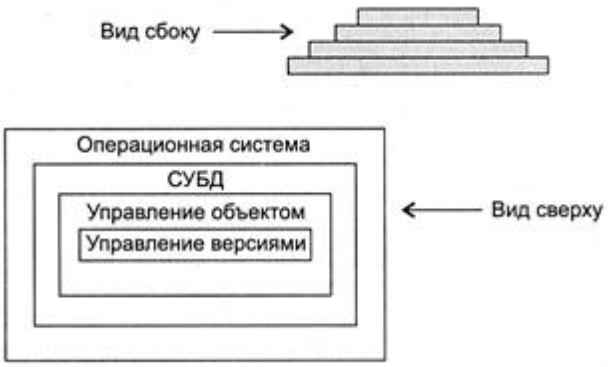
- упрощается такая модификация уровня, которая не влияет на другие уровни;

- отделение прикладных функций от функций управления БД упрощает оптимизацию всей системы.

- Модель абстрактной машины.

Модель абстрактной машины отображает многослойную систему.

Каждый текущий слой реализуется с использованием средств, обеспечиваемых слоем-фундаментом.

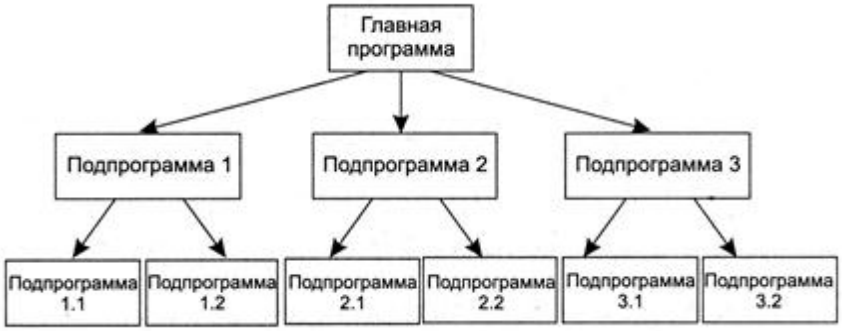


Модели управления:

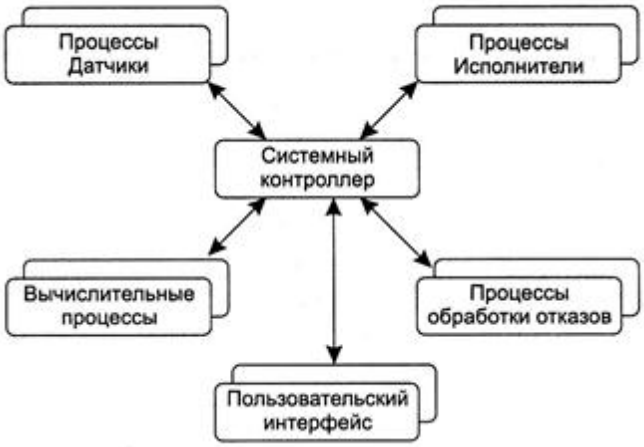
- Модель централизованного управления;

В модели централизованного управления одна подсистема выделяется как системный контроллер. Ее обязанности – руководить работой других подсистеме.

Различают две разновидности моделей централизованного управления: модель вызов-возврат и модель менеджера, которая используется в системах параллельной обработки.



Модель вызов - возврат



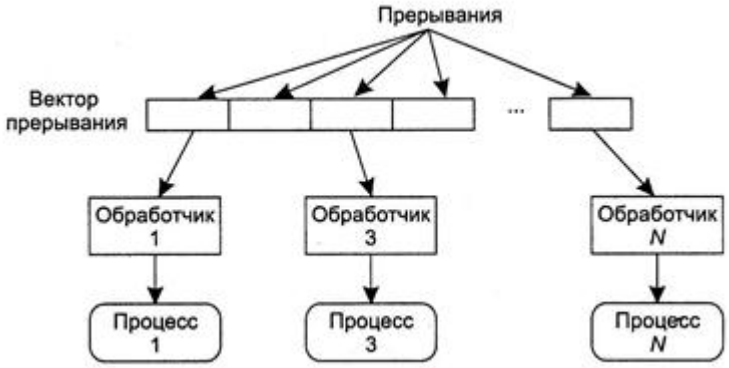
Модель менеджера

- Модель событийного управления.

В модели событийного управления системой управляют внешние события. Используются две разновидности модели событийного управления: широковещательная модель и модель, управляемая прерываниями.

В широковещательной модели каждая подсистема уведомляет обработчика о своем интересе к конкретным событиям. Когда событие происходит, обработчик пересылает его подсистеме, которая может обработать это событие. Функции управления в обработчик не встраиваются.

В модели, управляемой прерываниями, все прерывания разбиты на группы – типы, которые образуют вектор прерываний. Для каждого типа прерывания есть свой обработчик. Каждый обработчик реагирует на свой тип прерывания и запускает свой процесс.



Модульная декомпозиция

После этапа разработки системной структуры в процессе проектирования следует этап декомпозиции подсистем на модули.

Здесь рассматриваются две модели, используемые на этапе модульной декомпозиции подсистем.

- Объектно-ориентированная модель.

Система состоит из набора взаимодействующих объектов. Модули представляют собой объекты с собственными состояниями и определенными операциями над этими состояниями.

- Модель потоков данных.

Система состоит из функциональных модулей, которые получают на входе данные и преобразуют их некоторым образом в выходные данные. Такой подход часто называется конвейерным.

По возможности разработчикам не стоит принимать поспешных решений о том, будет ли система параллельной или последовательной. Проектирование последовательной системы имеет ряд преимуществ: последовательные программы легче проектировать, реализовать, проверять и тестировать, чем параллельные системы, где очень сложно формализовать, управлять и проверять временные зависимости между процессами. Лучше сначала разбить систему на модули, а на этапе реализации решить, как организовать их выполнение – последовательно или параллельно.

3 вопрос

Каждой стратегии соотвествует собственная модель.

Водопадная модель:

1) Системный анализ. На этом этапе определяется набор элементов компьютерной системыи взаимосвязь друг с другом. Формируется интерфейс между аппаратной и программной части. Формируется план-график проекта
2) **Анализ требований.** Предполагает изучение предметной области, уточнение и детализация функций, определение характеристик и интерфейсов. Результатом анализа требования является ТЗ. С обоснованием актуальности и необходимости разработки.
3) **Проектирование.** Состоит в разработке архитектуры ПО, модульной структуры, структуры данных, алгоритмических структур, входных и выходных интерфейсов.
4) **Кодирование.** Перевод с естественного языка на выбранный ЯП
5) **Тестирование.** Проверка правильности и корректности работы разрабатываемых функциональных блоков. Возможно выполняется функциональное тестирование и структурное тестирование.
6) **Сопровождение.**

Особенностью водопадной модели, что каждый следующий этап начинается после завершения и документирования предыдущего этапа. Любые ошибки и неточности приводят к необходимости возврата к начальным стадиям разработки.

Для снятия недостатков водопадной модели используется макетирование (прототипирование) – процесс создания модели требуемого ПО.

- Макет на основе компьютера.
- Работающий макет
- Предъявление аналога

Макетирование выполняется на этапе анализа требования и включает следующие этапы:

1) Выявление ожиданий
2) Построение макета
3) Оценка макета заказчиком

Спиральная модель БОЭМА

Данная модель предполагает разбиение проектирование ПО на 4 взаимосвязанных этапа

1) Планирование. Выполняется постановка задачи, цели, ограничения

2) Анализ риска. Этап проектирования, выполняется анализ вариантов решений и распознавание возможности реализаций
3) Конструирование. Этап реализации. Разрабатывается ПО следующего уровня
4) Тестирование

Инкрементальная модель

Основана на RUP технологии. Предполагает разбиение процесса создания ПО на набор инкрементов. Каждый инкремент заканчивается получением новой версии ПО. Основной особенностью является то, что все требования к системе предъявляются на начальном этапе. Каждый инкремент реализует часть требований с последующим объединением.

Этапы:

1) Анализ
2) Проектирование
3) Кодирование
4) Тестирование

Модель быстрой разработки приложений. RAD

Предполагает разработку ПО в экстремально короткий цикл. RAD является высокоскоростной адаптацией водопадной модели реализуемой за счет использование компонент-ориентированного подхода. Она позволяет создать полностью функционирующую систему за 60-90 дней. Ориентирован на разработку инфо-систем стандартных структур.

Этапы:

1) Бизнес-моделирование. Моделируются потоки между бизнес-функциями. Разрешаются вопросы получения информации.
2) Моделирование данных. Определяются информационные потоки, которые отображаются в физические наборы данных
3) Моделирование обработки. Определяется преобразование объекта данных с целью обеспечивание требуемых бизнес-процессов.
4) Генерация приложения. Предполагается использование методов ориентированных на языки 4-го поколения ЯП.
5) Тестирование и объединение частей.

Экстремальное программирование. eXtreme Programming (XP)

Основано на эволюционном процессе разработки. Основная идея: устранить высокую стоимость изменений характерную для приложений с используемых шаблонов и приложений работающих с базами данных. В жизненном цикле XP выделяют 4 базовых действия:

1) Кодирование
2) Тестирование
3) Выслушивание заказчика
4) Проектирование

XP называют живой разработкой. Данный процесс предполагает, что команда разработчиков постоянно взаимодействует с заказчиком, который входит в команду разработки. Команда

размещается в открытом офисе и предполагает коллективное владение кодом. Разработка программы ведется от тестирования к кодированию функционала.

Каждый функциональный блок разрабатывается двумя программистами.

40 часовая рабочая неделя.

Проектирование на основе метафор и архитектурных выбросов. Метафора – описание функционала на простом языке с последующей оценки механизмов работы. На основе метафор и архитектурного выброса планируется дальнейшая реализация. Для каждой реализации выполняется оптимистичная оценка проекта и неопределенная. Для следующей итерации берутся отличные оценки.

Первое решение самое верное.

Важнее разработанная программа чем документация к ней.

Использование при написаний кода внутренних корпоративных стандартов.

Классификация программных продуктов

Программные продукты:

- Системные. ОС, оболочки, утилиты
- Прикладные.
    - Для разработчиков программ. Среды разработки, отладка, системы программирования
    - Для пользователей. Общего назначения, профессиональные, САПРы, обучающие, развлекающие.
- Гибридные. Автоматизированные системы управления.

4 вопрос

Основными требованиями к программному продукту являются эксплуатационные требования.

Они определяют характеристики разрабатываемого ПО, предъявляемого в процессе его функционирования.

