1. **ИТ-проектом** - процесс, направленный на создание уникального продукта или услуги, связанный с модернизацией или интеграцией информационных систем в бизнес-процессы организации.

**Понятия**: задачи, ресурсы,

2. **Сетевой червь** – программа, способная к самораспространению путем многократного самокопирования и передаче в компьютерных сетях. Могут различаться: 1) способом передачи своих копий; 2) способом активации на компьютере; 3) способом проникновения в систему; 4) способом маскировки.

3. **Троянская программа** - компьютерная программа, скрытно осуществляющая действия, которые не осуществлял пользователь. «Классический» троян запускает скрытую внутри какой-либо легальной программы несанкционированную функцию, обеспечивающую выполнение действий, не предусмотренных автором легальной программы.

4. **Винлокер** – это вредоносная программа, цель которой заблокировать или затруднить работу с компьютером. Ее используют злоумышленники для получения вознаграждения за разблокирование. По сути это программа-вымогатель (смс-вымогатель). Блокирование операционной системы обычно осуществляется при помощи баннера, занимающего практически весь экран, и который невозможно убрать. (вид троянской программы)

5. **Необходимые условия применения измерительных методов**: 1) Наличие программы для измерительного исследования; 2) Наличие реальной вычислительной системы для прогона программы; 3) Наличие аппаратных или программных средств для проведения измерений; 4) Сниженные до приемлемого уровня изменения, вносимые в функционирование системы в процессе проведения измерений.

6. **Назначение измерительных методов**: 1) Измерение параметров потребления программами ресурсов вычислительной системы; 2) Предварительное измерение параметров системы для имитационных или аналитических моделей программ перед их последующим использованием; 3) Проверка адекватности имитационных или аналитических моделей и методов расчета характеристик выполнения программ по результатам моделирования.

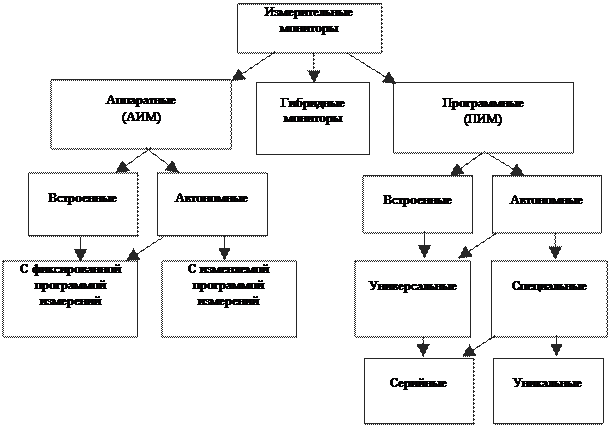
7. **Компоненты схемы измерений**: 1) Исследуемая ВС с установленными программами; 2) Средства регистрации параметров потребляемых ресурсов при выполнении данной рабочей нагрузки; 3) Архив для хранения результатов многочисленных измерений; 4) Обработка результатов изменений некоторой ВС; 5) Рабочая нагрузка - одна или несколько программ, или наборов данных для получения статистики проводимых измерений.

8. АИМ подразделяются на встроенные и автономные.

**Встроенные АИМ** - включаются в аппаратуру системы заводом изготовителем. Основное назначение встроенного АИМ - проверка, контроль и настройка ВС, также может использоваться и для измерения параметров программ. (фиксированный набор точек)

**Автономные АИМ** подключаются к измерительным точкам извне системы через специальные разъемы и могут задавать любые точки доступные для измерений.

9. **Виды**: аппаратные, гибридные и программные. Аппаратные и программные делятся на встроенные и автономные.



ИМ **нужны** для отслеживания и регистрации измерений параметров.

10. **Статическое тестирование** - ручное тестирование программ, начиная со стадии формирования требований к программе. На стадии кодирования некоторую часть маршрутов исполнения тестируют вручную, то есть без исполнения кода. Анализируют структуру, все связи программы, ее входные и выходные данные.

