**1. Понятия требований к разработке ПО, классификация, уровни требований**

Требования к разработке ПО — это спецификации, описывающие, что должно быть реализовано в программе. Они классифицируются на:

- Функциональные требования: описывают функции и поведение системы (например, вход пользователя, обработка данных).

- Нефункциональные требования: касаются производительности, надежности, безопасности и удобства использования.

Уровни требований включают: бизнес-требования (что нужно компании), системные требования (что требуется от системы), и требования к компонентам (что нужно отдельным модулям).

**2. Методологии и стандарты, регламентирующие работу с требованиями**

Существуют различные методологии для управления требованиями, такие как:

- Agile: гибкая методология, ориентированная на итеративную разработку и быструю адаптацию к изменениям.

- Waterfall: линейная методология, где каждый этап проекта происходит последовательно.

- Unified Process: включает модели разработки и управления, основанные на UML.

Стандарты, такие как IEEE 830 и ISO/IEC 25010, регламентируют оформление и качество спецификаций требований.

**3. Методы организации работы в команде разработчиков**

Методы организации работы включают:

- SCRUM: гибкий подход с короткими разбиениями на этапы (спринты) и регулярными встречами (stand-ups).

- Kanban: визуальный способ управления задачами, позволяющий контролировать поток работы и избежать перегрузок.

- Pair Programming: совместная работа двух программистов над одним кодом, что улучшает качество и обмен знаниями.

**4. Тестовое покрытие. Тестовый сценарий, тестовый пакет**

Тестовое покрытие — это мера, показывающая, какой процент кода или функционала покрыт тестами. Тестовый сценарий — это пошаговая инструкция для проверки конкретной функции программы. Тестовый пакет — это объединение нескольких тестовых сценариев, которые проверяют комплексную функциональность.

**5. Унифицированный язык моделирования UML. Нотации**

UML (Unified Modeling Language) — это язык моделирования, используемый для визуализации, спецификации и документации. Нотации UML включают:

-Диаграмма классов (Class Diagram): описывает классы, их атрибуты и методы, а также отношения между ними.

-Диаграмма компонентов (Component Diagram): отображает компоненты системы и их взаимодействие, полезна для представления сложных систем.

­­-Диаграмма развертывания (Deployment Diagram): показывает физическое   
развертывание программного обеспечения на аппаратных компонентах.

-Диаграмма пакетов (Package Diagram): группирует классы в пакеты и показывает зависимости между ними.

-Диаграмма объектов (Object Diagram): отображает экземпляры классов и их состояния в определенный момент времени.

-Диаграмма композитной структуры (Composite Structure Diagram): описывает внутреннюю структуру класса и взаимодействие его частей.

-Диаграмма профилей (Profile Diagram): позволяет создавать стереотипы для конкретных доменов и описывать их взаимосвязи.

**6. Диаграммы UML – общая характеристика**

Диаграммы UML подразделяются на два типа: структурные(например, диаграммы классов, объекта, компонентов) и поведенческие (например, диаграммы последовательностей, вариантов использования, состояний). Они помогают визуализировать архитектуру системы и взаимодействие между ее элементами.

**7. Предметы UML. Отношения UML**

Предметы UML включают:

- Классы: описывают объекты и их свойства.

- Объекты: экземпляры классов.

- Компоненты: модули системы.

Отношения UML: ассоциация (связь между классами), агрегация (композитная связь), композиция (сильная агрегация), относительно наследования (наследование характеристик).

**8. Создание в UML моделей анализа требований**

Создание моделей в UML для анализа требует создания диаграмм вариантов использования, чтобы показать, как пользователи взаимодействуют с системой. Также используются классовые диаграммы для визуализации структуры объектов и их взаимосвязей, что помогает выявить требования.

**9. Компонентные диаграммы UML**

Компонентные диаграммы UML отображают физическую структуру системы, показывая, как программные компоненты взаимодействуют друг с другом. Они помогают в управлении зависимостями и координации между разработкой разных частей системы.

**10. Описание и оформление требований (спецификация). Анализ требований и стратегии выбора решения**

Описание требований включает формальные документы с детальным изложением функциональных и нефункциональных требований. Анализ требований помогает выявить ключевые проблемы и требования пользователей, а стратегии выбора решения могут включать приоритетизацию, выбор реалистичных технологий и оценку затрат.

