1. **Технология разработки программного обеспечения(ПО)** - представляет собой комплекс организационных мер, производственных процессов, операций приемов, методов и методик, инструментальных средств, направленных на разработку программных продуктов высокого качества в рамках отведенного бюджета и в заданный срок.
   Декомпозиция СТС - разбиение её на ряд связанных между собой общими целями функционирования подсистем.
   Системообразующая роль встроенных в системы ЦВМ и ПО заключается в: объединении подсистем в единое целое, взаимодействии и координации систем, поддержке сложных функциональных задач
2. **Особенности коллективной разработки ПО СТС:**
   - Разделение работы и ответственности
   - Согласование и разработка связей по информации и управлению между частями ПО, которые обеспечивают работу отдельных физических процессов.
   - Защита критических ресурсов: Параллельное исполнение программ и процессов, разработанных различными людьми, создает угрозы совместного использования критических ресурсов.
   - В больших коллективах неизбежны кадровые изменения, поэтому необходимо тщательно документировать программный продукт.
   - Важно подтвердить, что разработанное по частям ПО действительно работает как единое целое.

   ПО и управление системой в нештатных ситуациях: ПО в сложных технических системах (СТС) играет двоякую роль: оно может быть источником ошибок, но также обладает возможностями логического анализа для обнаружения и парирования нештатных ситуаций (НС). НС возникают из-за отказов оборудования, ошибок ПО и отклонений от условий эксплуатации. Примеры успешного преодоления НС включают полет Ю. А. Гагарина, где было 11 НС, но благодаря избыточности и продуманным планам действий, полет прошел успешно. Разработка ПО для критических систем обязательно включает методы для выявления, идентификации и восстановления после ошибок и отказов.

3. **Проблема низкой производительности процессов разработки ПО. Как обеспечить растущие потребности в разработке ПО?:** Проблема недостаточной производительности процессов разработки ПО заключается в том, что прирост эффективности разработки отстаёт от роста потребностей в ПО
   Средство борьбы — повторное использование ранее разработанных программ и компонентов. В разработке ПО должны быть разработаны и доступны библиотеки повторно используемых программ.

4. **Проблема сложности ПО включает в себя несколько аспектов:**

   - **Сложность устройства ПО** (ПО сложно устроено)
   - **Сложность поведения ПО** (Сложно предсказать как себя ПО поведет)
   - **Технологическая сложность алгоритмов**

Средство борьбы - Структурирование, которое включает разбиение ПО на доступные для понимания человеком части.

Проблема борьбы с ошибками и обеспечения надежности ПО состоит из нескольких аспектов:

- Неизбежность ошибок в сложных программах
- Скрытые ошибки, которые проявляются только при определенных наборах исходных данных.

Средства борьбы:

- Для устранения скрытых ошибок необходимо проводить специальную работу по их выявлению, локализации и устранению, что называется отладкой.
- Использование объектно-ориентированного подхода к разработке ПО способствует уменьшению первичного потока ошибок.
- Конструирование ПО с устойчивостью к ошибкам. Это направление связано с автоматическим обнаружением ошибок встроенными в ПО средствами и их обработкой в процессе работы программы.

5. **Проблема изменяемости ПО заключается в следующем:**
   - Неизбежность изменения требований
   - Недостаточная определенность требований

   Средства борьбы:
   - Локализация изменений (важно, чтобы изменения были локализованы в определенных частях ПО)
   - Стратегия структуризации (необходимо заранее определить, где и каким образом могут потребоваться изменения)
   - Иерархическое структурирование (способствует локализации изменений)

   Проблема безопасности ПО заключается в следующем:
   - Уязвимость к атакам
   - Ограниченность традиционного подхода к безопасности (Проблема безопасности ПО не исчерпывается только предотвращением несанкционированного доступа и борьбой с вирусами. Ошибки в ПО все равно случаются и могут привести к значительным убыткам.)

   Средства борьбы:
   - Управление последствиями ошибок (Важно не только предотвращать ошибки, но и понимать, какие последствия они могут иметь.)
   - Внедрение в код защитных фрагментов, обеспечивающих обнаружение и парирование ошибок средствами самого ПО.
   - Минимизация ущерба от ошибок
6. **Оформленная технология разработки ПО** играет ключевую роль в современной индустрии программного обеспечения по нескольким причинам:
   - Обеспечение качества и соблюдение ограничений
   - Прозрачный и управляемый процесс

- Повышение производительности
- Легкость внедрения новых сотрудников
- Сертификация и привлечение заказов

7. **Три основных фактора, влияющих на качество ПО:**
   - Технология разработки ПО (ТРПО)
   - Материальные и временные ресурсы
   - Человеческий фактор

   Причины неудач программных проектов:
   - Плохо определенные и часто меняющиеся требования к ПО
   - Неправильная организация разработки и слабое управление изменениями проекта
   - Неосведомлённость управляющего проектом о точном состоянии проекта и плохая координация работ
   - Слабое взаимодействие между заказчиком и разработчиком
   - Отсутствие необходимых ресурсов (людских, материальных, временных)
   - Неопределенные и неосвоенные процессы разработки программного обеспечения

   Характеристики качества ПО, важные для пользователя:
   - Функциональные возможности (ПО должно выполнять требуемые функции)
   - Надежность (ПО должно быть безошибочным и устойчивым к ошибкам)
   - Эффективность
   - Практичность
   - Сопровождаемость (ПО должно быть способным к развитию, легко обновляться)
   - Мобильность
   - Безопасность (не наносить неприемлемого ущерба в случае проявления ошибки)

8. **Системный подход к разработке ПО** предполагает рассмотрение всей проблемы целиком, а не только отдельных аспектов. Этот подход реализуется в двух измерениях: во времени и в пространстве.