11. **Динамическое тестирование** – вид ручного тестирования, который подразумевает достаточно полную структурную и функциональную проверку выполнения программы. Вручную моделируется процесс выполнения программы на заданных исходных данных. Включает в себя проверку кода на следование набору правил и проведение модульного тестирования.

12. **Этапы формирования эталонов для тестирования**: 1) Использование аналитических выражений. 2) Использование моделирования на ЭВМ. Тестируемый объект создается иначе, чем оригинал. 3) Использование результатов испытаний предшествующих вариантов программ.

**Эталоны для тестирования**: 1) формализованные правила – требования различных стандартов и требования языков и технологий программирования (описание алгоритма, описание данных, структуры модулей); 2) программные спецификации (на модули, на данные, на взаимодействие модулей); 3) тесты (статические и динамические).

13. Классификация анализа вредоносных программ.

14. Этапы изучения вредоносного программного обеспечения и их описание.

15. Основными инструментами для исследования программ являются дисассемблеры и отладчики.

**Дизассемблер** - программа, преобразующая код программы в код на языке ассемблера. **Интерактивный дизассемблер** - программа, тесно взаимодействующая с пользователем в процессе дисассемблирования.

**Отладчик** - программа, предназначенная для анализа поведения другой программы, обеспечивающая остановку в указанных точках и позволяющая просматривать (редактировать) содержимое ячеек памяти, регистров процессора и команды программы. **Эмулирующий отладчик** - отладчик, который самостоятельно интерпретирует и выполняет команды программы (без использования реального процессора).

16. **Предназначение**: мониторинг и слежение за работой всей файловой системы, а также предоставление подробного описания событий в реальном времени. С помощью **файлового монитора** взломщик может отследить работу защищенной программы с файлами и обнаружить ключ, хранящийся в некотором файле.

17. Методы, применяемые для обнаружения модифицированного кода:

* подсчет контрольных сумм критических участков;
* использование контрольной суммы всего кода для расшифровки некоторого фрагмента;
* многопроходная расшифровка кода с ключом, вычисляемым на основе контрольной суммы всего кода либо критического участка;
* использование корректирующих кодов, позволяющих определить местоположение контрольного байта;
* контроль времени выполнения критического участка по сравнению с эталонным временем;
* контроль относительного времени выполнения участка программы (относительно другого участка).

**Преимущества**: эффективность, надежность, автоматизация.

**Недостатки**: могут не сработать, если участок кода не является критическим; сложность реализации и использования.

18. **Отладчик** - программа, предназначенная для анализа поведения другой программы, обеспечивающая остановку в указанных точках и позволяющая просматривать (редактировать) содержимое ячеек памяти, регистров процессора и команды программы. **Эмулирующий отладчик** - отладчик, который самостоятельно интерпретирует и выполняет команды программы (без использования реального процессора). **Отладчики с неполной эмуляцией** интерпретируют только некоторые команды, а остальные выполняют на реальном процессоре.

19. **Трассировка** - это пошаговое выполнение программы. **Назначение**: позволяет программистам анализировать процесс выполнения кода и искать ошибки. **Принцип работы**: устанавливается специальный флаг трассировки (TF), который приводит к генерированию после каждой команды исключительной ситуации - прерывания int 1 (трассировочное прерывание). При этом в стеке сохраняются значения регистра флагов и регистра IP (указатель на следующую выполнимую команду).

20. **Назначение**: для остановки выполнения программы в определенный момент и просмотра того, как она работает. Идея механизма заключается во внесении в программный код специального однобайтового кода (0xСС) - так называемой **контрольной точки останова**. Во время выполнения программы при достижении контрольной точки останова возникает исключительная ситуация - прерывание int 3. В этот момент процессор останавливает работу программы для дальнейших распоряжений пользователя. Для того, чтобы позже продолжить работу программы с точки останова, в стеке запоминаются значения регистра флагов, регистра CS (указатель текущего кодового сегмента) и регистра IP (указатель на следующую выполнимую команду).