**11. Оценки программного проекта. Определение метрики**

Оценки программного проекта проводятся для получения представления о его масштабах и сложности. Оценки могут основываться на временных, стоимостных и функциональных метриках. Определение метрики может включать использование стандартных моделей оценки, таких как COCOMO

Метрика — это количественная мера, позволяющая оценить определенное свойство системы, компонента или процесса. Метрики помогают измерять качество программного обеспечения и могут быть основаны на различных аспектах, таких как производительность, стоимость и эффективность.

Метрики можно классифицировать следующим образом:

1)Внешние метрики: оценка свойств, видимых пользователю (например, надежность и функциональность).

2)Внутренние метрики: оценка свойств, важных для разработчиков (например, количество строк кода и сложность).

Применение метрик позволяет:

* Сравнивать проекты по качеству.
* Прогнозировать качество разрабатываемого ПО.
* Определять затраты на достижение требуемого уровня качества.

12. Размерно-ориентированные метрики ПС

Размерно-ориентированные метрики помогают оценить объем и сложность программного кода. Они предоставляют количественные данные о проекте, позволяя разработчикам и менеджерам лучше понимать его характеристики.

* **LOC (количество строк кода)**: это просто количество строк кода в проекте. Эта метрика помогает понять, сколько работы было выполнено, и позволяет сравнивать разные проекты по объему написанного кода. Однако стоит помнить, что количество строк не всегда указывает на сложность проекта — разные языки программирования могут требовать разное количество строк для реализации одной и той же функции.
* **FP (функциональные точки)**: этот показатель оценивает размер программного обеспечения на основе его функциональности, то есть того, что система может делать для пользователей. FP учитывает такие вещи, как количество входных данных, выходных данных и интерфейсов. Это позволяет более точно оценить сложность системы с точки зрения ее возможностей, а не только объема кода.

Метрики, ориентированные на размер, просты в использовании и широко применяются, но их недостаток в том, что они могут зависеть от языка программирования и требуют определённых данных, которые не всегда легко собрать на начальных этапах проекта.

13. Функционально-ориентированные метрики ПС

Функционально-ориентированные метрики сосредоточены на функциях, которые предоставляет система. Они помогают оценить качество работы системы и то, насколько хорошо она удовлетворяет потребности пользователей.Например:

* **Количество внешних вводов**: это число всех данных, которые пользователь вводит в систему.
* **Количество внешних выводов**: Это число результатов, которые система возвращает пользователю.

Эти показатели полезны для оценки производительности системы и понимания того, насколько хорошо она выполняет свои функции. Они помогают разработчикам увидеть, какие функции работают хорошо, а где есть возможности для улучшения.

14. Объектно-ориентированные метрики ПС

Объектно-ориентированные метрики анализируют структуру системы и ее проектирование:

* **Количество классов и объектов**: это помогает понять, как организован код и насколько он сложен.
* **Глубина наследования**: эта метрика показывает, насколько глубоко классы вложены друг в друга. Чем глубже наследование, тем сложнее может быть понимание кода.
* **Количество связей между классами**: эта метрика показывает, как классы взаимодействуют друг с другом. Если классы сильно связаны, это может затруднить их изменение.

Эти показатели помогают разработчикам лучше понять сложность дизайна системы и выявить потенциальные проблемы на этапе проектирования. Они также способствуют повышению качества программного обеспечения за счёт поиска областей для оптимизации и рефакторинга кода.

**15. Понятие репозитория проекта, структура проекта**

Репозиторий проекта — это централизованное хранилище всех материалов проекта, включая код, документацию и тесты. Структура может включать каталоги для исходного кода, документации, скриптов, тестов и ресурсов, что обеспечивает организованность и легкость в доступе к необходимым материалам.

**16. Виды, цели и уровни интеграции программных модулей**

Интеграция программных модулей может быть:

- Инкрементальной: постепенное добавление модулей в систему.

- Системной: интеграция всех компонентов сразу.

Цели включают минимизацию конфликтов, упрощение совместимости и подготовку к тестированию.