   - **Во времени**: Системный подход охватывает весь жизненный цикл ПО, начиная от момента формирования неудовлетворенной потребности в ПО до момента её разрешения.
   - **В пространстве**: Системный подход предусматривает рассмотрение разрабатываемого ПО как части системы. На основе изучения информационных потребностей системы формулируются цели и набор функций ПО, а также документируются требования к ПО.

   Участники процесса разработки и их роли:

   - **Заказчик (покупатель, приобретающая сторона)**:

    - Формулирует требования к ПО.
    - Выбирает поставщика или разработчика.
    - Проводит договорную работу по оплате продукта.
    - Определяет цели проекта, требования к нему, сроки и бизнес-аспекты.
- **Поставщик (продавец, подрядчик)**:
    - Поставляет программный продукт заказчику.
    - Принимает решение о разработке ПО собственными силами, с привлечением субподрядчика или закупкой у третьих лиц.
- **Разработчик (исполнитель)**:
    - Разрабатывает программный продукт в соответствии с этапами жизненного цикла ПО.
- **Эксплуатационщик (пользователь, оператор)**:
    - Осуществляет эксплуатацию ПО в соответствии с эксплуатационной документацией разработчика.
- **Сопровождающая организация или разработчик ПО**:
    - В течение всего времени эксплуатации ПО выполняет обязательства по модификации ПО.
    - Устраняет ошибки, решает возникающие проблемы.
    - Переводит ПО в новую аппаратную среду и т.п.

9. **Жизненный цикл программного обеспечения** (ПО) – это концепция, отражающая системный подход к разработке ПО. Он представляет собой совокупность последовательных этапов разработки, начиная от формирования требований и проектирования до реализации, тестирования, внедрения и сопровождения.

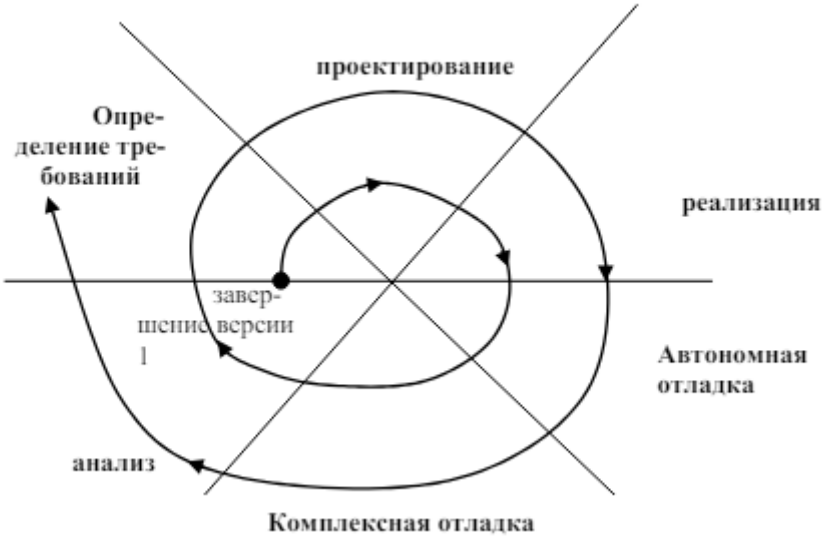
   Каскадная модель жизненного цикла ПО представляет собой последовательный тип разработки программного обеспечения, где каждый этап жизненного цикла должен быть завершен перед переходом к следующему. Этот тип модели хорош для проектов, где системные требования четко определены с самого начала.

   Вот **последовательность процессов разработки в каскадной модели**:

   - Анализ системных требований и области применения
   - Проектирование архитектуры системы
   - Анализ требований к программным средствам
   - Определение стратегии разработки программных средств
   - Менеджмент рисков разработки ПО
   - Проектирование архитектуры программных средств
   - Детальное проектирование программных средств
   - Конструирование программных средств
   - Комплексирование программных средств

- Квалификационное тестирование программных средств
- Менеджмент выпуска документации ПО
- Менеджмент повторного применения программ
- Сопровождение разработанного ПО в эксплуатации

10. **Спиральная модель** жизненного цикла ПО представляет собой итеративный и инкрементный подход к разработке программного обеспечения, который особенно полезен для проектов с неопределенными и изменяющимися требованиями. Она обеспечивает постепенное наращивание функциональности ПО и активное взаимодействие с заказчиком на каждом этапе.



