Подготовка к зачёту по ОПИ

1. Понятие программной инженерии, ее цели и задачи. Стандарты программной инженерии.

ПИ является отраслью информатики и изучает вопросы построения компьютерных программ, обобщает опыт программирования в виде комплекса общих знаний и правил регламентации инженерной деятельности разработчиков программного обеспечения.

Стандарт — типовой образец, эталон, модель, принимаемые за исходные для сопоставления с ними других предметов.

- корпоративные *(фирма захотела улучшить производство, ввела стандарт)*
- отраслевые *(действует в пределах организаций некоторой отрасли)*
- государственные *(имеют силу закона и принимаются гос. органами)*
- международные *(разрабатываются специальными организациями)*

2. Основные этапы разработки программ, их назначение и характеристики.

Этапы:

1. Постановка задачи
2. Выбор метода решения
3. Разработка алгоритма
4. Написание программы
5. Ввод программы в компьютер
6. Трансляция
7. Компоновка
8. Выполнение
9. Тестирование
10. Отладка
11. Документирование
12. Эксплуатация
13. Модификация
14. Снятие с эксплуатации

3. Порядок прохождения задач через ЭВМ.

Ввод программы → исходный код → (1), (2) или (3)

(1) → компилятор → объектный код → компоновщик → исполняемый файл
(2) → байт-код → виртуальная машина
(3) → интерпретатор

4. Понятие жизненного цикла программного обеспечения.

ЖЦПО — это период с момента принятия решения о создании ПО до изъятия из эксплуатации.

Основные понятия:

- *Артефакты* — создаваемые человеком информационные сущности – документы, в достаточно общем смысле участвующие в качестве входных данных и получающиеся в качестве результатов различных деятельностей.
- *Роль* — абстрактная группа заинтересованных лиц, участвующих в деятельности по созданию и эксплуатации системы, решающих одни и те же задачи или имеющих одни и те же интересы по отношению к ней.

- *Программный продукт* — набор компьютерных программ, процедур и, возможно связанных с ними документации и данных.
- *Процесс* — совокупность взаимосвязанных действий, преобразующих некоторые входные данные в выходные.

5. Основные процессы жизненного цикла программного обеспечения

- Приобретение
- Поставка
- Разработка
    - (Выбор модели жизненного цикла, Анализ требований к системе, Проектирование архитектуры системы, Анализ программных требований, Детальное конструирование ПО, Кодирование и тестирование ПО, Интеграция ПО, Квалификационное тестирование ПО, Интеграция системы, Квалификационное тестирование системы, Установка ПО, Приемка ПО)
- Эксплуатация
    - (Эксплуатация системы выполняется в предназначенной для этого среде в соответствии с пользовательской документацией, Включает установление эксплуатационных стандартов и проведение эксплуатационного тестирования)
- Сопровождение
    - (внесение изменений в ПО в целях исправления ошибок, повышения производительности или адаптации к изменившимся условиям работы или требованиям)

6. Вспомогательные и организационные процессы жизненного цикла программного обеспечения.

- Вспомогательные
    - Документирование
    - Управление конфигурацией
    - Обеспечение качества
    - Верификация
    - Аттестация
    - Совместная оценка
    - Аудит
    - Разрешение проблем
- Организационные
    - Управление проектом
    - Создание инфраструктуры
    - Оценка и улучшение жизненного цикла
    - Обучение

7. Качество программного обеспечения и пути его достижения. Стандарты качества программного обеспечения.

- Определение ISO: *Качество* — это полнота свойств и характеристик продукта, процесса или услуги, которые обеспечивают способность удовлетворять заявленным или подразумеваемым потребностям.
- Определение IEEE: *Качество* — степень, в которой ПО обладает требуемой комбинацией свойств.