**17. Организация работы команды в системе контроля версий**

Организация работы включает использование инструментов, таких как Git, для управления версиями кода. Практики включают создание веток для функций и исправлений, код-ревью перед слиянием, и регулярные слияния для предотвращения конфликтов.

**18. Отладка программных продуктов**

Отладка — это процесс нахождения и устранения ошибок в программном обеспечении. Это включает в себя воспроизведение ошибок, пошаговое выполнение кода и анализ переменных, что позволяет точно определить причину сбоя и исправить его.

**19. Инструменты отладки. Отладочные классы**

Инструменты отладки, такие как IDE (например, Visual Studio, Eclipse), поддерживают установку точек остановки, наблюдение за переменными и тестирование кода. Отладочные классы включают специальные функции для логирования и диагностики, которые помогают выявить проблемы в коде.

**20. Инструментарии анализа качества программных продуктов в среде разработке**

Анализ качества программного обеспечения включает использование инструментов статического и динамического анализа, таких как SonarQube и PMD. Эти инструменты помогают выявить потенциальные проблемы, невыполненные требования и улучшить качество кода.

**21. Обработка исключительных ситуаций**

Обработка исключительных ситуаций включает в себя использование конструкций try-catch для управления ошибками. При возникновении ошибки система должна быть в состоянии корректно обработать исключение, возможно, сохранив состояние или предоставив пользователю понятное сообщение об ошибке.

**22. Методы и способы идентификации сбоев и ошибок**

Идентификация сбоев и ошибок в программном обеспечении является ключевым аспектом обеспечения его качества. Существует несколько методов и способов, которые помогают выявлять проблемы на различных этапах разработки и эксплуатации.

Методы идентификации сбоев и ошибок

1. **Ведение журнала**: запись информации о событиях в системе позволяет отслеживать поведение приложения и выявлять аномалии. Журналы содержат данные о выполнении операций, ошибках и предупреждениях.
2. **Мониторинг в реальном времени**: использование инструментов для отслеживания состояния системы позволяет оперативно реагировать на сбои, включая мониторинг производительности и доступности сервисов.
3. **Регулярное тестирование**:
   * **Функциональное тестирование**: Проверка соответствия программного обеспечения функциональным требованиям.
   * **Тестирование производительности**: Оценка работы системы под нагрузкой.
   * **Автоматизированное тестирование**: использование скриптов для быстрого выполнения тестов после внесения изменений в код.
4. **Статический анализ кода**: анализ исходного кода без его выполнения помогает выявить потенциальные ошибки и уязвимости на ранних этапах разработки.
5. **Методы проверки модели**: формальные методы проверки соответствия модели программного обеспечения заданным требованиям, такие как модельная проверка.

Эти методы способствуют не только выявлению ошибок, но и повышению качества программного обеспечения, обеспечивая его надёжность и стабильность в работе.

**23. Инструментально-объектный подход к разработке программного средства**

Инструментально-объектный подход предполагает использование инструментов и библиотек для создания объектно-ориентированных приложений. Такой подход стандартизирует проектирование и разработку, позволяя разработчикам сосредоточиться на бизнес-логике, а не на низкоуровневых деталях реализации.

Вот информативные ответы на каждый из ваших экзаменационных билетов:

**24. Основные классы инструментальных сред разработки и сопровождения программных средств. Инструментальные среды программирования.**

Инструментальные среды разработки (IDE) включают классы, такие как:

- Редакторы кода: программы, поддерживающие написание кода (например, Visual Studio Code, Sublime Text).

- Инструменты отладки: помогают выявлять и устранять ошибки (например, GDB, WinDbg).

- Системы управления версиями: отслеживают изменения в коде (например, Git, SVN).

- Инструменты для автоматизации сборки: упрощают процесс компиляции и сборки проекта (например, Maven, Gradle).

- CASE-средства: поддерживают полный жизненный цикл разработки от анализа до тестирования.

**25. Базовые стратегии разработки ПС, достоинства и недостатки.**

1. Каскадная стратегия (Waterfall)

Суть

Каскадная модель — это линейный подход к разработке ПО, при котором каждый этап (анализ требований, проектирование, кодирование, тестирование и сопровождение) завершается перед началом следующего.