11. Стандарты по разработке ПО: виды и значение

Виды стандартов:

1. **Стандарты процессов разработки**:
   - Устанавливают единые и обязательные требования к процессам разработки ПО.
   - Пример: ГОСТ Р ИСО/МЭК 12207 — описывает процессы жизненного цикла программных средств.
2. **Стандарты качества ПО**:
   - Определяют требования к качеству самого программного обеспечения.
   - Пример: ГОСТ Р ИСО/МЭК 9126 — устанавливает характеристики качества ПО.

Значение стандартов:

- **Интеграция и обобщение опыта**: Позволяют обобщить и интегрировать лучшие практики разработки ПО со всего мира.

- **Совместимость**: Обеспечивают совместимость ПО, созданного разными производителями.
- **Единая терминология**: Создают единую терминологию и подходы к разработке, что облегчает коммуникацию и понимание между разработчиками.
- **Передача передовых технологий**: Доводят передовые технологии разработки до всех разработчиков.
- **Гибкость и обновляемость**: Стандарты должны быть достаточно гибкими, чтобы не препятствовать техническому прогрессу.

Процессы разработки ПО в соответствии со стандартом ГОСТ Р 12207:

Процессы разработки ПО в контексте системы:

1. **Процессы приобретения и поставки ПО** (2 процесса):
   - Описание деятельности по приобретению и поставке ПО.

2. **Процессы организационного обеспечения проекта** (5 процессов):
   - Включают менеджмент жизненного цикла, управление качеством и другие организационные задачи.

3. **Процессы проекта** (7 процессов):
   - Включают планирование, управление решениями, управление рисками и т.д.

4. **Технические процессы** (11 процессов):
   - Охватывают анализ системных требований, проектирование архитектуры системы, реализацию, комплексирование системы, квалификационное тестирование и т.д.

Собственно процессы разработки ПО:

5. **Процессы реализации программных средств** (7 процессов):
   - Включают анализ требований к ПО, проектирование архитектуры ПО, конструирование ПО и т.д.

6. **Процессы поддержки программных средств** (8 процессов):
   - Включают менеджмент документации ПО, обеспечение качества ПО, верификацию ПО и т.д.

7. **Процессы повторного применения программных средств** (3 процесса):
   - Описание деятельности по повторному использованию программных средств.

12. Виды документов, выпускаемых по этапам разработки ПО системы.

Можно выделить три категории документов на ПО:

- Документация разработки или проектные документы

- Документация на программную продукцию (конструкторская, эксплуатационная, ТУ)

- Документация управления проектом

Проектные документы: Проектная документация ПО при коллективной разработке ПО является средством связи между различными разработчиками, обеспечивающим реализацию совместных требований к программному продукту на всех стадиях его ЖЦ, средством согласования структуры и принятых параметров разработки между всеми её участниками.

Конструкторская документация на ПО: Документация на программную продукцию, систему или изделие, по которой они изготавливаются, собираются и испытываются , называется конструкторской. Конструкторская документация на ПО - тексты программ ПО или код программ ПО, загружаемый на исполнение в ЦВМ.

Эксплуатационная документация на ПО: Для эксплуатации необходимо наличие эксплуатационной документации на программную продукцию, в которой приводится техническое описание и оговариваются действия эксплуатационного персонала, по запуску ПО в работу, по управлению ПО и системой в процессе работы, по останову ПО и системы, по методам перезапуска ПО в случае ошибочных действий персонала, по действиям в нештатных ситуациях и т.п.

Документация управления проектом: Определяет круг обязанностей и ответственности участников разработки, план-график разработки, определяет контрольные точки, делающие видимым процесс разработки ПО для руководителей и участников разработки, отчетные документы в контрольных точках.

13. Методы ранней верификации и отладка ПО.

Верификация — это процесс, направленный на контроль качества программного обеспечения и обнаружение в нем ошибок.

Методы верификации:

- Статический анализ программ
- Тестирование
- Верификационный мониторинг

Отладка ПО — это процесс поиска и устранения ошибок или «багов», выявленных в ходе тестирования. Задача отладки определить причины возникновения дефекта и его устранение.

14. «Тяжелые и легкие» технологии разработки ПО.

Тяжелые технологии разработки ПО - технологии с каскадной моделью ЖЦ. Они базируются на хорошо определенных требованиях к разрабатываемому ПО и жестко регламентированы, требуют документирования всех работ.
Легкие технологии (подвижные, адаптивных технологии) разработки ПО - Эти технологии имеют спиральную модель жизненного цикла и используются для разработки ПО, первоначальные требования к которому плохо определены и постоянно меняются с течением времени разработки.

15. Основные особенности легких технологий разработки ПО.

- Гибкость
- Простота
- Коллаборация
- Быстрые результаты
- Постоянное улучшение

Четыре принципа легких технологий: понимание, автоматизация, малые поштучные поставки и здравый смысл.
Границы применимости легких и тяжелых технологий разработки ПО могут быть определены на основе размера и критичности программного обеспечения. **Легкие** — небольшие проекты с невысокой степень критичности, где важнее быстрые результаты и гибкость. **Тяжелые** — крупные и критичные проекты с формальными процессами и регламентом.
Размер ПО и его критичность: Легкие — небольшие и некритичные, тяжелые — крупные и критичные.
Связь с «весом» технологии заключается в том, что для каждого конкретного проекта или типа ПО может быть оптимальным использование как легких, так и тяжелых технологий в зависимости от его характеристик, требований и контекста.