21. **Тестирование ПО** – процесс исследования, испытания программного продукта, целью которого является проверка соответствия между реальным поведением программы и ее ожидаемым поведением. **Методы:** инспекция кода, черный, белый, серый ящик, тестирование модели.

22. **Инспекция кода** - систематическая проверка исходного кода программы с целью обнаружения и исправления ошибок, которые остались незамеченными в начальной фазе разработки. **Целью** просмотра является улучшение качества программного продукта и совершенствование навыков разработчика; нахождение багов и ошибок.

23. **Тестирование моделей**. Объект тестирования – не сама система, а ее модель, спроецированная формальными средствами. Тестировщик получает в свое распоряжение достаточно мощный инструмент анализа обшей целостности системы. На модели можно создать такие ситуации, которые невозможно создать в тестовой лаборатории для реальной системы. Работая с моделью программного кода системы, можно анализировать свойства, оптимальность алгоритмов кода или устойчивость системы.

24. **«Черный ящик» (структурный подход).** Тестировщику доступны лишь требования на систему и сама система, исходные тексты скрыты. **Основная задача** – последовательная проверка соответствия поведения программы ее требованиям. Обычно находят два типа проблем: несоответствие поведения системы требованиям; неадекватное поведение в ситуациях, непредусмотренных требованиями.

25. **«Стеклянный/белый ящик».** У тестировщика помимо требований к системе и самой системы, есть доступ к ее текстам. Тестировщик может увидеть соответствие требованиям участков программы и определить, на весь ли код есть требования. Код, не покрытый требованиями, является потенциальным источником неадекватного поведения системы. Кроме того, прозрачность системы позволяет углубить анализ ее участков, вызывающих проблемы.

26. **Эмулирующий отладчик** - отладчик, который самостоятельно интерпретирует и выполняет команды программы (без использования реального процессора). Они самостоятельно (без помощи процессора) интерпретируют и выполняют команды исследуемой программы. Против таких отладчиков бессильны любые приемы противодействия, поэтому сегодня уже мало кто решается противодействовать отладчику и включать в свое приложение антиотладочный код.

27. **Методы проникновения вирусов на ПК**: электронные почтовые сообщения; зараженные веб-сайты; хакерские атаки из Интернета; файловые серверы; локальные сети; пиратское программное обеспечение; многопользовательские компьютеры.

28. **Вредоносное программное обеспечение** − ПО, в результате действия которого в компьютерной системе осуществляются непредусмотренные пользователем действия, наносящие вред ему или другим субъектам.

**Виды ВПО**:

* **Компьютерный вирус** – самовоспроизводящаяся программа, которая может присоединяться к другим программам и файлам, способная приводить к нарушению целостности и доступности информации, а также вызывать снижение эффективности работы или повреждение компьютерной системы.
* **Сетевой червь** – программа, способная к самораспространению путем многократного самокопирования и передачи в компьютерных сетях.
* **Логическая бомба** − программа, выполняемая периодически или при наступлении определенных условий.
* **Троянская программа** - компьютерная программа, скрытно осуществляющая несанкционированные пользователем действия.
* **Архивная бомба** − специальный вид архива, заполняющий память компьютера большим количеством бесполезных данных, образующихся при их разархивировании.
* **Программы-шпионы** – программы, предназначенные для скрытого сбора и передачи информации о пользователе компьютера.

29. **Метрика измерений** – набор стандартов измерений, по которым можно понять степень качества продукции.