Качество ПО::

- внешнее качество, заданное требованиями заказчика в спецификациях и отражающееся характеристиками конечного продукта
- внутреннее качество, проявляющееся в процессе разработки и других промежуточных этапах жизненного цикла ПС
- качество при использовании в процессе нормальной эксплуатации и результативность достижения потребностей пользователей с учетом затрат ресурсов (эксплуатационное качество)

Стандарты качества:

- ISO/IEC 9126:1-4:2002 (ГОСТ Р ИСО/МЭК 9126-93)
- ISO/IEC 14598 – набор стандартов, регламентирующий способы оценки характеристик качества

В совокупности они образуют модель качества — SQuaRE (Software Quality Requirements and Evaluation)

Чтобы принять решение о том, достаточно ли качественным является разработанное ПО, надо сравнить стоимость непоставки программного продукта с эксплуатационной стоимостью в случае его поставки.

Система обеспечения качества программных средств — это совокупность методов и средств организации управляющих и исполнительных подразделений предприятия, участвующих в проектировании, разработке и сопровождении комплексов программ с целью придания им свойств, обеспечивающих удовлетворение потребностей заказчиков и потребителей при минимальном или допустимом расходовании ресурсов.

Основы системы обеспечения качества:

- Методы обеспечения качества представляют собой техники, гарантирующие достижение определенных показателей качества при их применении.
- Методы контроля качества позволяют убедиться, что определенные характеристики качества ПО достигнуты.

Обеспечение и удостоверение качества сложных программных средств должно базироваться на проверках и испытаниях:

- технологий обеспечения жизненного цикла ПО, поддержанного регламентированными системами качества
- готового программного продукта с полным комплектом адекватной эксплуатационной документации

8. Характеристики, используемые для оценки качества программного обеспечения

- *Функциональные возможности* — способность программного обеспечения реализовать установленные или предполагаемые потребности пользователей
- *Надежность* — способность программного обеспечения сохранять свой уровень качества функционирования при установленных условиях за установленный период времени

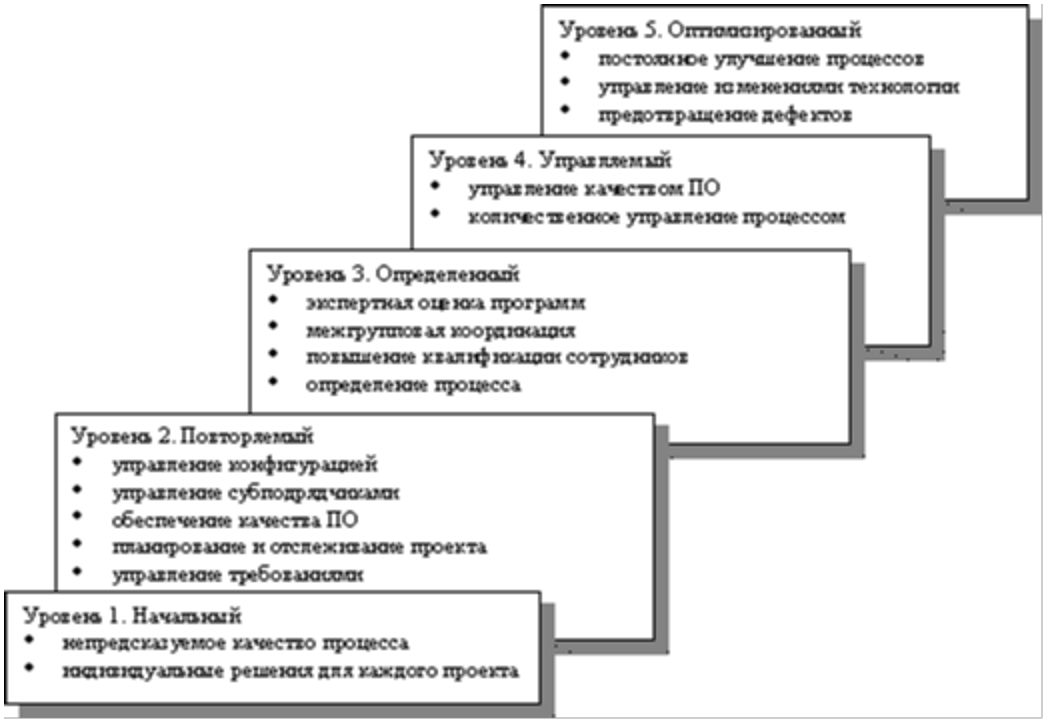
- *Практичность* — характеризуется объемом работ, требуемых для использования программного обеспечения определенным или предполагаемым кругом пользователей
- *Эффективность* — определяется соотношением между уровнем качества функционирования программного обеспечения и объемом используемых ресурсов при установленных условиях
- *Сопровождаемость* — характеризует объем работ, требуемых для проведения конкретных изменений (модификаций)
- *Мобильность* — способность программного обеспечения быть перенесенным из одного окружения в другое