16. Различия в технологиях разработки трех типов ПО

Три типа ПО:

- Небольшие бизнес системы или программы для научных исследований.
- Сложные информационно справочные системы и системы сбора и обработки информации
- Встроенное ПО для управления сложными техническими критическими системами

Различия в технологиях разработки ПО:

Типы ПО / Методы И подходы	**Небольшие бизнес системы, небольшие программы для научных исследований**	**Сложные информационно справочные системы и системы сбора и обработки информации**	**Встроенное ПО для управления сложными техническими критическими системами**
Модели жизненного цикла	Многоитеративные и инкрементные подходы	Каскадная модель Спиральная модель для частей ПО	Каскадная модель

	спиральной модели.		
Работа в реальном времени	Не требуется	Не требуется	Требуется
Планирование и управление разработкой	Адаптивное планирование по мере надобности,	Базовое заблаговременное планирование.	Детальное планирование и контроль хода разработки
Требования к ПО	Меняющаяся спецификация требований	Формальная,меняющаеся спецификация требований	Формальная не меняющаяся спецификация требований
Критичность проявления ошибок в ПО	Не критично	Критично. Претензии пользователей к разработчику	Критично. Большой ущерб, ущерб экологии, ущерб для людей
Методы и характер проектирования (конструирование)	Простая архитектура. Проектирование часто совмещается с кодированием	Проектирование архитектуры. Неформальное детальное проектирование отдельных структурных элементов.	Проектирование безопасной многозадачной архитектуры. Проектирование аварийной защиты. Формальное документированное детальное проектирование.
Конструирование (кодирование)	Парное или индивидуальное программирование	Парное или индивидуальное программирование. Конструирование с защитой от ошибок. Обзоры кода по мере необходимости	Парное или индивидуальное программирование. Конструирование устойчивых к ошибкам программ. Формальные процедуры регистрации кода.
Тестирование и отладка (конструирование)	Разработчики отлаживают собственный код совместно с тестировщиками. Предварительная разработка тестов.	Разработчики отлаживают собственный код совместно с тестировщиками. Предварительная разработка тестов. Отдельная группа тестирования	Разработчики отлаживают собственный код. Предварительная разработка тестов. Отдельная группа тестировщиков. Использование динамической модели внешней среды для комплексной отладки

Тип («вес») технологии разработки	Легкие технологии	Средние технологии или комбинация легких и тяжелых технологий	Тяжелые технологии

17. Scrum —легкая технология разработки ПО.

Scrum —легкая технология разработки ПО. Scrum реализует итеративный и инкрементальный подход — то есть ориентируется на то, чтобы двигаться по пути разработки ПО поэтапно, последовательно создавая через определенные промежутки времени – итерации потенциально готовое к выпуску работающее ПО с новыми возможностями.

Структура процессов разработки ПО в Scrum:

- Владелец продукта информирует о запросах на выполнение работ и определяет список требований к ПО (product backlog).
- Встреча по упорядочиванию беклога, где определяются приоритеты и решения о дальнейшей работе.
- Планирование Спринта для определения конкретных задач на итерацию.
- Ежедневные оперативные совещания - летучки для обмена информации и поддержания состояния проекта.
- Подведение итогов Спринта, где команда представляет результаты и проверяет соответствие целям проекта.
- Ретроспектива Спринта для анализа процесса и улучшения эффективности.

Пять основных типов встреч в Scrum:

- Встреча по упорядочиванию беклога (Backlog Refinement Meeting).
- Планирование Спринта (Sprint Planning Meeting).
- Ежедневные оперативные совещания - летучки (Daily Stand-up Meeting).
- Подведение итогов Спринта (Sprint Review).
- Ретроспектива Спринта (Sprint Retrospective).

18. Легкая технология Kanban.

Kanban — это система постановки задач, при которой все этапы проекта визуализируются на специальной доске. Члены команды могут видеть текущее состояние задачи на любой момент времени.

Экстремальное программирование (Extreme Programming, XP) - это один из видов гибкой разработки ПО, набор принципов и приёмов, позволяющих повысить продуктивность работы разработчиков и улучшить качество ПО. Принципы XP – минимальность, простота, участие заказчика, короткий цикл, тесные взаимодействия разработчиков.

19. Итеративный характер проектирования ПО.

Итеративный характер проектирования ПО подразумевает постепенное уточнение и развитие проекта через серию итераций, где каждая итерация включает в себя цикл разработки, тестирования и уточнения результатов. Такой подход

позволяет улучшить качество и функциональность конечного продукта, а также своевременно обнаружить и исправить ошибки на более ранних стадиях разработки.

Стадии проектирования ПО:

- Формирование требований к системе
- Техническое предложение на систему или разработка концепции системы
- Разработка и утверждение технического задания (ТЗ) на систему
- Эскизный проект на систему
- Технический проект, на этой стадии выпускается рабочая документация

Архитектура (структура) ПО включает в себя качественное построение и управление иерархически организованными элементами системы, отображаемыми в виде графических схем, диаграмм и моделей, которые помогают понять и контролировать работу программного обеспечения.

20. Иерархическая структура ПО и иерархия управления системой.