**Виды**: метрики производительности, метрики качества продукции и технические характеристики продукта.

30. **Репозиторий** – актуализируемое хранилище данных, в котором сохраняется история изменений. **Локальный репозиторий** — репозиторий, расположенный на локальном компьютере разработчика в каталоге. **Удаленный репозиторий** — репозиторий, находящийся на удаленном сервере. Это общий репозиторий, в который приходят все изменения, и из которого забираются все обновления. **Коммит** - зафиксированное состояние репозитория, сохранение изменений. **Ветка** — отдельная история изменений в рамках одного репозитория, можно создавать несколько веток. **Слияние** — слияние изменений из какой-либо ветки репозитория с любой веткой этого же репозитория. **Пул** — получение последних изменений с удалённого сервера репозитория. **Пуш** — отправка всех неотправленных коммитов на удалённый сервер репозитория.

31. **Рабочая копия** – копия репозитория, которая содержит все изменения, внесенные в код проекта. Она позволяет программистам работать над проектом локально, внося изменения и просматривая их результаты, не затрагивая при этом основной репозиторий.

32. **Тестирование ПО** – процесс исследования, испытания программного продукта, целью которого является проверка соответствия между реальным поведением программы и ее ожидаемым поведением. **Характеристика**: заранее планируется, проводится независимым тестировщиком, начинается до утверждения требований, завершается созданием отчета о тестировании. **Виды**: модульное, интеграционное, системное, выходное и приемочное.

33. **Система управления версиями** – ПО, предназначенное для работы с постоянно изменяющейся информацией. **Возможности**: поддержка хранения файлов в репозитории; поддержка истории версий файлов; нахождение конфликтов при изменении файлов; обеспечение синхронизации при работе в команде; отслеживание авторов изменений. **Применяется** при коллективной разработке ПО.

34. Делятся на локальные, централизованные и распределенные.

**Локальные СКВ** – ПО на локальном компьютером, имеют простую БД, которая хранит записи о всех изменениях в файлах, осуществляя тем самым контроль версий.

**Централизованные СКВ** используют единый сервер, содержащий все версии файлов и некоторое количество клиентов, которые получают файлы из централизованного хранилища, синхронизируя изменения с ним.

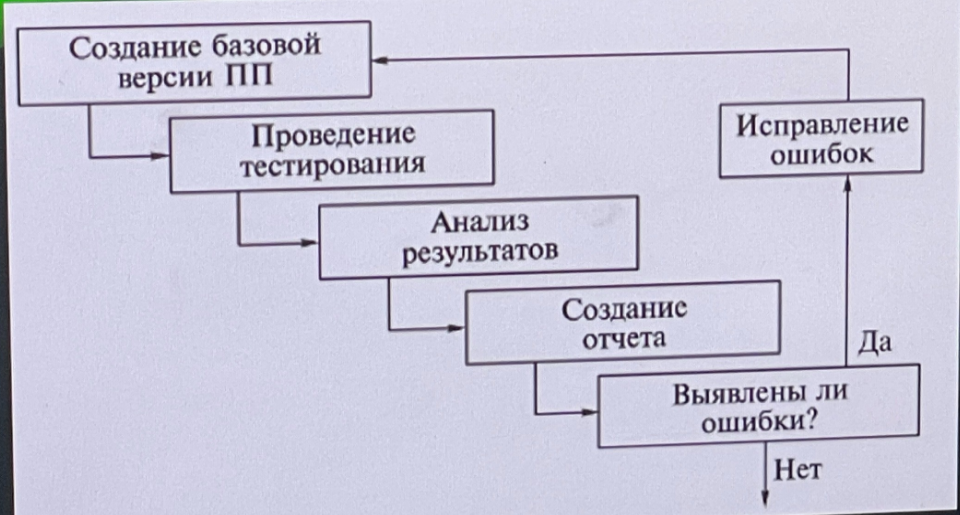
**Распределенные СКВ** клиенты не просто скачивают снимок всех файлов (состояние файлов на определенный момент времени), они полностью копируют репозиторий. Есть возможность создавать изменения.