9. Современные модели оценки качества программного обеспечения.

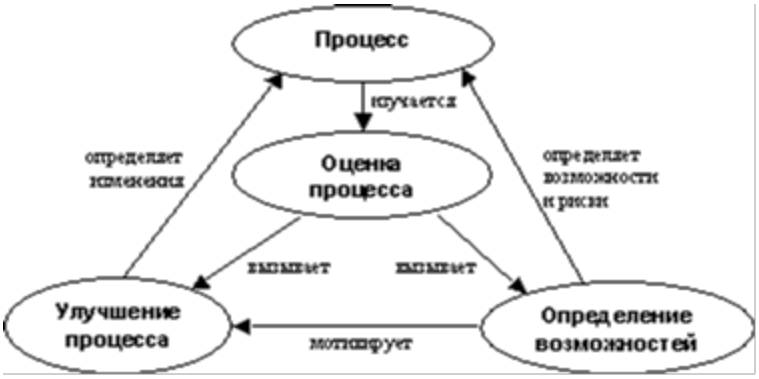
- ISO 9000:2000 Системы управления качеством – Основы и словарь.
- ISO 9001:2000 Системы управления качеством – Требования. Модели для обеспечения качества при проектировании, разработке, производстве, установке и обслуживании. Определяет общие правила обеспечения качества результатов во всех процессах жизненного цикла

Набор стандартов ISO 9000 регулирует общие принципы обеспечения качества процессов производства во всех отраслях экономики.

Пять уровней зрелости в модели:
CMM (Capability Maturity Model)



Основные элементы стандарта:
ISO/IEC 15504 (SPICE)



10. Системы управления версиями. Типичный цикл работы с проектом при использовании систем управления версиями.

Система управления версиями — ПО для облегчения работы с изменяющейся информацией.

Делятся на централизованные (напр., svn) и децентрализованные (напр., git).

Типичный цикл работы с проектом (для централизованной VCS):

1. Подключение к серверу под соответствующей учетной записью
2. Добавление в репозиторий нового проекта
3. Извлечение рабочей копии проекта
4. Модификация проекта
5. Фиксация изменений

11. Разработка алгоритмов. Свойства алгоритмов. Процесс алгоритмизации. Способы описания алгоритмов. Примеры.

Алгоритм — это полное и точное описание на некотором языке конечной последовательности правил, указывающих исполнителю действия, которые он должен выполнить, чтобы за конечное время перейти от исходных данных к искомому результату.

Свойства алгоритмов:

1. Дискретность
2. Понятность
3. Определенность
4. Конечность
5. Массовость
6. Эффективность

Процесс алгоритмизации:

1. декомпозиция на шаги
2. установление взаимосвязей между отдельными шагами
3. точное описание содержания каждого шага на языке выбранной алгоритмической системы
4. проверка составленного алгоритма на предмет, действительно ли он реализует выбранный метод и приводит к искомому результату

Способы описания алгоритмов:

1. словесно-формульный
2. структурный или блок-схемный
3. с использованием специальных алгоритмических языков
4. с помощью сетей Петри
5. программный

12. Что характеризует тип данных? Простые и структурированные типы данных.

Информация характеризуется типом данных, который определяет:

1. диапазон допустимых значений для данных
2. диапазон допустимых операций над данными
3. количество оперативной памяти, и выбор представления в ней.