Требования:

- Правильность. Т.е. функционирование в соответствии с ТЗ
- Универсальность. Обеспечение правильной работы ПО при любых допустимых входных данных, и защита от неправильных
- Надежность. Полная повторяемость результата даже в случае различного рода сбоев.
- Проверяемость. Возможность проверки получаемых результатов
- Точность. Получение результата с погрешностью не превышающие допустимую
- Защищенность. Обеспечение конфиденциальности информации
- Программная и аппаратная совместимость. Совместимость со всем
- Эффективность. Использование минимальное количество ресурсов, в частности технических средств для достижения результат

- Адаптируемость. Возможность быстрой модификации с целью приспособления к изменяющимся внешним условиям
- Повторная входимость. Возможность повторного запуска без перезагрузки системы
- Реентерабельность. Возможность параллельного использования несколькими процессами
- Рентабельность. Определяет, что задача, решаемая с помощью компьютера гораздо выгоднее, чем без него

Сложность многих программ не позволяет сразу определить четкие требования к ним. Чаще всего для перехода от идеи создания до ее реализации необходимо выполнить исследование предметной области, результаты которого заносятся в ТЗ. Техническое задание является начальным документов для разработки, входит в этап пред проектного проектирования и исследования и оформляется в соответствии с гост 19.201. При выполнении пред проектного исследования должны быть преобразованы нечеткие знания в точные требования, при этом разрешается 2 вида неопределенностей:

1) Могут быть неизвестны методы решения формулируемой задачи.
2) Неизвестная структура автоматизированных процессов. Требует досконального изучения существующих бизнес процессов

При снятии неопределенности второго типа:

а) Определить структуру и взаимосвязи

б) Распределить структуру между человеком и системой

в) Четко разделить. Выделить для решения задачи ресурсы

Разработка ТЗ

Это документ, в котором сформулированы основные цели разработки, требования ПО, сроки, этапы разработки, регламентирован процесс испытаний. В разработке ТЗ принимает участие представитель заказчика и исполнителя. Основными факторами, определяющими характеристики исходного ПО:

1) Исходные данные
2) Среда функционирования
3) Взаимодействие с другим ПО

ТЗ включает в себя:

- Введение. Наименование, краткая характеристика продукта, область применения, основное назначение. Должно быть доказано, что разрабатывать данный продукт необходимо. Для обоснования актуальности может быть приложен документ с информацией о аналогах и прототипах, создаваемой системы, с указанием причины несостоятельности аналогов.
- Основания для разработки. Определяются документы на основании которых ведется разработка, указывается предприятие, создавшее документы.
- Назначение разработки. В данном разделе указывается, какие функциональные и эксплуатационные характеристики, выгодные для пользователя.
- Требования к программному изделию.

    - Требование к функциональным характеристикам. Должны быть указаны все функции, состав и характеристики данных и результатов. Могут быть определены критерии эффективности.
    - Требования к надежности. Требуется указать как ведет себя система в нештатных ситуациях.
    - Требования к условиям эксплуатации.
    - Требования к составу и параметрам технических средств. Необходимы технические средства, требования по памяти, частоте и т.д.
    - Требования к программной совместимости. Указывается язык программирования, среда программирования, требования к ОС, требования к пользовательскому интерфейсу, требования по защите информации.
- Требования к программной документации. Обязаны определить перечень документов, с которыми будет поставляться программа.
- Технико-экономические показатели.
- Стадии и этапы разработки. Разрабатывается календарный график.
- Порядок контроля и приемки. Указываются виды проводимых испытаний и общие требования

Особенности этапа проектирования

Проектирование – итерационный процесс при помощи которого требования к ПО транслируются в инженерное представление программной системы. Сначала выполняется концептуальное проектирование с последующим уточнением, позволяющие сформировать этапы. Обычно проектирование выполняется в несколько ступеней:

- Предварительное проектирование. Позволяет сформировать архитектуру программы и определить набор обрабатываемых данных.
- На этапе детального проектирования, наборы обрабатываемых данных преобразуются в структуру данных, а описанные в архитектуре командные циклы в алгоритм работы программы.

По архитектуре ПО классифицируется:

- Однопользовательское ПО.
    - Программы. В любой непонятной ситуации пиши программное обеспечение
    - Пакеты программ.
    - Программные комплексы.
    - Программные системы.
- Многопользовательское ПО.
    - Система клиент-сервер

Программные архитектуры различаются в зависимости от используемого пользовательского интерфейса:

- Примитивный – один сценарий работы
- Меню – реализует множество сценариев работы, иерархическая цепочка

- Со свободной навигацией. Множество интерфейсов.
- Интерфейс прямого манипулирования. Все действия над графическими объектами

Стандарты:

- Проектирования
    - Конфигурация рабочих мест
    - Фиксация решений
    - Набор моделей
    - Совмещение работы
- Оформление документации
    - Комплектность
    - Согласование и утверждение
- Интерфейс пользователя
    - Оформление экранов
    - Оформление текстов помощи
    - Перечень стандартных сообщений

На этапе детального проектирования осуществляется 3 основных вида деятельности:

- Выполнение структурирование системы.
    - Модель хранения данных. В соответствии с данной моделью, разрабатываемое ПО предполагает набор подсистем, которое разделяет и обрабатывает данные и размещает в некоторой памяти. Чаще всего такие данные организуются в виде БД, а подсистема обеспечивает манипуляцию.
    - Модель клиент-сервер. Реализуется система, где данные могут располагаться в различных подсистемах, связь между подсистемами осуществляется при помощи локальной сети. Конфликты по доступу решаются либо сервером, либо доступом к сети.
    - Модель абстрактной машины. Представляется в виде многослойной структуры, каждый уровень которого, использует средства нижестоящего слоя.
- Моделирование управления системы
    - Модель централизованного управления. Одна из подсистем выступает в качестве системного контроллера и в ее обязанности входит управление работы другими подсистемами. Различают 2 модели:
        - Модель «Вызов-возврат». Есть основная программа, она вызывает подпрограммы, возврат осуществляется, когда завершается подпрограмма.
        - Модель «Системный контроллер». Предполагает, что существует программа, которая в любой момент времени может вызвать какой-либо из процессов системы. Процессы взаимодействовать не могут, только через системный контроллер.

- Модель событийного управления. Предполагается, что любая система работает на основании возникающих внешних событий.
    - Широковещательная модель. Работает на основе системы очереди.
    - Модель управления прерываниями. Относительно вектора прерывания, выбирается обработчик прерывания. Все события сопоставляются определенному прерыванию. Все прерывания разбиваются на типы или группы. Типы прерываний образуют вектор прерываний. В векторе для каждого прерывания имеющего особый код ставится соответствующий обработчик.

- Декомпозиция системы на подсистемы. 1 – модель потоков данных. В основе лежит разбиение модуля по функциям. 2 – модель объектов основана на сцеплении сущностей, имеющих собственные наборы данных и наборы операций.

5 вопрос

Программный модуль – любой фрагмент описания процесса, который оформляется как самостоятельный программный продукт, пригодный для дальнейшего использования. Один и тот же модуль может входить в состав нескольких программ. В этой связи модуль рассматривается как средство борьбы со сложностью программ.

Модульность – свойство системы подвергаться декомпозиции на ряд внутренне связанных и слабо зависящих друг от друга модулей. Оптимальный модуль должен удовлетворять 2м критериям: снаружи он должен быть проще чем изнутри, и модуль должно быть проще использовать, чем построить.

Основной характеристикой модуля является информационная закрытость. Предполагает, что модуль внутри должен быть сложнее чем снаружи. Пользователю модуля должны быть известны подробности доступа функции модуля без подробностей их реализации. Модуль состоит из двух частей:

1) Интерфейс. Должна доступна пользователю. То, что известно снаружи.
2) Внутренняя структура модуля. Должна быть скрыта

Для организации связи между модулями и оценки степени их связи существует понятие связность. Связность – внутренняя характеристика модуля, определяющая мера зависимости ее частей. Чем выше связность, тем лучше результат проектирования. Для измерения связности используют понятия Сила Связности(СС). Существуют 7 сил связности. Каждая из них имеет вес, чем больше вес – тем больше связность.

Степени связности:

1) Связность по совпадению (0). Присутствует в модуле, где отсутствуют явные связи. Существуют, когда в программной системе изредка вызывают функции друг друга
2) Логическая связность (1). Возникает в модуле, где все функции взаимосвязаны на одной и той же операции.
3) Временная связность (3). Части модуля могут быть не связаны, но выполняются в одно и то же время.
4) Процедурная связность (5). Части модуля связаны порядком выполнения действий. Реализует некоторый сценарий поведения. Все связаны общим процессом

5) Коммуникативная связность (7). Части модуля связанны по данным, обрабатывают одну и туже структуру. Довольно технологично, так как в случае изменения структуры меняется один модуль.

6) Последовательная связность (9). Все функции модуля выполняются друг за другом. Выходные данной одной части используются как входные части другой части модуля.

7) Функциональная связность (10). Части модуля вместе реализуют одну функцию, без привлечения внешних обработчиков. Функционально связанный модуль содержит элементы, участвующие в выполнении одной и только одной проблемной задачи.

Сцепление модулей

Мера взаимосвязи модулей, их внешняя характеристика. Чем меньше сцепление, тем лучше.

Выделяют 6 типов сцепления (каждый из них имеет собственный вес):

1) Сцепление по данным(1). Предполагает, что модуль А вызывает модуль В, при этом все входные и выходные параметры вызываемого модуля — это простые элементы данных (скаляры).

2) Сцепление по образцу(3). В качестве данных возвращается структура данных.

3) Сцепление по управлению(6). Предполагает, что модуль А явно управляет функционалом модуля В. Решается установкой флагов/переключателей.

4) Сцепление по внешним ссылкам(5). Есть некий модуль А и В, они работают с общей областью.

5) Сцепление по общей области(7). В одном случае скаляр, в другом структура.

6) Сцепление по содержимому. Когда один модуль ссылается на содержимое другого модуля.

Рутинность – независимость модуля от предыдущих обращений к нему. Модуль называют рутинным, если результат обращения к нему зависит только от значения передаваемых в него параметров. Модуль называют зависящим от предыстории (нерутинным), если результат обращения к нему зависит от внутреннего состояния модуля.