Иерархическая структура ПО и иерархия управления системой позволяют разделить процессы управления на более мелкие и понятные компоненты, что упрощает разработку, поддержку и модернизацию системы.

Развитие системы и проведение изменений в иерархической структуре ПО Необходимо создавать возможность развития и изменений в ПО, изолируя изменения в определенных частях системы. Модернизация должна быть локализована и ограничена только необходимыми компонентами.

21. Цикличность решения задач управления в системах с ЦВМ.

Цикличность решения задач управления в системах с ЦВМ обусловлена необходимостью периодического сбора, обработки и выдачи информации для управления объектами или процессами. Это связано с тем, что объекты и процессы требуют постоянного контроля и коррекции, чтобы достичь требуемых результатов. Поэтому задачи управления в системах с ЦВМ исполняются периодически, повторяясь через определенный интервал времени.

22. Три источники многозадачности ПО.

1) Декомпозиция и необходимость коллективной разработки большого ПО, приводящие к разделению ПО на функционально ориентированные части и возможность параллельного выполнения задач.

2) Работа ряда задач в реальном времени, требующая быстрой реакции программного обеспечения на внешние сигналы и события.

3) Желание повысить временную эффективность работы ПО и избежать простоев оборудования ЦВМ, что также приводит к многозадачной работе.

Угрозы безопасности, вытекающие из многозадачности включают в себя возможность взаимоблокировки задач из-за конкуренции за ресурсы ЦВМ, проблему синхронизации задач, а также возможность удаления или повреждения файлов других задач в случае совместной работы нескольких задач.

Поддержка многозадачности со стороны ОС организуя параллельное выполнение нескольких программ на одном процессоре. ОС действует как изолятор и синхронизатор,

обеспечивая разделение ресурсов между пользователями, программами и устройствами, управляя процессами, обеспечивая доступ к файлам, обрабатывая системные вызовы и управляя прерываниями.

23) Многозадачность — это способность операционной системы (ОС) выполнять несколько программ и процессов одновременно. Она позволяет запускать параллельные задачи. Задачи, процессы и потоки - Процесс — это основная единица выполнения, представляющая собой контейнер, содержащий память, ресурсы, открытые файлы и потоки. - Поток — это легковесная единица выполнения в рамках процесса, которая делит ресурсы и память с другими потоками того же процесса. Контекст процесса и его применение: - Контекст процесса включает все необходимое состояние процесса в данный момент времени: содержимое регистров, память, счетчики команд и времени и прочее. - Контекст позволяет "заморозить" выполнение процесса, чтобы потом продолжить. Это важно при переключении процессов или обработке прерываний, когда необходимо временно приостановить один процесс. - Время переключения процессов, которое зависит от времени сохранения и восстановления контекста, является важным показателем эффективности ОС.

24. Временная диаграмма работы системы и ПО. Способы её реализации. Представление работы ПО в виде набора параллельных процессов.Временная диаграмма работы ПО в вариантах использования системы.

Временная диаграмма работы системы и ПО визуализирует параллельные процессы. ПО управляет подсистемами дискретно. Для её реализации используются вертикальные сечения, где создаются пакеты программ, и ведущая программа, или ПКФ. Многозадачный режим работы ПО предполагает параллельное выполнение программ на одном процессоре, с возможными информационными и управляющими связями между ними.

Временная диаграмма работы ПО в разных вариантах использования системы помогает увидеть, как ПО управляет подсистемами параллельно во времени. Каждый вариант использования имеет свою диаграмму, отображающую последовательность работы подсистем и дискретную работу ПО.

25) Обобщенная схема возможных вариантов совместного использования информации взаимодействующими процессами:

1. Связь между задачами: Обеспечивает обмен данными и синхронизацию между задачами.
2. Единый процесс с параллельными потоками:
   1. Плюс: Быстрый обмен данных в едином адресном пространстве.
   2. Минус: Больший риск ошибок синхронизации и управления памятью.
3. Отдельные процессы:
   1. Плюс: Повышенная безопасность через изоляцию памяти.
   2. Минус: Более медленный обмен данными.
4. Методы взаимодействия:
   1. Файловая система: Медленно, синхронизация на разработчике.
   2. Средства ОС: Например, каналы (pipe) в UNIX.

3. Разделяемая память: Самый быстрый метод, требует контролируемой синхронизации.

26. Реализация многозадачности за счет параллельных вычислений. Технология Open MP. Закон Амдала.

Многоядерные процессоры позволяют параллельную обработку задач. Open MP позволяет эффективно распределять работу между ядрами. Программа создает главный поток команд, а дополнительные потоки выполняют параллельные вычисления. Синхронизация необходима для правильного завершения выполнения. Несмотря на потенциальные выигрыши в производительности, есть затраты на загрузку кода и синхронизацию. Однако, с помощью Open MP можно значительно ускорить выполнение программ, особенно при обработке хорошо распараллеливаемых задач. Закон Амдала утверждает, что увеличение числа параллельных процессоров не приводит к пропорциональному ускорению выполнения задачи из-за необходимости последовательного выполнения части кода и наличия не распараллеливаемых участков программы. Эффективность увеличения числа процессоров ограничена долей не параллелизуемых операций. Это означает, что даже при большом числе процессоров время выполнения задачи уменьшается не более, чем в два раза, если половина программы не может быть распараллелена.