На машинном уровне тип данного представляет собой выбор машинных команд, в выполнении которых участвует это данное.
Особенностью простого данного того или иного типа является именно простота организации (неструктурированность).

Логическая и математическая модель организации данных называется *структурой данных*.

Выбор структуры данных зависит от следующих параметров:

1. Структура данных должна отражать фактические связи данных в реальном мире.
2. Структура данных должна быть достаточно проста, для простоты ее обработки.

Таким образом, СД – это организация и тип данных.
Она характеризует:

1. характер организации данных
2. множество допустимых значений
3. набор допустимых операций

13. Общая классификация структур данных.

1. по наличию связей между элементами: *несвязные* (векторы, массивы, строки, стеки, очереди) и *связные* (связные списки)
2. по характеру упорядоченности элементов: *упорядоченные* и *неупорядоченные*
3. по распределению элементов в памяти: *линейные* (массивы, строки, стеки, деки, очереди) и *нелинейные* (списки)
4. по изменчивости: *статические*, *полустатические* и *динамические*.

В свою очередь, статические СД делятся на:

- простые
- составные (агрегативные)

а динамические СД делятся на:

- несвязные динамические структуры
- связные динамические структуры
- файлы

14. Тестирование программ. Цели и задачи тестирования. Методы тестирования программного обеспечения.

Тестирование — это процесс исполнения программы с целью обнаружения ошибок.
Цель тестирования — показать, что программа корректно исполняет предусмотренные функции

Тестирование методом «черного» ящика:

1. Это тестирование с управлением по данным (или с управлением по входу-выходу) без знания кода программы
2. Целью тестирования является выяснение обстоятельств, в которых поведение программы не соответствует ее спецификации

Функциональные тесты:

- *Тепличные* — проверяющие программу при корректных значениях исходных данных;
- *Экстремальные* — проверяющие поведение программной системы в экстремальных ситуациях, которые могут произойти и на которые она должна корректно реагировать ;
- *Запредельные* — проверяющие ситуации, бессмысленные с точки зрения постановки задачи, которые могут произойти из-за ошибок пользователя.

Тестирование методом «белого» ящика:

1. Это тестирование внутреннего поведения программы
2. Проверяется корректность построения всех элементов программы и правильность их взаимодействия

Структурные тесты:

- контроль обращений к данным
- контроль вычислений
- контроль передачи управления
- контроль межмодульных интерфейсов

Уровни тестирования:

- *Модульное тестирование (юнит-тестирование)* — тестируется минимально возможный для тестирования компонент, например, отдельный класс или функция.
- *Интеграционное тестирование* — тестируются интерфейсы между модулями.
- *Системное тестирование* — тестируется интегрированная система на её соответствие требованиям.

15. Тестирование «черным ящиком». Метод эквивалентного разбиения. Классы эквивалентности тестов и способы их выделения. Построение тестов.

Стратегии черного ящика: *метод эквивалентного разбиения*.

Правильно выбранный тест этого подмножества должен обладать двумя свойствами:

- уменьшать, причем более чем на единицу, число других тестов
- покрывать значительную часть других возможных тестов

Класс эквивалентности — это класс, в рамках которого все тесты являются эквивалентными, т.е. такими которые приводят к одному и тому же результату.

Два типа классов эквивалентности:

- правильные классы эквивалентности, представляющие правильные входные данные программы
- неправильные классы эквивалентности, представляющие все другие возможные состояния условий

Выделение классов эквивалентности:

- Если входное условие описывает множество входных значений и есть основание полагать, что каждое значение программа трактует особо
- Если входное условие описывает ситуацию «должно быть» (например, «первым символом идентификатора должна быть буква»), то определяется один правильный класс эквивалентности (первый символ – буква) и один неправильный (первый символ – не буква)
- Если есть любое основание считать, что различные элементы класса эквивалентности трактуются программой неодинаково, то данный класс эквивалентности разбивается на меньшие классы эквивалентности