6 вопрос

Восходящее проектирование – это проектирование, при котором выполнение процедур Многоуровневый иерархический подход к проектированию позволяет проектировать сложные программные изделия по принципу *сверху — вниз* с позиции назначения и наилучшего решения основной целевой задачи всей системы. Декомпозиция проекта, разделение на иерархические уровни требует некоторых затрат. В целом ресурсы используются более эффективно, чем при отсутствии четкой иерархии, за счет экономного построения и упрощения компонент проекта на каждом уровне.
Иногда основному проектированию сверху — вниз сопутствует разработка компонент проекта *снизу — вверх.* Разработка начинается от компонента нижнего уровня, далее переходят к разработке компонента следующего уровня иерархии и т.д. Достоинством этого принципа является то, что при переходе к разработке компонентов более высокого уровня иерархии компоненты проекта нижних уровней можно считать готовыми и подключать их к проектируемым компонентам верхнего уровня.
Структурный подход. Предполагает выделение основных функций системы, разработку алгоритмов их реализующих, установку взаимосвязей между потоками данных, циркулирующих в системе.

Структурный подход

Определение спецификации при структурном подхода, предполагает точное описание функций и ограничений разрабатываемого ПО, при этом выделяют функциональную спецификацию и эксплуатационную.

Точность спецификации определяет посредством формальных моделей. Формальные модели делят на 2 группы:

1) Зависящие от подхода.
2) Не зависящие от подхода.

Каждый тип модели целесообразно использовать для своего специфического класса ПО. Методология структурного проектирования и анализа основана на моделировании потока данных и использует представление о проектировании системы в виде комплекта моделей.

Модели:

- Диаграмма потока данных.
- Диаграмма спецификации процессов.
- Словари терминов
- Диаграмма «сущность-связь»
- Диаграммы состояния
- Функциональные диаграммы

При построении моделей структурного подхода изначально определяется концептуальная диаграмма (контекстная) наиболее общим образом описывающая проектирование процессов. На концептуальной диаграмме определяется набор внешних данных, как входных таки выходных и набор воздействий. Выделяется внешние и внутренние сущности системы с минимальным описание передаваемых функций. Затем выполняется постепенная детализация автоматизируемого процесса. Для каждого процесса составляется спецификация процесса. Если процесс использует хранилище данных, то используется «сущность-связь», если управляющим, то диаграмма переходов состояний, для регламентирования используемых в системных терминов используется словарь терминов. Детализация выполняется до того уровня, пока алгоритм процесса не приобретает вид известного и элементарного.

Для каждого детального процесса и концепции может быть составлена диаграмма потока данных. Она описывает взаимодействие источников и приемников информации через процессы системы. Позволяет специфицировать функции. Существует 2 основных нотации для отображения диаграмм:

- Йордана
- Гейна-сарсона

Основными элементами является:

- Внешняя сущность. Материальный объект выступающий в качестве источника или приемника информации.
- Функция или процесс. Выполняет преобразование входных потоков данных с выходными, в соответствии с алгоритмом. Каждый процесс имеет свое имя и номер. Имя для однозначной идентификации, а номера для иерархии
- Поток данных. Определяет дуги, соединяющие между собой процессы и хранилище. Каждый поток имеет имя, соответствующий передаваемой информации.

- Хранилище данных. Определяется как абстрактное устройство для хранения информации. Тип устройства, способ занесения и извлечения информации не регламентируется. Физически в качестве устройства используется БД, файл, картотека и тд.

Контекстная диаграмма состоит из одного элемента процесса и набора элементов внешних сущностей.

Детализация. Правила выполнения детализации. Все входные и выходные воздействия должны сохранятся. Не допускается появления новых воздействий. Необходимо выполнять нумерацию детализированных процессов. Детализация выполняется до тех пор, пока:

1) Процесс взаимодействует не более чем с двумя входными и выходными потоками данных
2) Описание процесса – последовательный алгоритма
3) Процесс выполняет единственную логическую функцию преобразования входной информации в выходную

Для не детализированных процессов составляется спецификация, которая содержит описания логики поведения данного процесса. Для определения спецификации используются следующие методы:

- Псевдокод. Представляет собой формализованное текстовое описание алгоритма, приближенное к языку программирования высокого уровня. Согласно алгоритмическим структурам в псевдокоде определяют конструкции:
    - Следование
    - Ветвление
    - Цикл с пред условием
    - Пост условием
    - С заданным количеством повторений
- Flow-формы. Графическая нотация для описания структурных алгоритмов. Каждый символ Flow-формы определяет одну из возможных вершин, символы могут быть вложены друг в друга. Вершин может быть 6 видов.
- Диаграммы Насси-Шнейдермана. Графическое представление с другими условными обозначениями.

Для каждого процесса составляется словарь терминов – краткое описание понятий для составления спецификации. Словарь нужен для того, чтобы заказчик понимал разработчика.

Термин

Категория

<Краткая категория>

Для описания структуры ПО используют структурные карты. Структурные карты используются на этапе проектирования для демонстрации того как программный продукт будет выполнять системные требования.

Типы карт:

- Структурные карты Константийна. Представляет собой отношение между модулями программы. Узлы соответствуют модулям и областям данных, потоки отображают межмодульные связи.

- Модуль. Реализация\подсистема\библиотека
- Поток определяющий вызов модуля.
- Связь между модулями по данным.
- Связь между модулями по управлению
- Область данных. Содержит только глобальные переменные

- Схема Джексона. Выявляет соответствие между потока данных и выходных данных.
  - Операция
  - Следование
  - Выбор
  - Итерация
- Диаграмма Варение-Орра. Вывод структуры программы и структуры входных потоков исходя из структур выходных данных. Включает в себя физические хранилища информации. Магнитные носители, печатные копии и потоки ввода-вывода
- Hipo. Иерархическая схема описывающая входные данные, процедуры обработки и набор выходных параметров

Диаграмма «сущность-связь». Используется нотация IDEF1. Согласно ей, вводятся понятия:

- Сущность. Представляет собой класс однотипных объектов информация о которых должна учтена в модели. Сущность имеет наименование. Именуются существительным в ед. числе Им. Падеже. Сущность определяет параметры каждого элемента. Параметры определяются набором свойств (атрибуты сущности). Наименование атрибута – существительное. Каждая сущность должна иметь ключ или первичный индекс. Ключ сущности – не избыточный набор атрибутов, значение которых обеспечивает уникальность каждой сущности. Ключевой атрибут пишется первым и подчеркивается. Сущности связаны между собой. Связь – отношение одной сущности к другой или к самой себе. Виды связи:
  - Один-к-одному
  - Каждый элемент связан с несколькими. Один ко многим
  - Много-ко-многим

  Связи могут как быть обязательными, так и не обязательными.

Инфологическая модель данных – средство описания области. Определяет набор хранимых и обрабатываемых объектов. Из предположений инфологической модели определяют сущности на основе которых формируют логическую модель.

Физическая модель определяет правила хранения и правила определения ряда атрибутов. Физическая модель привязывается к будущей и связывается с выбранной СУБД. Необходимо выполнить соответствие имеющих типов. При преобразовании логической модели к физической. Устраняются связи многие ко многим. За счет ведения дополнительных таблиц. В случае использования обязательной связи выполняется транспорт ключевого поля родительской таблицы к дочерней. Такой ключ называют внешним.

Диаграммы состояний определяют набор состояний системы и правило перехода между этими состояниями. Диаграмма состояний реализуется в виде конечного автомата. На этапе анализа требований диграмма состояний демонстрирует поведение системы. Под управляющей информации понимается информация передаваемая системе из вне.

7.Объектно-ориентированная методология. Характеристики объектов. Протокол и роль объекта. Определение спецификации и проектирования ПО при объектном подходе. UML.

ООП. Предполагает выделение предметной области подлежащей автоматизации, набора действующих объектов и объектов принимающих действия, определение их функционала, с последующей разработкой требуемых классов.

Роли объектов

- Актер - не разрешать влиять на данные
- Сервер – объект представляющий ресурсы, которые подлежать воздействию
- Агент – Способен как подвергаться воздействию, так и подвергаться воздействием на другие объекты. Является посредником между актером и сервером

Для определения спецификации в объектном подходе используют UML (Unified Modeling Language — унифицированный язык моделирования). Он является стандартным средством описания проекта, и описателем для ООП. Любая сложная система может быть описана через совокупность моделей. Выделяют концептуальную и физическую модель системы. Концептуальная модель определяет логическое представление системы, т.е. определение правил работы системы конечного пользователя с описанием внешних и внутренних структурных отношений и представление системы во время ее функционирования с учетом масштабируемости отдельных компонентов и производительности отдельных узлов.

Физическая модель определяется через реализацию и размещение компонентов на физическом устройстве. За концептуальную отвечает конечный пользователь. За реализацию – программист, за размещение – системный администратор.

Для описания моделей в UML введен набор предметов. Предмет – абстракция, являющаяся основным элементом модели, которая взаимодействует через отношение и объединяется через диаграмму. В UML существует 4 разновидности предмета:

- Структурные. Являются существительными в UML моделях, представляют статическую часть их существует 8:
  - Класс. Представляет описание множества объектов разделяющий одинаковые свойства, выполняющие одинаковые операции и общую семантику. Имя, набор свойств и операций. Способен реализовать различные интерфейсы. Интерфейс – набор операций, который определяет услуги класса или компонента. Он описывает поведение компонента, видимого из вне. Интерфейс описывает полные или частичный набор услуг
  - Кооперация. Определяет взаимодействие совокупности ролей одного или нескольких объектов. Обычно реализуется через шаблоны
  - Актер. Набор согласованных ролей, которые может играть пользователь при взаимодействии с системой. Каждая роль требует от системы соответствующего поведения.
  - Прецедент. Определяет описание последовательности действий, выполняемых системой в интересах отдельного автора. Прецедент создает видимый для автора результат.
  - Компонент. Физическая часть системы, соотвествующая набору интерфейсов, чаще всего реализовано в виде отдельного файла. Компоненты существуют различного рода: библиотечные, исполняемые и тд
  - Узел. Физический элемент, который существует в процессе работы системы, представляющий собой ресурс для выполнения ПО. Бывают узлы с памятью и без.

- Поведения.
    - Взаимодействия. Набор сообщений, которыми обмениваются объекты в конкретном контексте для достижения определенной цели. Взаимодействие определяется самим сообщением, последовательностью действий и связи между объектами.
    - Конечный автомат. Определяет последовательные состояния объекта, в которые он переходит при возникновения тех или иных событий, элементы являются: переходы, состояния и события.
- Группирующие. Организационная часть UML, по сути пакет документов, описывающий интерфейс системы
- Поясняющие. Разъясняющая часть UML, проявляющаяся в виде описания, комментария и тд

В UML есть 4 типа отношений:

- Отношение зависимость. -----→ . Семантическое отношение между 2 предметами, при котором изменение в одном предмете влияет на семантику в другом предмете. Тот который влияет – независимы, а наоборот зависимый
- Ассоциация. Структурные отношение, которое описывает набор связей, соединяющих 2 объекта. Они именуются и ассоциируются.
- Обобщение. Отношение специализации или обобщения, в котором объекты специализированного элемента могут объединять объектами общего.
- Отношение реализации. Семантическое отношение, применяемое для описания реализации отдельных интерфейсов между предметами.