27) Критический ресурс ЦВМ

Критический ресурс — это ресурс, к которому может иметь доступ только один процесс одновременно для предотвращения конфликтов.

Основное правило заключается в том, что процесс не должен менять состояние критического ресурса, пока другой процесс имеет к нему доступ.

Задачи синхронизации процессов

1. Взаимное исключение
   - Обеспечивает, что только один процесс может находиться в своей критической секции в любой момент времени.
2. Читатели-писатели (Readers-Writers)
   - Допускает одновременное чтение данных несколькими процессами, но запись данных может выполняться только одним процессом.
3. Обедающие философы (Dining Philosophers)
   - Классическая задача распределения ресурсов, используемая для демонстрации проблем синхронизации.

Временная и логическая синхронизация

1. Временная синхронизация
- Планирование последовательности выполнения процессов во времени.
- Не всегда надежна из-за изменчивости времени выполнения процессов.

2. Логическая синхронизация
- Обеспечивает выполнение процессов в правильной последовательности, основываясь на логических условиях.
- Более надежная и безопасная по сравнению с временной синхронизацией, так как не зависит от времени выполнения.

28. Задачи синхронизации процессов. Взаимное исключение процессов. Использование мьютексов. Задача синхронизации «Читатели-писатели». Часто встречающиеся ошибки синхронизации.

Задачи синхронизации процессов включают планирование последовательности их выполнения и обеспечение правильного взаимодействия. Грубая синхронизация достигается путем программного планирования работы процессов, но не всегда надежна из-за случайных факторов и изменений в окружении. Тонкая синхронизация требует более надежного и безопасного логического разрешения, не зависящего от времени выполнения процессов. Задачи синхронизации включают в себя взаимное исключение, работу с общими ресурсами и другие типовые ситуации в программировании. Взаимное исключение процессов - это методика, позволяющая координировать работу параллельных процессов, когда они обращаются к общим ресурсам, которые могут использоваться только по очереди. Для этого используются мьютексы, специальные объекты, которые гарантируют, что в любой момент времени только один процесс имеет доступ к критическому ресурсу.

Процессы или потоки могут запросить доступ к критическому ресурсу с помощью операции Wait. Если ресурс доступен (мьютекс равен 1), процесс его захватывает, изменяя мьютекс на 0, и продолжает выполнение. Если ресурс занят (мьютекс равен 0), процесс блокируется, ожидая освобождения ресурса. После использования критического ресурса процесс освобождает его с помощью операции Signal, что возвращает мьютекс в состояние 1 и разрешает другим процессам использовать ресурс.

Однако неправильное использование мьютексов может привести к проблемам, таким как взаимоблокировки. Поэтому важно, чтобы программист правильно управлял операциями Wait и Signal для обеспечения согласованности и избежания тупиковых ситуаций.

Задача "читатели-писатели": несколько процессов нуждаются в доступе к общему ресурсу, например, базе данных. Читатели считывают данные, писатели пишут. Проблема: возможность конфликта, когда один процесс читает, а другой пытается писать. Решение: использование мьютексов для синхронизации доступа. Ошибка может проявиться, если прерывание произойдет в момент чтения данных.

Частые ошибки синхронизации включают гонки данных, ожидание потоков слишком долго, потерю сигналов и заброшенные замки. Гонки данных возникают при конфликте потоков за доступ к общим данным. Ошибки ожидания связаны с превышением допустимого времени ожидания потоком. Потеря сигналов происходит, когда поток ожидает событие, которое уже произошло. Заброшенные замки возникают, когда поток не освобождает захваченный ресурс.

29) Конструирование ПО и внутренние характеристики качества ПО Характеристики качества разработчика

1. Минимизация сложности разработки:

- Понятность кода: Понятный и легко читаемый код.
- Структуризация ПО: Разделение на небольшие части (модули).

2. Безопасный порядок доступа задач:
   - Безопасность многозадачности: Правильное управление доступом к критическим ресурсам при параллельной работе.
3. Готовность к изменениям:
   - Гибкость архитектуры: Подготовленность к изменениям благодаря модульности и использованию стандартизированных абстракций.
4. Устойчивость к ошибкам:
   - Толерантность к ошибкам: Способность ПО корректно обрабатывать или избегать критических ошибок.
5. Удобство тестирования:
1. Тестируемость: Легкость проведения тестов для проверки кода.

Стандартные приемы в конструировании:

2. Рефакторинг: Улучшение структуры кода для уменьшения сложности и повышения его качества.
3. Инкапсуляция: Сокрытие внутренней реализации и предоставление четко определенных интерфейсов.

Технологические программные заглушки:

4. Заглушки (stubs): Программные имитаторы, временно заменяющие ещё не разработанные модули. Используются для раннего тестирования и отладки.

30. Повышение приспособленности ПО к изменениям. Сокрытие информации, как метод избавления от повторной и нерациональной работы при изменениях. Рефáкторинг.

Повышение приспособленности ПО к изменениям включает использование метода сокрытия информации. Это означает изоляцию нестабильных элементов программы, таких как бизнес-правила или зависимости от оборудования, в отдельные компоненты или классы. Затем проектируются интерфейсы таким образом, чтобы изменения в одном компоненте не влияли на другие.