Достоинства

* **Четкая структура**: Каждый этап имеет ясные цели.
* **Легкость управления**: Простой процесс управления проектом.
* **Документация**: Хорошая документация на каждом этапе.

Недостатки

* **Низкая гибкость**: Трудно адаптироваться к изменениям требований.
* **Высокий риск**: Ошибки могут быть выявлены слишком поздно.
* **Неэффективно для сложных проектов**: не подходит для проектов с частыми изменениями.

2. Итеративная разработка

Суть

Итеративная модель включает повторяющиеся циклы разработки, где каждая итерация включает планирование, проектирование, реализацию и тестирование.

Достоинства

* **Обратная связь**: Возможность получения отзывов от пользователей.
* **Раннее выявление ошибок**: Ошибки обнаруживаются быстрее.
* **Гибкость**: Легче адаптироваться к изменениям требований.

Недостатки

* **Сложность управления**: Усложняет управление проектом из-за множества итераций.
* **Потребность в ресурсах**: требует больше времени и ресурсов для тестирования.

3. Гибкий подход

Суть

Agile — это гибкая методология, которая делает акцент на быстром реагировании на изменения и взаимодействии с заказчиком посредством коротких итераций (спринтов).

Достоинства

* **Гибкость**: Легко адаптируется к изменениям.
* **Скорость разработки**: Быстрое создание работающих версий продукта.
* **Командное взаимодействие**: Стимулирует сотрудничество в команде.

Недостатки

* **Неопределенность в планировании**: сложно предсказать сроки и стоимость проекта.
* **Зависимость от команды**: Эффективность зависит от уровня самоорганизации команды.

4. Прототипирование

Суть

Прототипирование включает в себя создание предварительных версий продукта для демонстрации заказчику, что помогает лучше понять требования.

Достоинства

* **Визуализация требований**: Позволяет пользователям видеть функциональность заранее.
* **Улучшение коммуникации**: способствует лучшему взаимопониманию между разработчиками и заказчиками.
* **Снижение рисков**: Выявление проблем на ранних стадиях.

Недостатки

* **Время на доработку**: может потребовать значительных усилий для доработки прототипа.
* **Ограниченная функциональность**: прототипы могут не отражать все аспекты конечного продукта.

Каждая из этих стратегий имеет свои сильные и слабые стороны, и выбор подходящей модели зависит от конкретных условий проекта.

**26. Основной компонент инструментальных систем технологии программирования репозитория.**

Основной компонент репозитория — это система управления версиями (например, Git), которая обеспечивает хранение кода, отслеживание изменений, совместное использование, а также управление конфликты между командами и версиями проекта. Репозиторий служит единой точкой для доступа к актуальному состоянию проекта и его артефактам.

**27. Понятие CASE – средства, их назначение и применение.**

CASE (Computer-Aided Software Engineering) средства - это программные инструменты, поддерживающие процессы разработки ПО. Их функции включают:

- Поддержка моделирования и проектирования системы.

- Генерация кода и документации.

- Управление проектами и тестирование.

Современные CASE-средства обеспечивают интеграцию методов разработки, доступ к шаблонам, а также средства для визуализации процессов.

**28. Математические модели, принципы их построения, виды моделей.**

Математические модели — это абстрактные представления, которые используют математические методы для описания систем или процессов. Принципы их построения включают определение переменных, формулирование уравнений и ограничений, а также проверку модели.

Виды математических моделей

Математические модели можно классифицировать по различным критериям, включая статические и динамические, а также детерминированные и стохастические.

1. Статические и динамические модели

* **Статические модели**: эти модели описывают системы в определенный момент времени и не учитывают изменения во времени. Они полезны для анализа систем с постоянными параметрами.
* **Динамические модели**: эти модели учитывают изменения системы во времени. Они используются для описания процессов, которые развиваются и изменяются, таких как экономические или биологические системы.

2. Детерминированные и стохастические модели

* **Детерминированные модели**: в этих моделях все параметры и переменные известны и фиксированы. Результаты таких моделей предсказуемы и могут быть точно рассчитаны при заданных условиях. Например, уравнения движения в физике часто являются детерминированными.
* **Стохастические модели**: эти модели учитывают случайные факторы и неопределённость. Параметры могут изменяться в зависимости от вероятностных распределений, что делает результаты менее предсказуемыми. Стохастические модели часто используются в финансовом и экономическом анализе, где влияние случайных событий может быть значительным.