Используя предметы и решения разрабатываются диаграммы. Все диаграммы делят:

- Диаграммы поведения:
    - Диаграмма состояния объектов. Моделирует поведение объекта при переходе из одного состояния в другое. Состояния ассоциируются с набором выполняемых действий:
        - Простые
        - Составные. Включают 3 вида действия: при в ходе, в процессе нахождения и действие перед выходом
    - Диаграмма деятельности. Моделирует поведение системы в рамках различных вариантов использования. Для каждого актера приводится плавательная дорожка. Они имеют линии синхронизации, которые определяют начало или окончания действий отдельных актеров.
    - Последовательности. Характеризует набор действий выполняемыми объектами системы, развернутых во времени. Элементами диаграммы последовательности являются объекты с их временем жизни. Запуск каждого объекта осуществляется либо внешним воздействием, либо сообщением другого объекта.
    - Кооперации. Определяет процесс обмена сообщениями между объектами.
- Статические
    - Диаграмма классов. Для моделирования структуры для определения связи между элементами

- Вариантов использования. Диаграмма вариантов использования применяется для моделирования бизнес процессов и выявления требований к системе. Элементами являются:
    - Авторы
    - Прецеденты

  Авторы и прецеденты связанны различными видами отношений.

- Диаграммы реализации и развертывания.
    - Развертывания. Определяет топологию системы. Показывает набор узлов с размещаемыми на них компонентами.
    - Компонентов. Определяется набор интерфейсов и модулей из которых она состоит

8. Разработка интерфейса программного обеспечения. Критерии оценки пользовательского интерфейса

Вычислительную систему можно оценивать по ряду критериев:

- Точность
- Надежность
- Удобство

Интерфейс – способ, которым выполняется какая-либо задача при помощи специализированных средств. При проектировании графического интерфейса пользователя необходимо учитывать психофизические особенности человека. Психофизические особенности определяются моделью мозга.

Интерфейс пользователя

Ценности:

- Защита материальных ценностей: данные, программу и т.д.
- Интерфейсная защита от потери данных
- Сокращение времени на повседневные операции
- Принципы хороших интерфейсов
    - Простота ментальной модели
    - Доступность основных функций
    - Скорость работы
    - Обратная связь с интерфейсами
    - Минимизация ошибок при пользовании интерфейсом
    - Информативность интерфейса

Критерии оценки пользовательского интерфейса

- Простота в освоении и запоминании операций
- Скорость достижения целей задачи, решаемой с помощью системы
- Устойчивость системы к ошибкам пользователя

- Адаптируемость
- Субъективная удовлетворенность

Принципы проектирования интерфейса пользователя

- Учет знаний пользователя
- Согласованность
- Минимум неожиданностей
- Способность к восстановлению
- Руководство пользователя
- Учет разнородности пользователей

Типы и формы диалога

- Типы диалога
    - Управляемый пользователем
    - Управляемым системой
- Форма диалога
    - Директивная. Командная строка
    - Фразовая. Пример: компиляция строчек программ
    - Табличная. Обычные юзер-формы

Процесс проектирования интерфейса пользователя

1) Изучение и анализ действий пользователя
2) Создание проекта прототипа интерфейса, макета
3) Оценка проекта пользователем
4) Разработка концепта
5) Создание детального дизайна интерфейса
6) Исполняемый прототип
7) Реализация интерфейса

Структуры взаимодействия пользователя с системой

- Вопрос – ответ. Недостатки: дотошности вопросов, которые пользователь вполне может игнорировать
- Прямое манипулирование. Интуитивно понятное взаимодействие. Довольно сложная реализация с точки зрения программиста.
- Интерфейс типа «Меню». Медленный вариант для опытных пользователей.
- Экранные формы. Очень просто вводятся данные, но занимает пространство на экране
- Командный язык. Мощный и гибки, но труден для изучения
- Естественный язык. Подходит неопытным пользователям. Требует наличие ИИ

Требования к пользовательскому интерфейсу:

Оптимальная плотность распределения информации на экране должна быть выбрана исходя из следующих правил:

- Необходимо оставлять пустым половину окна.
- Если вы работаете с таблицами, то необходимо оставлять пустой каждую 5-ю строку и 3-4 пробела между столбцами
- Использовать минимальное количество цветов, не более 5
- Использовать разные цвета для отображения изменения состояния в системе
- Использовать цветовое кодирование: выделение аномальных элементов красным цветов, максимально разрешенных в зеленый
- Не использовать цвета дополнительного цветового диапазона (фиолетовый, люминесцентно-желтый и т.д.)
- При написании текста использовать контрастные цвета.

Оценка эффективности заполнения экрана:

- Метод прямоугольников. Экран разбивается на участки, каждый из которых забивается текстом. Через центр экрана проводится ось, позволяющая оценить сбалансированность загрузки экрана, чем больше получилось маленьких прямоугольников, тем хуже.
- Метод выделенных точек. Позволяет определить область экрана, к которой максимально будет привлечено внимание пользователя. Чем больше точек находится в том месте.

При разработке интерфейса и согласовании его с пользователем необходимо его описывать. Для описания интерфейса могу использоваться различные методики. Одна из наиболее популярных методик – описание интерфейса с помощью сети переходов. Сеть переходов отражает набор состояний, в которых может находится система и условий перехода из одного состояния в другое. На практике даже небольшие программные системы имеют обширную сеть перехода. Набор состояний – вершина сети, а на дугах условия. Для сокращения сети переходов и облегчения взаимодействия с заказчиком, выполняют агрегацию сети переходов:

- Из сети переходов удаляются вершины являющиеся не функциональными
- Если между двумя вершинами существуют несколько переходов по различным условиям, то переходы объединяются в одну дугу, а условие записываются через запятую.
- Если есть переход по умолчанию, то он заменяется подчеркиванием одного из условий

Для облегчения восприятия сети переходов строят таблицу вектора маршрутизации. Эта таблица содержит определение всех возможных переходов из всех возможных состояний системы. По строкам таблицы располагаются состояния системы. Столбцы:

- Exit. Действия, которые будут выполняться по умолчанию
- Jump. Определяются безусловные действия
- Оставшиеся столбцы – набор условий.

Для описания диалога необходимо определить синтаксис входных и выходных сообщений. Это определяется понятием грамматика. Чаще всего грамматика определяется набором правил продукционной системы. Эта система предназначена для хранения о предметной

области в формате «если-то». Таким образом грамматика – набор правил, определяющих конструкции языка. Грамматика включает в себя:

- Терминальные символы. Т.е. основные атомарные символы из которых формируются сложные структуры.
- Не терминальный символ. Символ, используемый для составления составных конструкций.
- Правило вывода. Правило, по которым строятся не терминальные символы определяющие основные конструкции языка.

При проектировании интерфейса необходимо использовать средства поддержки пользователя. Средства поддержки пользователя делятся на 2 категории:

1) Сообщения системы. В качестве сообщений системы чаще всего используются сообщения о пользовательских ошибках. Основным требованиями является максимальная однозначность и предоставление возможности пользователю устранить ошибку.
2) Справочная информация.

Факторы проектирования:

- Содержание. Указание на действий пользователя для их продолжения
- Опыт пользователя. В системе должны поддерживаться как для опытных, так и для начальных пользователей
- Профессиональный уровень. Должны использоваться термины, соответствующие программе и пользователю
- Стиль общения.
- Культура. Учет культуры страны.

Для поддержки пользователя разрабатывается пользовательская документация. Документация пользователя должна содержать функциональное описание, документы по инсталляции, справочное руководство и руководство по администрированию. Каждый тип документов предназначен для конкретного типа пользователя.

Функциональное описание – набор тестов соответствующие ТЗ.

9 вопрос. Тестирование

Любая программная система перед введением в эксплуатацию должна подвергаться проверке или инспектированию.

Основные формы инспектирование:

- Визуальный контроль. Проверка программы вручную без использования компьютера.
    - Соответствие комментариев тексту
    - Условные операторы
    - Сложные логические выражения
    - Завершенность цикла
- Статический контроль. Проверка программы по тексту с помощью инструментальных средств. Выделяют 4 формы статического контроля:

    - Синтаксический контроль. Выполняет компилятор.
    - Проверка структурированности. Выполняет линковщик, определяет количество модулей и связи
    - Контроль правдоподобия программы. Выделяются синтаксически правильные конструкции, которые могут нести конструкцию.
    - Верификация. Аналитическое доказательство правильности функционирования системы. Определяется модель системы и возможные спецификации. При формальной верификации проверяется ее надежность и корректность. Надежность характеризует как саму программу, так и ее окружение. Допускается определенная вероятность наличия ошибок в программе.
        - Целостность программы: Возможность защиты от отказа
        - Живучесть: способность к входному контролю данных и проверке их во время работы
        - Во время сбоев возможность восстановить состояние

      Частичная корректность и полная корректность.

- Динамический контроль. Заключается в проверке правильности По на компьютере. Чаще всего это называется тестированием.

Основная цель тестирования: доказать, что программа не работает.

При написании текстов необходимо придерживаться принципам Майерса:

- Частью каждого теста должно являться описание ожидаемых результатов работы программы
- Программа не может тестироваться его автором
- Результаты каждого текста должны проверяться. Тексты должны быть грамотно подобраны и многократно использоваться
- Принцип скопления ошибок: вероятность нахождения ошибки в том месте где она была определена прямо пропорционально числу обнаруженных ошибок.

Международный стандарт регламентирует следующий набор ошибок:

- Ошибка. Состояние программы при котором выдаются неправильные результаты, причинами которых является неверная технология обработки данных
- Дефект. Следствие ошибок разработчика, которые могут содержаться в исходных или проектных спецификациях, приводящих к сбою в работе программы. Сбой может трактоваться как отказ или восстанавливаемый сбой.
    - Отказ – отклонение программы от функционирования с невозможностью дальнейшей выполнением функций
    - Сбой – восстанавливаемый отказ, когда после неправильного функционирования программа может вернуться к предыдущему состоянию

Виды тестирования:

- Модульное тестирование. Когда берется отдельный класс или функция, а затем проверяется возможность использования этого компонента.
- Интеграционное тестирование. Проверяется наличие проблем в интерфейсах и в способах взаимодействия, интегрированных в компонент.