35. **Subversion (SVN)** – одна из наиболее известных систем контроля версий. Подобные системы позволяют узнать, кто, когда и как именно вносил изменения в файлы проекта. Облегчает групповую разработку.

36. **Уровни тестирования**: **модульное** (осуществляет сам разработчик), **интеграционное**, **системное**, **выходное** (завершающий этап, на котором проверяется готовность ПП к поставке заказчику, проверка инсталляции и документации), **приемочное** (выполняется с представителями заказчика, проводится организацией, отвечающей за инсталляцию, сопровождение программной системы и обучение конечного пользователя). Первые 4 выполняются внутри организации (2-4 – независимым тестировщиком).

**Цикл тестирования** – совокупность действий, выполняемых тестировщиком с момента передачи базовой версии ПП тестировщику для 2, 3 и 5 уровня до момента успешного завершения тестирования.

Создание базовой версии ПП => Проведение тестирования => Анализ результатов => Создание отчета => Выявлены ли ошибки? => (Да) Исправление ошибок => Начало



37. Точно настроив интегрированную среду разработки по своему вкусу, можно сохранить эти настройки на будущее. Для этого можно **экспортировать** настройки интегрированной среды в файл, а затем **импортировать** их для того, чтобы во всех установленных системах были одинаковые настройки.

38. **Обратное проектирование** - процесс преобразования в модель кода, записанного на каком-либо языке программирования.

**Применяемые способы**: 1) FirstLook – приближенная пробежка по телу программы. 2) Detailed Analysis – детальный анализ проекта. 3) RoundTrip – комбинация двух вышеперечисленных способов. Позволяет безболезненно строить и перестраивать разрабатываемые приложения по принципу круговой разработки.

**Процесс обратного проектирования** делится на два этапа: анализ и генерацию модели. На первом этапе производятся все подготовительные операции по анализу текста программы на отсутствие синтаксических ошибок. Второй этап – преобразование кода в модель.

39. **Применение и функции обратного проектирования**: понимание, как работает программа, какие данные использует, куда и что отправляет; анализ на вирусы и поиск уязвимостей (можно специально заражать компьютеры); разработка модов для игр; изучение программ, документация к которым утеряна.

40. **Анализ обратного проектирования** производится при помощи различных программ, например, IDA. Что программа делает: анализ и преобразование в визуальную модель классов и структур; генерация связей в модели междц классами или структурами; нахождение комментариев и они переходят к строкам, к которым они относятся.

41. **Модульное тестирование** – процесс проверки отдельных программных процедур и подпрограмм, входящих в состав программ или программных систем. **Элементы**: синтаксическая проверка (на наличие синтаксических ошибок в коде), проверка соответствия стандартам кодирования, технический обзор программного кода. **Результат**: все измененные модули и наборы тестов хранятся в БД проекта.

42. **Интеграционное тестирование** – проводится для проверки совместной работы отдельных модулей и предшествует тестированию всей системы как единого целого. **Элементы**: проверка функциональности (на соответствие требованиям), проверка промежуточных результатов (результатов и файлов на наличие и корректность), проверка интеграции (проверка того, что модули передают друг другу информацию корректно). **Результаты**: включаются в отчет о ходе тестирования. Ошибки, выявленные на этапе, заносятся в БД ошибок.

43. **Системное тестирование** – предназначено для проверки программной системы в целом, ее организации и функционирования на соответствие спецификациям требований заказчика. **Элементы**: граничное (в граничных условиях), прогоночное (проверка всех функциональных характеристик реальной работы системы), целевое тестирование (проверка в целевой платформе), проверка документации (на корректность) и другие виды тестирования. **Результаты**: включаются в отчет о ходе тестирования. Ошибки, выявленные на этапе, заносятся в БД проекта.

44. **Локальный репозиторий** — репозиторий, расположенный на локальном компьютере разработчика в каталоге.

**Функции локального репозитория**: 1) Хранение истории изменений. 2) Ветвление и слияние. 3) Синхронизация с удаленным репозиторием. 4) Откат изменений.