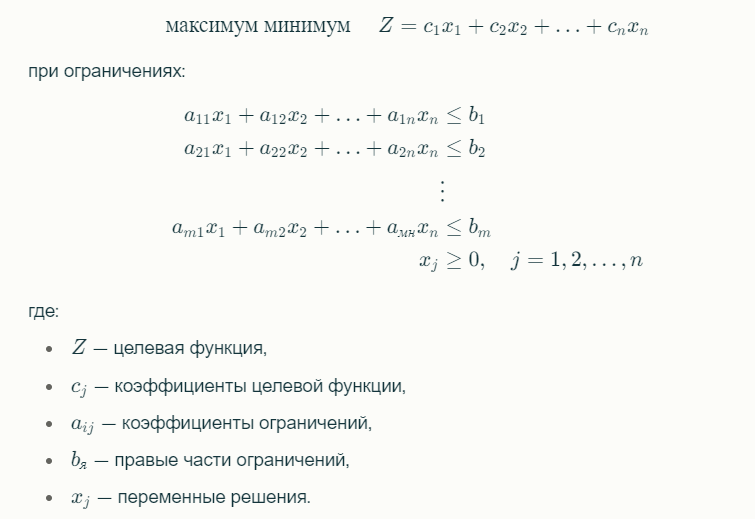
Принципы построения математических моделей

1. **Определение переменных**: Необходимо четко определить ключевые переменные, которые будут использоваться в модели.
2. **Формулирование уравнений**: на основе выбранных переменных создаются уравнения, описывающие взаимосвязи между ними.
3. **Верификация модели**: проверка модели на соответствие реальным данным и требованиям.

**29. Понятие решения. Множество решений, оптимальное решение. Показатель эффективности решения.**

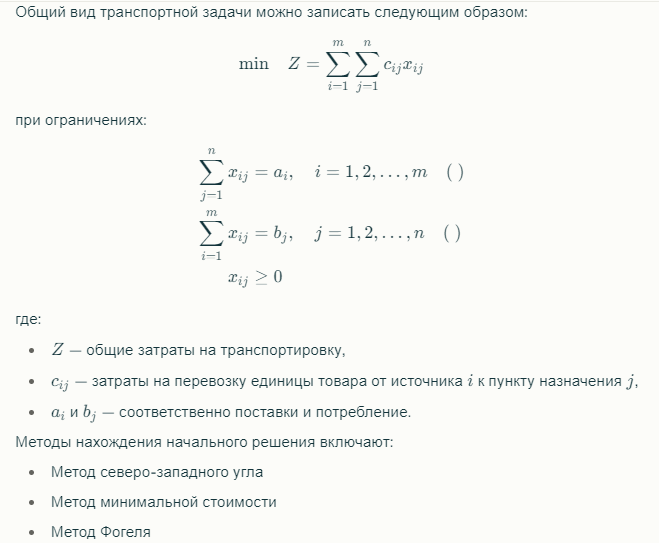
Решение — это конкретный ответ или способ достижения цели задачи. Множество решений — это все возможные альтернативы. Оптимальное решение — наилучший из возможных вариантов, минимизирующий или максимизирующий некоторую цель. Показатель эффективности может рассматриваться как измерение качества решения, например, цена, время, прибыль и т. д.

**30. Общий вид и основная задача линейного программирования. Симплекс – метод.**

**Общий вид задачи линейного программирования** можно записать в следующем виде:

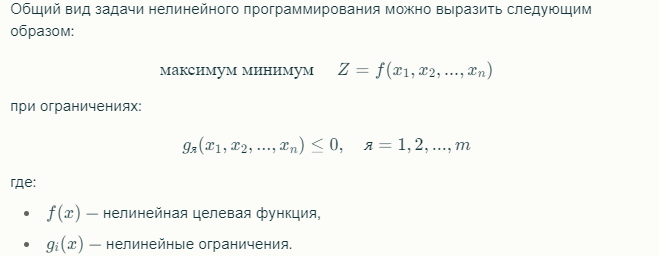
**Симплекс-метод** — это алгоритм для нахождения оптимального решения задачи линейного программирования, который перемещается по вершинам многогранника (области допустимых решений) для поиска максимума или минимума целевой функции.