- Системное тестирование. Тестируется система на соответствие исходным требованиям
    - Альфа тестирование. Предполагает имитацию реальной работы системы без реального ее выполнения. Выполняется разработчиком в качестве внутреннего приемочного тестирования. В большинстве случаев выполняется под отладчиком для обнаружения ошибок.
    - Бета тестирование. Предполагает распространение версии с ограничением для работы заказчику, выявляются ошибки связанные с восприятием системы, некорректные обработки данных. Цель – получение обратной связи от пользователей программного обеспечения

Любое тестирование обеспечивает обнаружение ошибок, демонстрацию соответствия функции программы и ее назначения, демонстрация реализации требований к характеристикам программ. Тестирование не предназначено для доказательства отсутствия дефектов в программе.

Функциональное тестирование

Функциональное тестирование – тестирование по методу черного ящика, которая базируется на том, что для проверки работоспособности системы используются тесты, основанные на спецификации разрабатываемого ПО. Поведение тестируемой системы определяется путем изучения соответствия входным данным выходным. Методология тестирования черного ящика предполагает, что тестировщик имеет доступ только к внешним элементам, без доступа к внутренним структурам.

Для выполнения функционального тестирования используется 4 метода:

- Эквивалентное разбиение. Выбирается подмножество программы, которое тестируется. Позволяет создавать тесты, включающие в себя подмножество входных данных, обладающих свойствами:
    - Каждый элемент подмножества уменьшает число тестов, которые должны быть разработаны для приемлимого тестирования.
    - Каждый элемент подмножества должен покрывать значительную часть других тестов, подтверждая наличие или отсутствие ошибок в данной функции программы.
    - Каждый тест должен влючать в себя столько условий, сколько необходимо для минимизации общего числа тестов. Входная область данных должна разбиваться на конечное число классов эквивалентности, при этом тест написанный для одного представителя данного класса должен гарантировать соответствующую обработку другого представителя класса.

  Виды данных:
    - Корректные. Данные, которые обязана обрабатывать программа.
    - Не корректные. Данные, которые не должна обрабатывать данные.

  Разработка тестов делится на 2 этапа:
    - Выделение классов эквивалентности.
        - Если входное условие описывает некую область значения, то определяется правильный класс эквивалентности и 2 неправильных
        - …
    - Построение тестов.

- Анализ граничных значений. Основан на методе эквивалентных разбиений, но применение обусловлено тем, что большинство ошибок возникают на границе. Основные отличия в том что тестовый вариант создаются для проверки границ и для создания тестовых вариантов учитывается область ввода и вывода (допустимое отклонение). Правила анализа:
    - Если условие ввода задается диапазоном, то строятся тестовые наборы для границ диапазона, и чуть ниже и чуть выше.
    - Если задается дискретное множество, то задаются тесты для проверки максимального из значений и минимальное. И значения чуть меньше минимума и максимума
    - Если структуры данных имеют границы, то тесты проверяют все границы для структуры. Если данные – упорядоченное множество, то проверятся первое и последнее значение множества.
- Функциональные диаграммы. Позволяет решит недостаток анализа граничных значений связанных с возможностью. Позволяет обнаружить неполноту и неоднозначность исходных спецификаций. Сама фукнциональная диаграмма представляет собой формальный язык на оторый транслируется спецификация написанная на естественном языке. На основании входных и ожидаемых результатов строятся тесты. Шаги:
    - Определение причины и следствия
    - Присвоение идентификатора
    - Граф причинно-следственных связей
    - Таблица решений из графа
    - Столбцы в тестовый набор
- Метод предположения об ошибке.

Структурное тестирование

Базируется на анализе логике ПО и подбирается набор данных, позволяющий протестировать все маршруты функционирования системы. Структурный подход выполняется:

- Метод покрытия операторов
- Метод покрытия решений
- Покрытия условий
- Комбинация методов

Недостатки:

- Не обнаруживает пропущенных маршрутов
- Не обнаруживает ошибок функционала
- Не дается гарантия, что выполняются функциональные требования

Управляющий граф программы – граф отображающий поток управления программы.

Каждый из методов предполагает соблюдение критерия. Критерий покрытия оператора подразумевает, что необходимо создать набор тестов, чтобы каждый оператор программы выполнялся по крайней мере только один раз. Значит, нужно такие маршруты, которые проходили через все вершины

Критерий покрытия решений. Для реализации данного критерия необходимо такое количество и состав теста, чтобы в результате было проверено каждое решение т.е. условная вершина принимала значение истина или ложь. Покрытие решения заключается в том, что необходимо подобрать тесты, чтобы каждая дуга содержалась по крайней мере в одном из пути тестирования.

Необходимо задать такое количество тестов, что результат каждого условия, внутри каждого решения принимали значения истина или ложь хотя бы 1 раз. Данный критерий дополняется предположением, что в каждой точке входа управление должно быть передано хотя бы 1 раз.

Комбинаторное покрытие условий предполагает выполнение критерия при котором формируется множество тестов в котором каждая комбинация условий выполняется хотя бы 1 раз.

Построение наборов тестов

Состоит из этапов:

- Конструирование управляющего графа
- Выбор тестовых путей
    - Статические методы
        - Эвристические методы
        - Адаптивные методы
    - Динамические методы
    - Методы путей
- Генерация тестов, соотвествующих тестовым путям

Статические методы предполагают, что каждый тестовый путь строится посредством постеменного удлинения пути за счет добавления следующих дуг, которые являются выходными для вершины, удлинение выполняется до достижения конечной вершины графа.

Основным достоинством является использование сравнительно небольшое число ресурсов.

Недостатки: не рассматриваются не реализуемые пути, сложно обойти все пути.

Статические методы реализуются с помощью эвристик и адаптивно.

Эвристики позволяют строить каждый тестовый набор по аналогии с ранее проверенными программами. Таким образом сокращается число генерируемых тестов.

Адаптивные методы заключаются в том, что каждый раз во входной тест добавляется только один тестовый путь, причем при выборе следующего пути руководствуются индуктивной стратегией. Чаще всего адаптивные методы используются для критериев покрытия операторов. С практической точки зрения реализация каждой покрытой вершины позволяет определить достигаемость состояния той или иной системы.

Динамические методы

Позволяют построить систему тестов, которые удовлетворяют заданным критериям. При использовании динамических методов решается задача построения покрывающего множества путей с одновременным формированием набора тестовых данных. При этом возможно автоматическое учет реализуемых и нереализуемых путей. Основная идея заключается в присоединений начальным реализуемым отрезкам путей дальнейших маршрутов так, чтобы реализуемость вновь полученных путей покрывала ранее протестированные участки программы.

Недостаток: ресурсоемкая методика

Достаток: высокий уровень проверки и оценка реализуемых путей.

Метод реализуемых путей

Основан на том, что прокрывающее множество путей строится на базе всех реализуемых путей в системе. Нереализуемые не рассматриваются, могут быть исключены как возможности неисправностей программы.

Генерация тестов соответствующим тестовым путям

На данной фазе на каждом пути тестирования необходимо найти каждый тест реализующий данный путь. Каждый путь описывается коньюнкцией предикатов. Где C_i предикат. Данный предикат задает значение в данной вершине. Конечный предикат формируется в процессе обратной подстановки. Т.е. формируется набор неравенств, реализующих тот или иной тест, ожидаемое значение предиката соответствует проверяемому значению тестовому пути.

Место конкретизации ошибки выполняется путем обратной раскрутки. Где для каждого введенного предиката решения определяется возможность его выполнения. Степень тестирования программы является показателем качеством программы. На практике построить полный набор тестов невозможно, поэтому проектируются эффективные тесты из соображений, что тестирование должно содержать не более 30% времени разработки ПО. Даже если удалось провести полное тестирование, возможно дальнейшее выявление ошибок.

- Программа может не соотвествовать внешней спецификации
- Возможность возникновения ошибок, которые зависят от обрабатываемых данных

Функциональное + структурное.

Особое внимание уделяется тестирование многомодульных программных коммплексов. Тогда методика тестирования разворачивается в виде спирали. Каждый ее виток определяет тестирование определенного уровня структуры. Тестирование элементов имеет своей целью индивидуальную проверку каждого закодированного модуля. На этом этапе используется методика структурного тестирования. Тестирование интеграций позволяет выявить ошибки связанные с этапом проектирования. Цель тестирования – определить правильность и целесообразность сборки модулей в программную систему. Тут применяется метод черного ящика, функциональные диаграммы

Тестирование правильности своей целью считает реализация программной системы с точки зрения ее поведения. Таким образом проверяются технические требования, которые выполнены на этапе анализа. Используются тут граничные значения

Системное тестирование проверяет правильность тестирование.

Тестирование многомодульного ПО

- Монолитное тестирование. Каждый модуль проверяется по отдельности, а потом целиком. Для вызова каждого модуля разрабатывается специальный модуль, который называется модулем драйвера. Модуль-драйвер моделирует вызов тестируемого модуля со стороны внешних процедур. Для тестирование механизма взаимодействий пишется набор заглушек, которые моделируют работу вызываемых модулей. Преимущество: возможность параллельного выполнения тестов с минимальными затратами машинного времени.
- Пошаговое. Каждый модуль подключается к оттестированной части программы. Основным достоинством: небольшая трудоемкость тестирования и ускоренная отладка за счет быстрого отслеживания ошибок.

- Нисходящее. Тестирование начинается с главного модуля. Выбираются модули ввода-вывода, а потом к логике.
- Восходящее. От не основных модулей, до глобальных. Основной модуль тестируется в конце.
- Комбинированное. Модули верхних уровней тестируются нисходящим, а нижних восходящим.

Виды тестирования ПО

При тестировании системы выделяются различные направления тестирования:

- Функциональное. Проверка функции системы на требования
  - Функциональное
  - Безопасности
  - Взаимодействие с пользователем
- Не функциональное. Исходит из того, что все функции реализованы и необходимо определить качество реализованных функций. Предполагает тестирование производительности, сложности установки, удобства использования, тестирование на отказ, конфигурационное тестирование
- Тестирование производительности.
  - Нагрузочное. Автоматизированное тестирование, имитирующее работу определенного числа пользователей с общим разделяемым ресурсом. Общей задачи является определение возможности масштабирования системы под нагрузкой, при этом замеряется время выполнение, количество пользователей, граница приемлемой производительности.
  - Стрессовое. Предполагает максимальной трудозатратной операции на предельной нагрузке. С определением момента деградации производительности.
  - Надежности. Соблюдение работоспособности под нагрузкой заданный период времени.
  - Объемное тестирование. Оценка производительности при увеличений объема данных. Измеряется время отклика и обработки.
- Тестирование установки. Успешность инсталяции и настройки По. Определяем необходимость обновления или удаления ранее установленного ПО.
- Тестирование удобства использования. Выявляется при бете.
- Тестирование на отказ и восстановление. Способность возврата к старому состоянию, бекапы.
- Конфигурационное тестирование. Вид тестирования направленный на проверку работы ПО в различных конфигурациях.
  - Проверка оптимальной конфигурации средств.
  - Совместимость конфигураций.
  - Проверка возможность реализации конфигураций.