Построение тестов:

- Назначение каждому классу эквивалентности уникального номера
- Проектирование новых тестов, каждый из которых покрывает как можно большее число непокрытых правильных классов эквивалентности, до тех пор пока все правильные классы эквивалентности не будут покрыты (только не общими) тестами
- Запись тестов, каждый из которых покрывает один и только один из непокрытых неправильных классов эквивалентности, до тех пор, пока все неправильные классы эквивалентности не будут покрыты тестами
- Выбор любого элемента в классе эквивалентности в качестве представительного при анализе граничных значений осуществляется таким образом, чтобы проверить тестом каждую границу этого класса
- При разработке тестов рассматривают не только входные условия (пространство входов), но и пространство результатов (т.е. выходные классы эквивалентности)

16. Тестирование «черным ящиком». Анализ граничных значений. Применение данного метода. Метод предположения об ошибке. Проектирование и исполнение теста.

Граничные условия — ситуации, возникающие непосредственно на, выше или ниже границ входных и выходных классов эквивалентности.

Анализ граничных значений.

Правила:

1. Выбор любого элемента в классе эквивалентности в качестве представительного при анализе граничных значений осуществляется таким образом, чтобы проверить тестом каждую границу этого класса.
2. При разработке тестов рассматривают не только пространство входов, но и пространство результатов.

Алгоритм:

1. Построить тесты для границ области и тесты с неправильными входными данными для ситуаций незначительного выхода за границы области.
2. Построить тесты для минимального и максимального значений условий и тесты, большие и меньшие этих значений.
3. Использовать правило 1 для каждого выходного условия.
4. Использовать правило 2 для каждого выходного условия.

Метод предположения об ошибке.

Предположение вероятных типов ошибок и разработка тестов для их обнаружения.

Проектирование и исполнение теста.

- определить цель теста
- написать входные значения
- написать предполагаемые выходные значения
- выполнить тест и зафиксировать результат
- проанализировать результат

17. Стратегии «белого» ящика: покрытие решений и комбинаторное покрытие условий. Стратегия разработки тестов.

Покрытие решений:

- каждое решение должно иметь результатом значения истина и ложь и при этом каждый оператор должен выполняться бы по крайней мере один раз;

Покрытие условий:

- число тестов должно быть достаточно для того, чтобы все возможные результаты каждого условия в решении выполнялись по крайней мере один раз;

Комбинаторное покрытие условий:

- создание такого числа тестов, чтобы все возможные комбинации результатов условия в каждом решении и все точки входа выполнялись по крайней мере один раз.

Стратегия разработки тестов.

1. Провести анализ граничных значений.
2. Определить правильные и неправильные классы эквивалентности для входных и выходных данных и дополнить, если это необходимо, тесты, построенные на предыдущем шаге.
3. Для получения дополнительных тестов рекомендуется использовать метод предположения об ошибке.
4. Проверить логику программы на полученном наборе тестов. Для этого нужно воспользоваться критерием покрытия решений, покрытия условий, либо комбинаторного покрытия условий, при необходимости подготовив недостающие тесты.

18. Отладка программ. Виды ошибок в программах и последовательность их обнаружения. Методы отладки.

Виды ошибок в программах:

- ошибки в описании задачи (программа правильно решает неверную задачу);
- ошибки в выборе алгоритма (использование неподходящего или неэффективного алгоритма);

- ошибки анализа (связаны с неполным учетом возможных ситуаций, либо с неправильным решением задачи);
- ошибки общего характера (не зависящие от языка программирования), aka тупой программист;
- ошибки физического характера (вызываемые неверными действиями программиста), aka человеческий фактор.

Обнаружение ошибок

Признаки ошибок:

- Отсутствует уверенность в том, что программа начала выполняться;
- программа начала выполняться, но произошел преждевременный останов с выдачей или без выдачи сообщения о системной ошибке;
- программа начала выполняться, но зациклилась, о чем можно судить по ее чрезмерно долгой работе;
- программа выдала неправильную информацию.