**31. Транспортная задача. Методы нахождения начального решения транспортной задачи. Метод потенциалов.**



Метод потенциалов используется для улучшения начального решения путем анализа потенциальной выгоды от перемещения грузов между пунктами.

**32. Общий вид задач нелинейного программирования. Графический метод решения задач нелинейного программирования.**



**Графический метод решения** задач нелинейного программирования используется для визуализации и поиска оптимального решения в двумерном пространстве. Этот метод позволяет отобразить область допустимых решений и целевую функцию, находя точку максимума или минимума на графике.  
Для решения графическим методом:

* Определяются области допустимых решений.
* Наносится целевая функция.
* Точка касания линии уровня с границей области допустимых решений ищется для нахождения оптимума.

**33. Основные понятия динамического программирования**

Динамическое программирование — метод оптимизации, основанный на разбиении задачи на подзадачи, которые решаются последовательно. Основные понятия включают:

- Оптимальность: находить глобально оптимальное решение через локальные.

- Мемоизация: сохранение результатов подзадач для избежания повторного вычисления.

- Разделяй и властвуй: метод для структурирования задач.

**34. Задача о нахождении кратчайших путей в графе и методы ее решения.**

Задача о нахождении кратчайшего пути заключается в поиске минимального расстояния между двумя вершинами графа. Методы ее решения включают:

- Алгоритм Дейкстры: ищет кратчайший путь от одной вершины ко всем остальным.

- Алгоритм Флойда-Уоршелла: эффективен для нахождения кратчайших путей между всеми парами вершин.

**35. Задача о максимальном потоке и алгоритм Форда–Фалкерсона.**

Задача о максимальном потоке заключается в нахождении максимального потока в сети с заданными емкостями. Алгоритм Форда–Фалкерсона использует метод поиска пути для непрерывного увеличения потока от источника к стоку до тех пор, пока это возможно, путём поиска увеличивающих путей в графе.

**36. Математические, имитационные и эвристические модели исследования операций.**

Математические модели используют строгие математические выражения для описания системы. Имитационные модели представляют собой компьютерные симуляции, которые помогают анализировать поведение системы. Эвристические модели используют методы приближения для поиска решений, когда точный расчет затруднен. Понятия допускаемого и оптимального решения применяются в контексте решения задач, связанных с линейным программированием, определяя качественные границы решений.

**37. Системы массового обслуживания: понятия, примеры, модели.**

Системы массового обслуживания — это модели, описывающие поведение сервисных систем, где группа клиентов обслуживается одним или несколькими обслуживающими устройствами. Примеры включают телефонные центры, банкоматы и сервисные центры. Модели, такие как M/M/1 или M/M/c, описывают интенсивность потока клиентов и количество обслуживающих систем.

**38. Схема гибели и размножения.**

Схема гибели и размножения описывает динамику популяций, основанную на двух процессах: гибели населения и их размножении. Это позволяет исследовать, как различные факторы влияют на популяцию во времени. Модели могут быть представлены через дифференциальные уравнения, описывающие скорость изменения популяции.

**39. Понятие прогноза. Количественные методы прогнозирования. Качественные методы прогноза.**

Прогнозирование — процесс предсказания будущих событий на основе имеющихся данных. Количественные методы используют статистические и математические модели (например, временные ряды, регрессия), чтобы предсказать будущие значения. Качественные методы, включая экспертные оценки и опросы, применяются в ситуациях, когда недостаточно данных для количественных моделей.

**40. Основные понятия теории игр. Антагонистические матричные игры: чистые и смешанные стратегии.**

Теория игр изучает математические модели конфликтов и сотрудничества между разумными игроками. Антагонистические матричные игры включают стратегии, где игроки достигают противоречивых целей. Чистые стратегии — это выбор одной определенной стратегии, в то время как смешанные стратегии подразумевают случайный выбор между несколькими возможными стратегиями с определенными вероятностями.