Виды тестирования ПО

- Связанный с изменением

- Дымовое тестирование. При вводе в эксплуатацию новой системы тестировалось удачность выполнения и тестирование на сбой. Короткий цикл тестов, который подтверждает работоспособность программ.
- Регрессионное тестирование. Направленно на проверку изменений, сделанных в приложений или окружающей среде для подтверждения того факта, что существующий ранее функционал остается доступным.
  - Регрессия багов. Нет новых багов
  - Регрессия старых багов. Нет старых багов
  - Регрессия побочного эффекта. Попытка доказать, что изменения не лишило работоспособности другие части системы.
- Тестирование сборки. Направленно на определение соответствии выпущенной сборки.

Критерии завершения

- Выполнены все методики проведения тестов и все тесты дают отрицательный результат.
- Когда количество выявленных ошибок не превышает допустимого значения.
- Когда количество выявленных тестов значительно снижается по сравнению с числом ошибок, которые выявленные ранее.
- Когда завершено время тестирования

Отладка

Процесс локализации и исправления ошибок, которые обнаружены при тестировании. Локализация – определение точки или операции выполнение которой вызвало нарушение нормального вычислительного процесса. Для исправления ошибки необходимо выявить ее причину. Причины могут быть как очевидными, так и скрытыми. Сложность отладки обусловлена следующими причинами:

1) В ряде случаев требуются глубокие знания специфики управления вычислительными средствами, ОС, средой, в которой реализуются процессы.
2) Одной из причин сложности отладки является психологический дискомфорт так как ошибки должны исправляться в ограниченное время
3) Возможно возобновление ошибок за счет пере размещения программных модулей по выделяемой оперативной памяти.
4) Отсутствуют четко сформулированные методики отладки.

Все ошибки выполнения делятся на:

- Ошибки определения данных. Ошибки, связанные с передачей. Ошибки преобразования. Ошибки перезаписи и неправильные данные.
- Логические ошибки.
  - Неправильное проектирование. Неверная парадигма, неверный алгоритм, неверные структуры
  - Кодирование. Неправильная работа с переменными, организация интерфейса, некорректная реализация алгоритмов и вычислений.
- Ошибки накопления погрешности. Вызваны ограниченной разрядной сеткой, либо игнорирование механизмов уменьшения погрешности.

Сложность отладки увеличивается в следствии влияния следующих факторов: возможность получения внешних появлений разных ошибок.

Методы отладки:

- Ручное тестирование
- Индукция. Анализ симптомов ошибок, которые могут проявляться как неверный результат, либо как сообщение об ошибках. Выявляется место возникновения ошибок, выдвигается гипотеза, почему эта ошибка появилась, если гипотеза не подтверждается, то делается новое предположение.
- Дедукция. Формируется набор причин, которые могли бы вызвать данное проявление ошибки, затем причины анализируют и исключают те, которые противоречат имеющимся данным. Если все причины исключены, то выполняется дополнительное тестирование, в противном случае выбирают наиболее вероятную гипотезу и пытаются ее доказать. Если она объясняет причины появления ошибки, то ошибка считается найденной, иначе проверяют следующую причину.
- Обратное прослеживание. Используется для небольших программ, где устанавливают точку получения неправильного результата. Для этой точки определяется набор основных характеристик. Исходя из полученной гипотезы делают предположение о месте неправильном получения характеристик, продолжается до тех пор, пока не удается получит причину ошибку. Если данных недостаточно, то используют дополнительные средства или отладочные средства. Так же могут использоваться независимые отладчики, которые могут выполнить любой фрагмент программы в пошаговом режиме и проверить значение переменных. Эти отладчики позволяют отлаживать ассемблерные программы.

Отладка программы

1) Изучение появления ошибки.
2) Локализация ошибки.
    a. Отсечение частей программы.
    b. Использование отладчиков.
3) Определение причины ошибки.
4) Исправление.
5) Повторное тестирование.

10 вопрос. CASE-технологии. Состав, структура, функции и классы CASE-средств

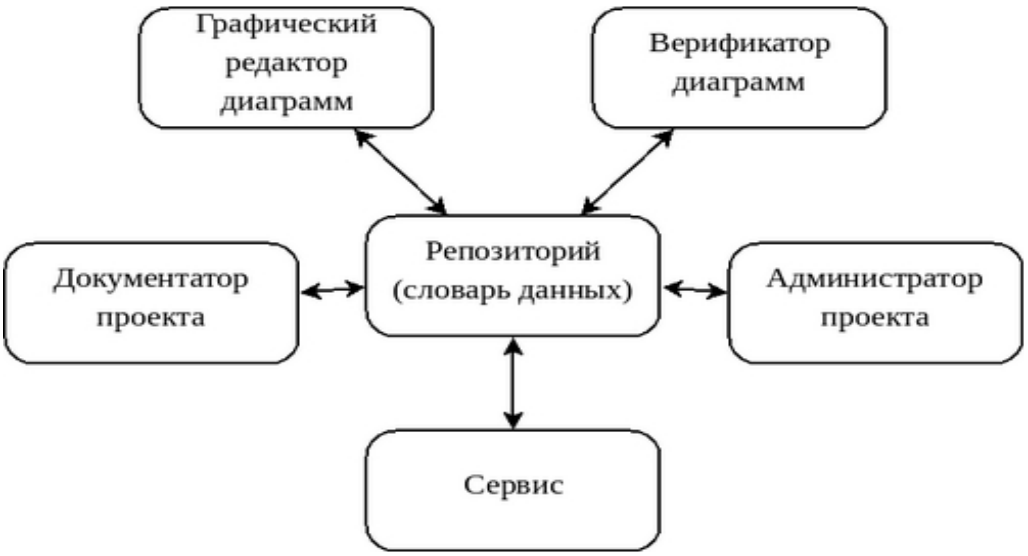
Средства автоматизации разработки программ (CASE-средства) — инструменты **автоматизации** процессов проектирования и разработки ПО для системного аналитика, разработчика ПО и программиста; программные средства для **поддержки** процессов **жизненного цикла ПО**.

Характерные особенности:

- Мощные **графические средства** для **описания и документирования ИС**, обеспечивающие удобный интерфейс с разработчиком и развивающие его творческие возможности

- **Интеграция отдельных компонент** CASE-средств, обеспечивающая управляемость процессом разработки ИС
- **Использование** специальным образом организованного **хранилища проектных метаданных** (репозитория)

Архитектура CASE-средств



Репозиторий данных

Специализированная **база данных** для отображения **состояния проектируемой информационной системы** (ИС) в любой момент времени. В нём хранится информация об объектах проектируемой системы и **все подсистемы обмениваются данными с ним**:

- Проектировщики и их права доступа к различным компонентам системы
- Организационные структуры
- Диаграммы
- Компоненты диаграмм
- Связи между диаграммами
- Структуры данных
- Программные модули
- Процедуры
- Библиотеки модулей

Графический редактор диаграмм

Предназначен для **отображения в графическом виде** в заданной нотации **проектируемой ИС**. Он позволяет:

- Создавать элементы диаграмм и взаимосвязи между ними
- Задавать описания элементов диаграмм
- Задавать описания связей между элементами диаграмм
- Редактировать элементы диаграмм, их взаимосвязи и описания

Верификатор диаграмм

Верификатор диаграмм служит для **контроля правильности построения диаграмм** в заданной методологии проектирования ИС. Он выполняет:

- Мониторинг правильности построения диаграмм
- Диагностику и выдачу сообщений об ошибках
- Выделение на диаграмме ошибочных элементов

Администратор проекта

Администратор проекта представляет собой инструменты, необходимые для выполнения следующих **административных функций**:

- Инициализация проекта
- Задания начальных параметров проекта
- Назначения и изменения прав доступа к элементам проекта
- Мониторинга выполнения работ

Документатор проекта

Документатор проекта позволяет **получать информацию о состоянии проекта** в виде различных **отчётов**. Отчёты могут строиться по нескольким признакам, например, по времени, автору, элементам диаграмм, диаграмме или проекту в целом.

Сервис

Набор **системных утилит по обслуживанию репозитория**. Данные утилиты выполняют функции **архивации** данных, **восстановления** данных и создания нового репозитория.

Функции

Функционал подробно рассмотрен в п. Архитектура

- Разграничение процесса проектирования от процесса кодирования и последующих этапов разработки
- Максимально автоматизировать процесс разработки
- Подготовка проектной документации
- Наглядное (графическое) представление ИС

Классификация

Классификация по типам – **функциональная ориентация** средств на те или иные **процессы** жизненного цикла разработки ПО:

- Средства анализа — предназначены для построения и анализа модели предметной области
- Средства проектирования баз данных
- Средства разработки приложений
- Средства реинжиниринга процессов
- Средства планирования и управления проектом
- Средства тестирования
- Средства документирования

Вспомогательные типы:

- средства планирования и управления проектом
- средства конфигурационного управления
- средства тестирования
- средства документирования

Классификация по категориям – **степень интегрированности** по выполняемым функциям:

- Отдельные локальные средства – решение небольших автономных задач
- Частично интегрированные средства – охватывают большинство этапов жизненного цикла
- Полностью интегрированные средства – охватывают весь жизненный цикл

11 вопрос. Сетевое ПО

- Операционные возможности
- Производительность. Средне суммарное быстродействие
- Время передачи сообщения. Пропускная способность, расстояние
- Стоимость обработки данных. Время и объем хранения данных

Все сети можно классифицировать по признакам:

- Корпоративность
    - Локальная
    - Глобальная
- Организация сетевого ПО
    - Одно ранговые. Все равны.
    - Сети централизованного управления.
- Метод доступа
    - Ethernet
    - Token ring
    - Arcnet
- Протоколы передачи данных
    - PTP. Peer to Peer
    - Множественные.
- Принципы коммутации
    - Каналы
    - Пакеты

Коммутация каналов предполагает, что сеть образует между узлами непрерывный физический канал, который построен из последовательно соединенных узлов, следовательно, перед передачей информации по сети передается управляющий пакет, который обеспечивает настройку физического соединения между транзитными узлами.