Сокрытие информации упрощает внесение изменений, так как позволяет избежать избыточного распространения информации по системе. Например, можно использовать константы или классы для управления параметрами, что облегчит изменения в случае необходимости. Также важно сосредоточить взаимодействие с пользователями в отдельных компонентах, чтобы изменения в этой области не затрагивали всю систему.

Рефакторинг - улучшение структуры кода при сохранении его функциональности. XP предлагает заниматься этим на протяжении всего проекта.

31) Ошибки ПО, отладка и тестирование

- Ошибки ПО неизбежны, так как человек ошибается.
- Отладка: поиск, локализация и устранение ошибок.
- Тестирование: выявление ошибок, часть отладки.

Необходимость анализа ошибок

Анализ помогает понять, почему ошибка произошла, её причины, и как её предотвратить в будущем.

Классификация ошибок ПО

- Программные ошибки: результат невнимательности, легко обнаруживаются.
- **Алгоритмические ошибки: возникают из-за неправильных представлений, сложны для выявления.**

32. Статическая отладка и динамическая отладка. Цель отладки. Принцип «белого» и «черного» ящика при динамической отладке ПО. Функциональная отладка.

Статическая отладка: анализ структуры программы и поиск шаблонных дефектов без ее фактического выполнения. Динамическая отладка: выполнение программы с тестовыми данными на реальном или эмулированном оборудовании. Цель - обнаружить ошибки в программе. Принцип "черного ящика" подразумевает проверку программы путем варьирования входных данных без знания ее внутреннего устройства. "Белый ящик", наоборот, предполагает анализ логических маршрутов программы с учетом ее структуры. Этот подход используется для более целенаправленного исследования программы на этапе разработки. Функциональная отладка проверяет, выполняет ли программа все свои функции. Но она может пропустить нефункциональные действия, не описанные в спецификациях. Для более надежной отладки часто комбинируется с другими методами.

33) **Структурная динамическая отладка идея:**

Из-за огромного числа данных полная отладка по принципу "черного ящика" невозможна. Использование структуры ПО помогает сократить объем работы.

Критерии отбора для отладки

- Графическое представление:
  Каждый тест — это путь на графе управления ПО.
- Покрытие всех путей:
  Проверка всех путей на графе гарантирует полноценную отладку, но это трудоемко.
- Покрытие связей:
  Проверка всех межпрограммных связей менее трудоемка, но менее надежна.
- Покрытие узлов:
  Проверка узлов графа без учета всех путей — наименее надежна.

Принципы тестирования

- Высокая вероятность обнаружения ошибки.
- Знать, когда остановиться.

- Трудно тестировать свою программу.
- Легкость воспроизведения тестов.
- Тесты для правильных и неправильных данных.
- Высокая квалификация тестировщиков.
- Не изменять программу для тестов.

34. Автономная отладка (АО) и комплексная отладка (КО) ПО. Имитационное математическое моделирование внешней среды для комплексной отладки ПО.
Автономная отладка проверяет отдельные части программы (юнит-тестирование), в то время как комплексная отладка (интеграционное тестирование) проверяет взаимодействие между этими частями. Автономная отладка проще и быстрее, но может пропустить сложные ошибки взаимодействия. Все функции программы должны быть проверены при отладке, но общая проверка не всегда гарантирует правильность внутренних взаимосвязей.
Имитационное математическое моделирование внешней среды для комплексной отладки ПО включает создание компьютерных моделей датчиков, объектов управления и других систем. В процессе моделирования эти модели генерируют данные и команды, которые поступают на входы программ по цепочке, как в реальной системе. Такая имитация помогает проверить взаимодействие между программами и убедиться, что все возможные варианты работы системы рассмотрены.

35) Модель Джелинского-Моранды : Математическая модель прогнозирования количества обнаруживаемых ошибок в программном обеспечении с течением времени
Основное уравнение:
$(dn)/(dt) = K \cdot (N_0 - n)$ Где:
- N_0 — начальное количество ошибок.
- n — количество обнаруженных ошибок.
- K — коэффициент пропорциональности.
- $(dn)/(dt)$ — скорость обнаружения ошибок.

Метод наименьших квадратов:
Используется для оценки параметров N_0 и K путём минимизации разностей между экспериментальными данными и модельными расчетами.
Процесс:
- Начальная оценка: задаём K.
- Итерации: корректируем K, пока разности не минимизируются.
- Определение N_0: решаем уравнение для N_0.

36. Модель(гипотеза) Джелинского – Моранды изменения по времени количества проявившихся ошибок в ПО. Оценка момента завершения отладки по наблюдению за процессом проявления ошибок.
Модель Джелинского – Моранды предсказывает, как изменяется количество ошибок в ПО со временем. Она говорит, что скорость появления ошибок пропорциональна оставшимся ошибкам. Это позволяет оценить надежность ПО и прогнозировать, когда ошибка станет редкой. Для определения параметров модели используют данные о количестве ошибок на

разных временных интервалах. Модель Джелинского-Моранды помогает определить момент завершения отладки ПО. Она предполагает прекращение отладки, когда количество оставшихся ошибок становится очень низким и интенсивность их проявления снижается. Однако модель имеет недостатки, так как непрерывна и не дает точного нуля ошибок. Тем не менее, она позволяет оценить, когда вероятность отсутствия ошибок становится высокой, что является сигналом для завершения отладки.