45. **Метод "серый ящик"** - это метод тестирования программного обеспечения, который комбинирует элементы методов "белый ящик" и "черный ящик". При использовании этого метода тестировщик имеет доступ к некоторым внутренним аспектам программы, но не обладает полным знанием о ее внутренней структуре.

**Процесс**: анализ требований программы, выбор тестируемых компонентов и их изучение, разработка и выполнение тест-кейсов, анализ результатов и если необходимо повторное тестирование.

46. **Для получения доступа** к GitHub, нужно зарегистрироваться на сайте и создать репозиторий. Для получению доступа через консоль, нужно ввести две команды: git config --global user.name “логин”, git config --global user.email “почта”.

**Принцип и функции**: GitHub позволяет интегрировать совместную работу непосредственно в процесс разработки. Работа организована по репозиториям, в которых разработчики могут устанавливать требования или давать указания участникам команды. Можно создавать ветви для работы с обновлениями, фиксировать изменения, чтобы сохранять их, открывать запросы на вытягивание, чтобы предлагать и обсуждать изменения, и выполняют слияние запросов на вытягивание после их согласования.

47. **Организация разработки** – это функция, позволяющая создать центральное место, где члены команды могут получать доступ к репозиториям и другим ресурсам и управлять ими.

Для этого на GitHub нужно зайти в раздел мои организации и создать новую. Необходимо добавить людей; Обеспечить возможность отправки версий и их слияния; Обеспечить отслеживания версий; Реализовать возможность комментариев.

48. **Уровни доступа GitHub**:

1. Pull Only: выборка и слияние с другим репозиторием или локальной копией. Доступ только для чтения.

2. Push and Pull: (1) наряду с обновлением удаленного репозитория. Чтение + Запись.

3. Push, Pull & Administrative: (1), (2), добавляются права создания команд, а также удаление аккаунтов организации. Чтение + запись + доступ администратора.

49. **3 способа слияния копий репозитория**:

1) Create a merge commit – все коммиты будут добавлены в главную ветку через отдельный комит.

2) Squash and merge – все комиты ветки будут соединены в один, и он будет добавлен в главную ветку.

3) Rebase and merge – все комиты будут добавлены в главную ветку, как ее комиты

50. Для создания кода в репозитории, GitHub предлагает две модели для создания копии репозитория, в котором можно создать новый код:

* Модель Fork & Pull – используется в общедоступном репозитории, на который у вас нет push-доступа
* Share Repository Model – используется в частном репозитории, на который у нас есть push-доступ. В этом случае форк не требуется.

51. **Архитектура MVC (Model-View-Controller)** - паттерн проектирования, который разделяет приложение на три основных компонента: **модель** (Получает данные от контроллера, выполняет необходимые операции и передаёт их в вид), **представление** (Получает данные от модели и выводит их для пользователя) и **контроллер** (Обрабатывает действия пользователя, проверяет полученные данные и передаёт их модели).

52. **При проверке на стороне клиента** проверяется модель представления. В модели MVC проверка осуществляется при помощи ASP.NET. Для этого есть класс DataAnnotations, в котором есть множество атрибутов проверки. Наиболее распространенные – Required (обязательное значение), StringLength (максимальное количество символов в строке), RegularExpression (ограничивает набор вводимых символов) и Range (ограничивает диапазон значений), MinimumLength, DataType (позволяет выбрать тип данных). Эти атрибуты указываются при создании полей класса.

При запуске приложения, функция jQuery проверяет правильность заполненных данных. Если данные были введены неверно, то jQuery об этом сообщит.

53. **JavaScript** является одним из самых популярных языков программирования в мире веб-разработки. У него есть множество инструментов и сред разработки, которые помогают разработчикам эффективно создавать и отлаживать программы на JavaScript.