Коммутация пакетов. Каждый пакет имеет заголовок и конец, которые определяют сообщение.

- Сети с динамической коммутацией. Позволяют устанавливать соединения по инициативе пользователя сети. Коммутация выполняется только во время сеанса связи, а затем по инициативе одного из пользователя разрывается. В любой момент времени пользователь может соединиться с любым пользователем сети. Время соединение между пары пользователей составляет от нескольких секунд, до часов и завершаются при окончании передачи связи. Примеры: локальные сети, TCP/IP
- Сети с постоянной коммутацией. Позволяют паре пользователей заказать постоянное длительное соединение на требуемый период времени, при этом соединение устанавливается не пользователем, а оборудованием. Обычно выделяется длительная аренда канала, на котором выделяется обслуживание. Сети с постоянной коммутацией называют сетями с выделенным каналом. Наиболее популярными сетями являются SDH. В настоящее время сети SDH реализуются на технологиях, которые позволяют к дополнительной коммутации использовать динамическую коммутацию с целью разделения трафика.

Существуют 2 основных класса компьютерных сетей с коммутацией пакетов: дейтаграммы сети, сети с виртуальными каналами.

Диаграммных сетях каждый пакет обязан содержать информацию о получателе, причем этот адрес имеет иерархическую структуру. Каждый получатель анализирует фрагмент адреса и на основании собственной таблицы маршрутизации формирует адрес следующего по маршруту узла. Таким образом в случае дейтаграммы сетей паке имеющие одинакового получателя, могут быть направлены по разным маршрутам. Обычно дейтаграммы сети определяются как сети без гарантии доставки и чаще всего используются в случае, если потеря не существенная.

Сети с виртуальным каналом характеризуются:

- Маршрут.
- Номером виртуального канала.
- Записями в таблице маршрутизации.

Стратегии повторной передачи

Stop and wait. Каждый следующий пакет отправляется только тогда, когда получен предыдущий пакет. В целях реализации введено понятие квитанций. Оно определяет, что доставка пакета прошла успешно.

Go Back N. Предыдущий узел может отправить пакеты без ожидания. Каждый пакет имеет номер в пределах окна. Принимающий узел отправляет положительную квитанцию и номер следующего пакета. Нет затрат на хранение пакетов, которые были получены, но не требуемые.

Выборочная стратегия. Плавающее окно, но на принимающем узле есть маска, в которой отмечается номера пакетов, которые приняты без подтверждения. В этом случае при повторной передачи передаются только те пакеты, которые не приняты на принимающей стороне.

Распределенная обработка информации предполагает организацию доступа к одним и тем же информационных блокам различным приложениям. В данном случае под информационным блоком можно понимать ПО так и непосредственные блоки данных. Распределенная обработка разрешает недостатки чисто архитектуры клиент-сервер, так как включает в состав программно аппаратного средства дополнительный промежуточный слой, который содержит набор сервисов, отвечающий за разделение информации. Введение промежуточного слоя обусловенно недостатками:

- Ограниченные возможности масштабирования архитектуры
- Необходимость изменения клиентских приложений при изменении работы серверной логики

Middleware – класс ПО предназначенный для объединения компонентов распределенного клиент-сервисного приложения или различных приложений в единую информационную систему. Данное ПО представляет собой набор сервисов обращение к которым позволяет различным приложениям расположенных на разных станциях взаимодействовать между собой. При этом разработчик не обязан вникать в тонкости удаленной реализации:

1) Прозрачный доступ к другим сервисам и приложениям
2) Независимость от сервисов
3) Высокая надежность и постоянная готовность

Используемый в настоящее время промежуточное ПО можно разбить на 2 класса:

1) ПО межпрограммного взаимодействия
2) ПО доступа к БД

Протоколы и продукты по межпрограммному взаимодействию обычно испльзуются в корпоративных ИС и могут быть реализованные в 3-х формах:

1) Удаленный вызов процедур. Разработана и реализована ксероксом (компания). Программа выполняемая на машине клиента использует процедуры, находящиеся на машине сервиса абсолютном прозрачным способом. Работа выполнена в соответствии с работой прерывания. Стаб клиент моделирует работу связанную приостановкой выч процесса и получение результата вызываемой процедуры. На сервере стаб сервер – спец ПО, которое обеспечивает прием вызываемых параметров, вызов требуемой процедуры, получение результатов и обратная связь с клиентом.
2) Сервисы обмена сообщений. Ассинхронная система, которая обеспечивает взаимодействие между клиентом и сервером на базе обмена сообщений. Под сообщение понимается текстовый блок, который может быть как программа так и данными. Для передачи сообщения используют байт ориентированные протоколы (Http, smtp). Обмен сообщений
    a. Непосредственны. Сообщения появляющиеся у приложения передаются менеджеру, который осуществляет связь с протоколами передачи данных и передает сообщения на другую машину.
    b. Очереди. Для каждого приложения создается своя очередь, которая буферизируется как принимаемые, так и передаваемые. Обслуживаются менеджером на машине, который обеспечивает сетевое взаимодействие. Наличие очередей обеспечивает доставку сообщения, чаще всего по расписанию. При получении сообщения сервер должен указать успешность передачи данных через механизм повторной передачи и контрольной суммы.
    c. Публикация\подписка. Обслуживание клиента по подписке работает по принципу, сходим на почту. Одно приложение публикует информацию в сети, а приложения подписываются на нее. Как только информация появляется в сети, то приложение ее может захватить. Для этого на машинах запускается ПО, которое принято называть демоном контактов. Основное достоинство: динамическая реконфигурация и полная независимость приложений.
3) Распределенные объектные системы

Мониторы обработки транзакции позволяют контролировать передачу данных от клиента при работе с распределенными БД. Он обеспечивает целостность данных и регулирует завершенность транзакции.

Монитор транзакции отслеживает:

1) Атомарность. Обеспечивает, что транзакция является неделимым блоком при невозможности транзакции, при возможности отката
2) Согласованность. По завершению транзакции все ресурсы должны находится в определенном состоянии.
3) Изолированность. Одновременный доступ к транзакции различных приложений к разделяемым ресурсам координируется так, чтобы исключить влияние приложений друг на друга.
4) Долговременность.

Монитор транзакции отслеживает выполнение критерий и обеспечивает распределенную обработку баз данных.

Одним из механизмов является механизм программных гнезд. Это точка соединения, предназначенная для передачи блоков информации. Программные гнезда – сокеты. Могут быть клиентскими и серверными. Клиентское обеспечивает точки доступа к требуемой информации. Непосредственную передачу и прием информации. Серверное используется для взаимодействия с одним или нескольких клиентов.

Модели сетевой архитектуры

Физический уровень определяет тип соединения среды, физические топологии и способы передачи данных. Распределение каналов связи с использование мультиплексирование. Физический уровень не включает описание среды передачи, однако способ передачи различный. Не регламентируется программно, а аппаратно

Канальный уровень определяет логическую топологию сети, а так же правила получения в среде передачи данных. Решаются вопросы адресации физических устройств, а также вопросов сервисы соединения:

- Организация фреймов по передаче на физический уровень
- Обнаружение ошибок при передаче
- Управление потоками данных
- Идентификация компьютера в сети по их физическим адресам.
- Вопросы повторной передачи.

Сетевой. Служит для образования единой транспортной системы объединяющий несколько сетей, причем эти сети могут использовать различные технологии передачи данных. На сетевом уровне решаются вопросы согласования различных топологий, вопросы адресации и маршрутизации. Сообщение, формируемое на сетевом уровне – пакеты. Каждый пакет определяет адрес и отправителя, и получателя с целью обеспечения маршрутизации. Основными протоколами сетевого уровня являются протоколы формирования маршрутных таблиц, протоколы маршрутизации и разрешения сетевых адресов.

Транспортный уровень. Обеспечивает приложению возможность доступа к протоколам более низкого уровня. Отвечает за качество предоставляемых сетевых услуг, при этом критерием качества является сложность передачи, возможность восстановления, мультиплексирование соединений и способность обнаружения и исправления и передачи данных.

Сеансовый уровень обеспечивает управление диалогом, а именно фиксирование какая из взаимодействующих сторон является активной, а какая пассивной. Средства синхронизации. Диалог реализуется одним из 3-х способов:

- Симплекс. Передача в одну сторону.
- Дуплекс. Обе стороны.
- Полудуплекс. Не одновременно, а по очереди в обе стороны.

Представительский. Осуществляет разбор передачи от приложении информации и преобразовании ее к внутреннему формату. Разрешает синтаксические различия в кодировке символов. Например, SSL.

Прикладной уровень. Включает набор протоколов, с помощью которых пользователь получает доступ. Http, FTP, SMTP, Telnet и т.д.

TCP/IP

Физический (канальный и физический).

Сетевой уровень. IP/ICMP/RIP/OSPF/ARP

Транспортный и сеансовый. TCP, UDP

Прикладной.

1) 0.0.0.0 – служебный адрес, который используется хостом при загрузке
2) Нулевой номер сети – текущая сеть
3) Все единицы – широковещательный

DNS

Предназначена для представления символьного адреса в логический. Есть дерево, в корне которого находится самая последняя точка. Деление доменного пространства между ДНС серверами осуществляется по средством механизма зон. Зона – БД, в которой содержатся записи о соответствии множества доменных имен, IP адреса. Каждая зона представляет собой фрагмент доменного пространства. Зона рассматривается как основной административный элемент. На уровне зоны происходит управление пространства имен. Границы зоны не определяются доменной структурой, по суди одна зона может включать в себя несколько доменов. Зона может быть размещена на одном вычислительном узле или на нескольких. В случае разделении зоны по серверам обычно выделяется основной сервер и только он имеет право на внесение изменений в базу. Остальные реплицированные сервера только хранят копию и доступны для чтения.

DNS – приложение предназначено для ответа на DNS запрос. Выделяют несколько типов ДНС:

- Авторитативный ДНС. Он отвечает за доменную зону
- Мастер сервер\первичный сервер. Сервер имеющий право внесения изменений в данные зоны. Авториративный
- Слейв\Вторичный сервер. Хранит информацию о зоне, не имеющий право на внесение изменений. Авторитативный
- Кэширующий. Сервер, не являющийся авторитативный. Сокращает время получения адресов
- Локальный DNS сервер. Обычно для локальной сети

- Перенаправляющий ДНС. Перенаправляет полученные запросы выше стоящему кеширующему северу в виде рекурсивных запросов. Обычно используется для снижения нагрузки на авторитативные и другие кэширующие сервера.
- Корневой DNS. Авторитативный, который отвечает за корневые домены
- Регистрирующий DNS.

