**1. Понятие корректности программного кода**

**Корректность программы** означает её полное соответствие установленным техническим требованиям и спецификациям. Такая программа должна:

* Обрабатывать все разрешенные входные значения
* Соблюдать начальные условия выполнения
* Формировать ожидаемые выходные данные
* Соответствовать заявленным функциональным свойствам

**Различие между корректностью и надежностью:**  
Корректность отражает соответствие требованиям, а надежность - устойчивость к сбоям при длительной работе.

Примеры:

1. *Надежная некорректная система*: Банковское приложение, стабильно работающее (надежность), но иногда производящее ошибочные расчеты (некорректность)
2. *Корректная ненадежная система*: Математический алгоритм, всегда дающий точный результат (корректность), но периодически зависающий при обработке больших данных (ненадежность)

**2. Формальные требования к программному обеспечению**

**Техническая спецификация** представляет собой строгое описание:

* Предполагаемого поведения системы
* Допустимых параметров ввода
* Требуемых выходных значений
* Условий выполнения операций

Отсутствие спецификации делает невозможным объективную оценку корректности программы.

**Основные виды требований:**

1. Функциональные - перечень обязательных возможностей системы
2. Предварительные условия - требования к состоянию перед выполнением
3. Постусловия - гарантированные результаты при соблюдении условий
4. Инварианты - неизменные свойства в течение работы

**Пример для функции calculate\_positive\_average:**

* *Предварительное условие*:

python

Copy

Download

assert all(n > 0 for n in numbers) and len(numbers) > 0

Входной массив должен содержать хотя бы один элемент, все числа положительные

* *Гарантируемый результат*:

python

Copy

Download

assert result > 0 and abs(result - sum(numbers)/len(numbers)) < 0.001

Возвращаемое значение - среднее арифметическое с заданной точностью

**3. Способы верификации корректности**

**Практическое тестирование:**

* *Принцип работы*: Проверка на конкретных тестовых примерах
* *Преимущества*:
  + Легкость создания тестов
  + Доступность для разработчиков разного уровня
* *Недостатки*:
  + Ограниченность проверяемых случаев
  + Отсутствие полной гарантии
* *Тестовый сценарий*: Конкретный пример ввода и ожидаемого вывода

**Анализ исходного кода:**

* *Суть метода*: Проверка текста программы без запуска
* *Обнаруживаемые проблемы*:
  + Ошибки синтаксиса
  + Отклонения от стандартов
  + Потенциально опасные конструкции
* *Пример инструмента*: SonarQube для выявления проблем с памятью

**Математические методы верификации:**

* *Основная идея*: Формальное доказательство соответствия спецификации
* *Инвариант цикла*: Логическое условие, сохраняющееся при каждом повторении
* *Цель применения*: Обоснование правильности алгоритмов

**Выбор метода проверки:**

* Для компактных функций: Комплекс тестов + статический анализ
* Для сложных систем: Формальная верификация ключевых модулей + тестирование + анализ кода

**Практические советы:**

1. Начинайте с детальной спецификации
2. Для ответственных систем применяйте формальные методы
3. Автоматизируйте процессы проверки
4. Помните об ограничениях каждого подхода