**Одним из самых популярных IDE** для разработки на JavaScript является Visual Studio Code. Он предлагает множество полезных функций, таких как подсветка синтаксиса, автозаполнение кода, отладка и интеграция с системами контроля версий. Visual Studio Code также имеет широкий выбор плагинов и расширений, включая дополнительные функции для работы с JavaScript.

54. **Проверка на стороне сервера** – это проверка, которая возникает на сервере после отправки данных. Серверный код используется для проверки данных перед их сохранением в базе данных. Если данные не проходят проверку валидности, ответ отправляется обратно клиенту, чтобы сообщить пользователю, какие исправления должны быть сделаны.

На сервере проверяется правильность введенных данных.

Преимущество выполнения проверки на стороне сервера по сравнению с проверкой на стороне клиента заключается в том, что проверку на стороне клиента можно обойти / манипулировать ею: Конечный пользователь мог бы отключить JavaScript.

55. **Библиотека** — набор готовых функций, классов и объектов для решения каких-то задач. Часто подключается отдельно.

Доступ к членам класса в библиотеке классов осуществляется путем создания экземпляра класса или обращения к статическим членам.

При создании экземпляра класса можно получить доступ к его полям и свойствам (присваивать им значения или считывать) и использовать его методы.

Для доступа к статическим членам класса в библиотеке классов не требуется создавать экземпляр класса.

56. **Язык Ассемблера** - низкоуровневый язык программирования, является звеном между высокими языками программирования и машинным кодом. Возможности: 1) Преобразование языков программирования в их эквиваленты на машинном языке; 2) Преобразование символических операндов в эквивалентные им машинные адреса; 3) Построение машинных команд; 4) Формирование и запись объектного модуля; 6) Выдача листинга.

**Достоинства**: Максимально оптимальное использование средств процессора, использование меньшего количества команд и обращений в память, и как следствие — большая скорость и меньший размер программы.

**Недостатки**: трудный синтаксис, из-за чего на ассемблере трудно писать; Недостаток переносимости программы между различными компьютерными архитектурами; Для запуска длинных программ, написанных на ассемблере, требуется больший объем или память компьютера

57. **Сегментная адресация** - это метод организации памяти компьютера, при котором каждый сегмент получает свой уникальный адрес. Каждый сегмент представляет собой непрерывную область памяти определенного размера.

**Примеры применения**: 1) Многозадачные операционные системы: сегментная адресация позволяет разделить память на отдельные сегменты для каждого запущенного процесса или задачи. 2) Базы данных: сегментная адресация позволяет организовать данные в базе данных в виде сегментов, что облегчает быстрый доступ к определенным записям или частям базы данных. 3) Графические системы: сегментная адресация широко используется в графических системах для организации видеопамяти.

58. **Дизассемблер** - программа, преобразующая код программы в код на языке ассемблера.

**Возможности**: анализ машинного и исходного кода, чтение и разбор бинарных файлов.

**Преимущества**: можно исправлять ошибки, использовать для обратного проектирования, изучение архитектуры программы.

**Недостатки**: сложный язык ассемблера, дизассемблеры не всегда могут восстановить исходный код.

59. **Мониторинг работоспособности системы контроля версий** (СКВ) позволяет отслеживать состояние и производительность системы, а также обнаруживать возможные проблемы и устранять их вовремя. Главная цель мониторинга - обеспечить непрерывную доступность и работу СКВ для команды разработчиков.

**Инструменты:** Git, Mercurial, CVS (Concurrent Versions System), Bazzar.

60. **Коллективная разработка** - это процесс, в котором группа людей совместно работает над созданием или развитием какого-либо продукта или проекта. Она основывается на принципе совместного участия, взаимодействия и вклада всех членов коллектива.

**Есть две модели организации коллективной разработки**: 1) иерархическая модель – случай, когда есть начальник, который руководит процессом и своими подчиненными; 2) модель группы – разработка, когда обязанности распределены между разработчиками, нет централизованности.