Алгоритм:

1. Выявить точку обнаружения — место в программе, где ошибка себя проявляет или становится очевидной.
2. Выявить точку происхождения — место в программе, где возникают условия для появления ошибки

Отладка — этап разработки программы, на котором обнаруживают, локализуют и устраняют ошибки.

Два взаимодополняющих метода отладки:

1. Использование отладчиков.
2. Вывод отладочной информации в критических точках (журналирование).

19. Структурное программирование. Два подхода к проектированию и разработке программ в рамках структурного программирования.

Структурное программирование — методология, в основе которой лежит представление программы в виде иерархии блоков.

Блоки:

- структуры следования
- структуры ветвления
- структуры повторения

Первоначально, Дейкстра ставил обязательным условием принцип *SESE* (single entry, single exit), однако сегодня, благодаря Макконнеллу, этот принцип не является обязательным.

Подходы, используемые в структурной (и не только) парадигме:

- нисходящая разработка (сверху-вниз, анализ)
- восходящая разработка (снизу-вверх, синтез)

20. Модели жизненного цикла программного обеспечения. Побудительные причины изучения моделей жизненного цикла. Подходы к моделированию жизненного цикла.

Каскадная модель ЖЦ ПО.

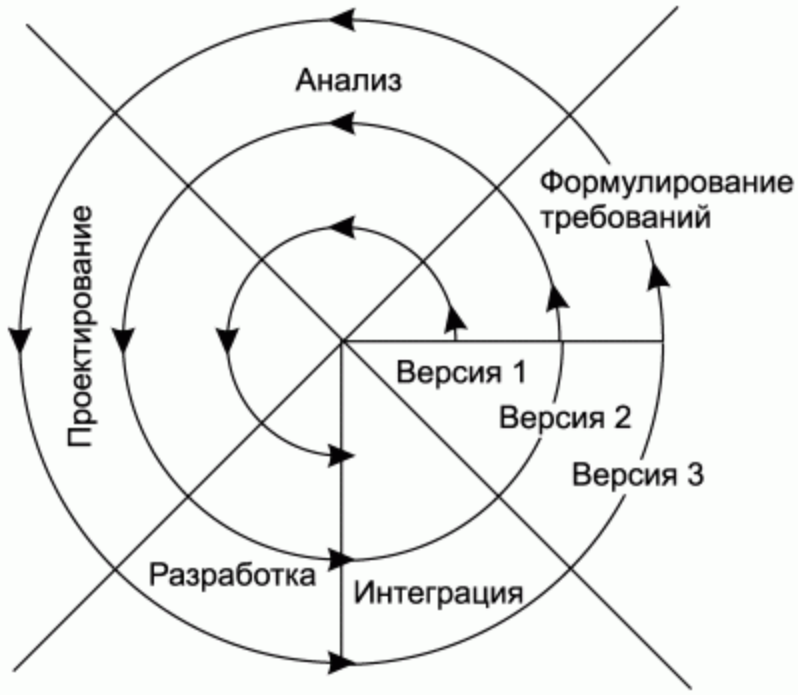
См. билет 21.

Итерационная модель.

См. билет 22.

Спиральная модель.

Отличительной особенностью этой модели является специальное внимание рискам.



21. Последовательная модель жизненного цикла и ее модификации.

Последовательная (каскадная) модель включает себя следующие этапы по порядку:

1. анализ
2. проектирование
3. реализация
4. тестирование
5. внедрение
6. сопровождение

Каскадный подход хорошо зарекомендовал себя при построении систем, для которых в самом начале разработки можно чётко сформулировать требования. Губительным недостатком оказалось запаздывание с получением обратной связи от пользователей. Таким образом, появилась разновидность с обратной связью, т.е. с возможностью возвращаться на предыдущие этапы.

22. Эволюционные и инкрементные модели жизненного цикла.

Итерация — мини-проект, этапы которого совпадают с этапами из последовательной модели.

Инкремент — новые возможности, появившиеся в результате итерации.