37) Системы Контроля Версий (СКВ)

Централизованные Системы (ЦСКВ)
- Пример: SVN, CVS

Особенности:
- Один центральный сервер.
- Все изменения хранятся на центральном сервере.
- Если сервер недоступен, работа останавливается.

Распределённые Системы (РСКВ)
- Пример: Git, Mercurial

Особенности:
- У каждого разработчика своя копия репозитория.
- Можно работать автономно без подключения к серверу.
- Повышенная надёжность и гибкость.

Итог
- ЦСКВ: Подходит для небольших команд; проще контроль доступа.
- РСКВ: Скалируемость, надёжность; лучше для больших команд.

38. Процедуры регистрации кода ПО. Технология внесения изменений в ПО. Подлинники и копии документации на ПО. Отчуждение подлинника от разработчика. Архив подлинников.

Процедуры регистрации кода ПО включают формализованный процесс учета изменений в программном обеспечении (ПО). Технология внесения изменений в ПО - это методы и инструменты для безопасного внедрения обновлений. Подлинники и копии документации на ПО представляют официальные и дублированные версии документации. Отчуждение подлинника от разработчика - передача прав на программное обеспечение. Архив подлинников - хранилище официальных версий исходного кода и документации ПО.

39) Устойчивые к ошибкам программы

Устойчивость к ошибкам - это способность ПО продолжать работать при наличии ошибок, минимизируя их последствия.

Виды контроля работы ПО
- Мониторинг производительности
- Логирование и анализ логов
- Тестирование (регрессия, нагрузочные тесты)
- Аварийное завершение и восстановление

Контроль работы ПО без прекращения его функционирования
- Системы самоконтроля и самовосстановления
- Аварийные планы и резервирование
- Избыточность систем для обеспечения бесперебойности

Технология контроля работы ПО
- Встраиваемые системы мониторинга
- Системы аварийного восстановления
- Использование эталонных значений и поведения для проверки текущего состояния

Эталоны для контроля работы ПО
Эталоны – это базовые значения и сценарии, с которыми сравнивается работа ПО для проверки его корректности и производительности.

40. Устойчивость ПО к ошибкам. Низкоуровневая и высокоуровневая защита от ошибок в программах.

Устойчивость ПО к ошибкам означает его способность работать надежно при возникновении проблем. Низкоуровневая защита включает встроенные механизмы языков программирования и операционных систем. Высокоуровневая защита достигается через использование методологий и паттернов программирования, которые делают код более структурированным и тестируемым.

41) Внешние низкоуровневые средства:
- Отладчики - шаг за шагом выполняют код, помогают найти ошибки.
- Мониторы системы - следят за состоянием системы в реальном времени.
- Инструменты профилирования - анализируют производительность программ.

Встроенные в ПО низкоуровневые средства:
- Ассерты - проверяют условия в коде, генерируют ошибки при их нарушении.
- Логи - записывают ключевые события и состояния программы.
- Обработчики исключений - ловят и обрабатывают ошибки, предотвращая крахи программ.

42. Три основных шага для обнаружения ошибок во входных данных. Встроенные в ПО низкоуровневые методы обработки обнаруженных ошибок во входных и выходных данных

Три основных шага для обнаружения ошибок во входных данных:
1. **Проверка формата:** Проверка, соответствуют ли входные данные ожидаемому формату. Например, это может быть проверка наличия необходимых полей или правильности типов данных.
2. **Валидация по бизнес-правилам:** Проверка входных данных на соответствие бизнес-правилам и логике программы. Это может включать в себя проверку диапазона значений, правильности формата даты и другие бизнес-правила.
3. **Тестирование граничных значений:** Проверка поведения программы при экстремальных значениях входных данных, таких как минимальные и максимальные значения. Это помогает выявить возможные проблемы с обработкой граничных условий.

Встроенные в ПО низкоуровневые методы обработки обнаруженных ошибок во входных и выходных данных могут включать:
1. **Генерацию исключений:** При обнаружении ошибок программа может генерировать исключения, которые позволяют обработать проблему в другом месте кода или уведомить пользователя о проблеме.
2. **Логирование ошибок:** Система может записывать обнаруженные ошибки в журнал (лог), что позволяет разработчикам отслеживать и анализировать возникающие проблемы.
3. **Возвращение кодов ошибок:** Функции или методы могут возвращать специальные коды ошибок, которые указывают на тип ошибки и позволяют вызывающему коду принять соответствующие меры.

43) Безопасность — минимизация ущерба от отказа ПО
Меры для снижения последствий от сбоев, например, резервные копии и автоматическое восстановление.
Аварийная защита в политике безопасности СТС
Механизмы, которые активируются при критических ошибках для предотвращения серьёзных проблем, например, аварийное выключение системы.
Отмена ошибочных команд
Функции, позволяющие откатывать действия, если была совершена ошибка, например, отмена транзакций или изменений в документах