1) Рекурсивный метод.
2) Итеративный.

12 вопрос. Основы технологии COM и CORBA.

COM (Component Object Model) - это стандарт Microsoft, определяющий структуру и взаимодействие компонентов программного обеспечения в современных операционных системах MS Windows. Архитектура современных Windows -приложений основана на COM: мир этих приложений - это мир COM -компонент. Компоненты COM обладают уникальностью и предоставляют другим компонентам COM стандартным образом описанные интерфейсы, позволяющие получить доступ к методам этих компонентов. COM определяет механизм связи только между локальными (т.е. находящимися на том же компьютере) компонентами.
CORBA (Common Object Request Broker Architecture) - это набор открытых спецификаций интерфейсов, определяющий архитектуру технологии межпроцессного и платформо-независимого манипулирования объектами. Разработчиками данных интерфейсов являются OMG и X/Open.
CORBA определяет, каким образом программные компоненты, распределенные по сети, могут взаимодействовать друг с другом вне зависимости от окружающих их операционных систем и языков реализации. Центральным элементом архитектуры CORBA является ORB (Object Request Broker) - программное обеспечение, обеспечивающее связь между объектами, в том числе позволяющее:
- найти удаленный объект по Объектной Ссылке (IOR - Interoperable Object Reference),
- вызвать метод удаленного объекта, передав ему входные параметры (marshaling parameters),
- получить возвращаемое значение и выходящие параметры (unmarshaling parameters).

Тем самым ORB является связующим звеном между распределенными частями основанной на технологии CORBA системы, позволяя одной части системы не заботиться о физическом расположении других частей (объектов) системы. На рынке представлены ORB разных производителей (например, VisiBroker, WebLogic),но все они соответствуют единой спецификации CORBA. Поэтому в принципе CORBA позволяет строить распределенные системы, одновременно используя ORB разных производителей, и строя систему одновременно на различных платформах и различных сетевых протоколах (это в терминологии CORBA называется интероперабельностью - interoperability).В архитектуре CORBA каждый объект, методы которого доступны другим объектам (обычно его называют CORBA -объектом) имеет уникальную по всей доступной сети Объектную Ссылку (IOR - Interoperable Object Reference),по которой к нему можно обратиться. Искать CORBA - объекты можно как по IOR, так и по символическим именам, если они зарегистрированы (обычно при создании) в специальном сервисе имен (NameService).Для обращения к методам CORBA -объекта последний имеет открытый для всех остальных CORBA - объектов интерфейс. Интерфейсы CORBA -объектов принято описывать на специальном, определенном спецификацией CORBA языке IDL (Interface Definition Language). Производители ORB поставляют вместе с ORB также и утилиты, преобразующие описания интерфейсов CORBA -объектов в конструкции соответствующих языков программирования.
Основой интероперабельности является протокол GIOP - General inter-ORB Protocol,предназначенный для связи между объектами и ORB в сети. Стандартизация коммуникационного протокола позволяет разработчикам различных частей корпоративной системы совершенно не заботиться об используемых ORBах в других частях (ORB доменах)

системы. Почти все современные ORBbi строятся на основе IIOP - Internet inter-ORB Protocol (это версия общего протокола GIOP,предусматривающая использование в качестве транспортного протокола TCP/IP).
Спецификация CORBA предусматривает также ряд стандартизованных сервисов (CORBA Services) и горизонтальных и вертикальных Общих Средств (Common Facilities). Сервисы представляют собой обычные CORBA -объекты со стандартизованными (и написанными на IDL) интерфейсами. К таким сервисам относится, например, уже упомянутый сервис имен NameService,сервис сообщений, позволяющий CORBA -объектам обмениваться сообщениями, сервис транзакций, позволяющий CORBA -объектам организовывать транзакции. В реальной системе не обязательно должны присутствовать все сервисы, их набор зависит от требуемой функциональности. На сегодня разработано всего 14 объектных сервисов.
Между объектными сервисами и общими средствами CORBA нет четкой границы. Последние тоже представляют собой CORBA -объекты со стандартизованными интерфейсами. Common Facilities делятся на горизонтальные (общие для всех прикладных областей) и вертикальные (для конкретной прикладной области). Например, разработаны Common Facilities для медицинских организаций, для ряда производств и т.п.

13 вопрос. Оценка качества процессов создания ПО

Каждая фирма разработчик должна иметь сертификат качества, который подтверждает то, что фирма создаст нормальный продукт.

Стандарты качества:

- Международный стандарт ISO 9000
- CMM – модель зрелости предприятия
- SPICE – определение возможности улучшения программных продуктов

ISO 9000

Особенностью стандарта является то, что определяется качество не конкретного создаваемого продукта, а процесс его производство. Считается, что при качественном процессе производства может быть достигнуто качество ПО. Является стандартом менеджмента качества и в нем есть 8 принципов:

1) Принцип ориентации на потребителя. Организация всегда зависит от потребителя этого продукта. Необходимо четкое понимание потребностей как текущих, так и будущих. Продукт должен превзойти ожидания заказчика.
2) Лидерство руководителей. Руководитель любого уровня обеспечивает единство целей предприятия и управляет деятельностью. Он обязан создавать и поддерживать внутреннюю среду, в которую исполнители вовлекаются
3) Вовлечение работника. Работники составляют основу организации, они должны использоваться с максимальной выгодой и максимальным вовлечением в процессе.
4) Процессный подход. Желаемый результат достигается эффективнее, когда деятельностью и ресурсами предприятия управляют как процессом, тогда деятельность предприятия можно рассматривать как процесс на вход которого подаются потребности и в результате получается продукция, которая удовлетворяет потребителя. Она имеет цикличную структуру, цель которой является выпуск продукта. Выпуск идет под руководством руководителя, который имеет в распоряжении набор ресурсов. Под ресурсами понимаются как человеческие, так и материальные. После выпуска предприятие обязано провести анализ продукта и провести его улучшение с обратной связью с заказчиком. Именно измерение анализа улучшение определяет постоянно улучшение менеджмента качества.

5) Системный подход к менеджменту. Организация взаимосвязи процессов в единую систему с целью максимальной эффективностью для достижения целей.
6) Постоянное улучшение.
7) Принятие решение основанных на фактах. Любые решения должны приниматься после качественного анализа данных и информации, полученных из внешних источников.
8) Взаимовыгодные отношения с поставщиками.

С точки зрения разработки ПО ISO 9126-1 – ISO 9126-4:

1) Характеристики качества.
    a. Функциональные возможности. Пригодность продукта для применения по назначения. Взаимодействие со средой
    b. Защищенность
    c. Эффективность. Временная и ресурсная.
    d. Применяемость. Понятность функции документации, простота использования и возможность изучения функционала.
    e. Сопровождаемость. Анализируемость. Возможность модификации комплекса. Тестируемость.
    f. Мобильность. Простота инсталяции и замещаемость компонента.
    g. Внешние метрики качества
        i. Категорийные. Описание функциональных возможностей программы наиболее адекватным способом.
        ii. Количественные. Измерение надежности эффективности
        iii. Качественные. Определяет сопровождаемость программы, мобильность и практичность.
2) Внешние метрики качества. Полученный продукт соответствует спецификации и требованиям заказчика. Чаще всего анализируется соответствие заявленному ТЗ.
3) Внутренние метрики качества. Проверяет качество выполнения внутренних этапов процесса создания ПО. Оценивается его жизненный цикл.
4) Метрики качества в использовании. Состоит в оценке покупателями или заказчиками работы поставщиков, разработчиков, менеджеров, инженеров по сопровождению. В данном разделе обосновываются и комментируются показатели сферы использования программы с целью максимального достижения заявленных потребностей пользователя с учетом использования минимальных ресурсов.

Основной функцией для оценки является выбор и обоснование показателей качества. Обычно выбираются 2 показателя:

- Обоснование набора исходных данные.
- Обоснование набора шкал и метрик.

Общая схема оценки характеристик:

- Установка исходных требований для оценки.
- Селекция метрик.

- Планирование и проектирование процессов оценки.
- Выполнений измерений.

CMM – Capability Maturity Model

Стандарт был разработан в 90-х года в Америке. Основной проблемой считалось, то что предприятия являющиеся не зрелыми, не способными создавать приемлемых продуктов, не могут быть выпущены на рынок товаров, которые они не смогут качественно производить.

- Процедуры создания ПО
- Процедуры управления.

Любое предприятие должно пройти 5 уровней зрелости:

1) Начальный. Организация не может обеспечивать устойчивый процесс разработки ПО. Отсутствует культура управления. Проекты одноразовые. Отсутствует система согласованности проектов.
2) Повторяемый. Существуют политики управления и процедуры их выполнения. Эффективный процесс характеризуется наличию документов, возможности обучение персонала. Организации второго уровня могут создавать проекты на основе ранее созданных, за счет чего повышается продуктивность.
3) Определенные. Появление стандарта по разработке. На определенном уровне процесс разработки документирован и в документации включены как процессы интегрирование проектов. При помощи менеджеров техническому персоналу доводится требования о выполнений определенных операций. На данном уровня осуществляется координация производственных процессов. Руководящее звено обязано владеть навыки выполнения определенных ролей.
4) Управляемый. Устанавливается количественный показатель качества. Как для ПО, так и для процессов его разработки. В ходе проектов контроль надо процессами сужается до количественных пределов. Риски с обучение персонала управляемы.
5) Оптимизирующий. Мечта. Предполагается непрерывное совершенствование продукта.

SPICE

Разработан международным комитетом стандартизации. Рабочей версии стандарта вышла спустя 10 лет. Основан на более ранних стандартах.

Уровни процессов:

1) Процесс не выполняется. Требуются максимальные материальные затраты для запуска разработки продукта. Покупка оборудования и т.д.
2) Выполняемый процесс. Существует инфраструктура, позволяющая выполнить заявленный функционал. Есть все необходимые ресурсы.
3) Управляемый процесс. Предполагает наличие ресурсов для создания продуктов, частичную реализацию функционала с возможность прогнозирования получения результатов. Возможность создания продуктов в указанные сроки.
4) Установленный процесс. Предполагает опыт организации в разработке требуемого функционала, наличие внутренних документов, наличие и отслеживание ресурсов для организации процессов.
5) Предсказуемый процесс. Предполагает наличие инструментов измерения производства и рычаги его управления. Возможность до обучения персонала.
6) Оптимизирующий процесс. Мечта. Улучшение продукта.