Т.о., проект разбивается на ряд итераций и эволюционирует, что позволяет оперативно реагировать на желания пользователей. Однако, долгое время (до первого прототипа) целостное понимание проекта отсутствует.

23. Жизненный цикл ПО и методологии программирования. Жесткие и гибкие стратегии в методологиях программирования, их характерные черты и границы применимости. Примеры.

ЖЦПО — это период с момента принятия решения о создании ПО до изъятия из эксплуатации.

Методология — стратегия, предписываемая для построения и выполнения системы деятельностей, набор методов, средств и инструментов, которые согласованы с этой стратегией.

Примеры методологий:

- Rational Unified Process (RUP)
- Модель процессов MSF
- Экстремальное программирование (XP)

Жесткие методологии	***Гибкие методологии***
Ориентация на предсказуемые процессы разработки программного обеспечения с четко обозначенными целями	Осознание того, что процессы разработки программного обеспечения в принципе непредсказуемы
Распознавание ситуаций и применение готовых методов	Распознавание ситуаций и конструирование методов для работы в них
Планирование, в котором определяются этапы с объемом работ, ресурсами, сроками и уровнем качества работ	Соблюдение баланса между параметрами проекта: объем работ, ресурсы, сроки и уровень качества работ
Заказчик — внешний по отношению к проекту субъект, влияющий на разработку только через предоставление ресурсов и контроль результатов, в том числе по поэтапным срокам выполнения проекта	Заказчик (его представитель) — член команды разработчиков, наделенный правом влиять на разработку; его главной целью является отслеживание актуальности решаемых задач
Ролевое разделение труда работников проекта	Совместная деятельность сотрудников и деперсонифицированная ответственность
Дисциплина и подчинение	Самодисциплина и сотрудничество
Обезличенный процесс, исполнители которого определяются только по квалификационным требованиям	Процесс, максимально учитывающий личностные качества исполнителей

24. Защитное программирование. Принципы защитного программирования. Рекомендации по реализации защитного программирования.

Защитное программирование — стиль написания программ, основанный на трёх основных принципах:

- входные данные каждого модуля должны тщательно анализироваться в предположении, что они ошибочны
- каждая программная ошибка должна быть выявлена как можно раньше, что упрощает установление ее причины
- ошибки в одном модуле должны быть изолированы так, чтобы не допустить их влияние на другие модули

Основные рекомендации:
- использование утверждений каждый раз, когда программист делает какие-то предположения
- реализация ветки default в switch'ах
- использование автоматической проверки границ, контроль за переполнением и т.д.
- контроль правдоподобности значений

25. Защитное программирование. Подходы к выбору метода обработки ошибки. Утверждения. Условная компиляция.

Программная ошибка — конфликт, возникший в ходе нормального функционирования программы. Например, переполнение памяти или нехватка прав на чтение файла.

Методы возбуждения ошибки:
- вернуть некорректное значение из функции (синхронный код)
- записать в переменную-ошибку (как errno в си)
- бросить исключение (синхронный код)
- вызвать соответствующее событие (асинхронный код)
- вызвать функцию-келлбек с объектом-ошибкой (асинхронный код)

Методы обработки:
- устранение ошибки. Например, создание файла в случае его отсутствия
- информирование пользователя
- повторение операции. Например, повторный запрос в ответе на ошибку 503.
- прекращение работы программы. Например, нехватка памяти.
- запись в лог-файл и продолжение работы.

Обычно, методы совмещаются.

Ошибка программиста — дефекты кода, приводящие к некорректному функционированию программы. Например, вызов функции с некорректными параметрами. Такие ошибки не обрабатываются.

Методы возбуждения ошибки:
- бросить исключение (если функция является публичным интерфейсом)
- утверждение (если функция явялется приватной для модуля)

Утверждения — конструкция (или функция/макрос) ЯП, при несоблюдении которой программа аварийно завершается. Условная компиляция позволяет разворачивать утверждения в пустой оператор, чтобы не тормозить код в продакшене.