**1. Понятие корректности программного кода**

Корректность программы подразумевает её строгое соответствие техническим требованиям и спецификациям. Корректная программа должна:

* **Обрабатывать все допустимые входные данные** в рамках заданных параметров
* **Соблюдать предусловия** перед началом выполнения операций
* **Генерировать ожидаемые выходные значения** согласно требованиям
* **Соответствовать заявленному функциональному поведению**

**Отличие корректности от надежности:**

* Корректность = соответствие требованиям
* Надежность = устойчивость к сбоям при продолжительной эксплуатации

**Примеры:**

1. *Надежная, но некорректная система*: Платежный сервис, стабильно работающий (надежность), но иногда допускающий ошибки в расчетах (некорректность)
2. *Корректная, но ненадежная система*: Алгоритм шифрования, всегда дающий правильный результат (корректность), но периодически аварийно завершающийся при больших объемах данных (ненадежность)

**2. Формальные требования к ПО**

Техническая спецификация должна четко определять:

* Ожидаемое поведение системы
* Допустимые диапазоны входных параметров
* Требуемые выходные значения
* Условия выполнения операций

**Без спецификации невозможно объективно оценить корректность программы.**

**Основные категории требований:**

1. **Функциональные** - обязательные возможности системы
2. **Предусловия** - требования к состоянию перед выполнением
3. **Постусловия** - гарантированные результаты при соблюдении условий
4. **Инварианты** - свойства, сохраняющиеся в течение работы

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

*Предусловие:*

python

Copy

Download

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

(Массив должен содержать минимум один положительный элемент)

*Постусловие:*

python

Copy

Download

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

(Результат должен быть положительным средним значением с заданной точностью)

**3. Методы верификации корректности**

**1. Практическое тестирование**

* *Принцип:* Проверка на конкретных тестовых случаях
* *Плюсы:*
  + Простота создания тестов
  + Доступность для разработчиков любого уровня
* *Минусы:*
  + Ограниченный охват возможных сценариев
  + Не дает полной гарантии корректности

**2. Статический анализ кода**

* *Суть:* Проверка исходного кода без его выполнения
* *Выявляет:*
  + Синтаксические ошибки
  + Отклонения от стандартов кодирования
  + Потенциально опасные конструкции
* *Инструменты:* SonarQube, Pylint, ESLint

**3. Формальные методы верификации**

* *Подход:* Математическое доказательство соответствия спецификации
* *Применение:*
  + Инварианты циклов
  + Доказательство правильности алгоритмов
  + Верификация критически важных компонентов

**Рекомендации по выбору методов:**

* Для небольших функций: комбинация тестов и статического анализа
* Для сложных систем: формальная верификация ключевых модулей + тестирование + анализ кода

**Практические советы:**

1. Всегда начинайте с детальной спецификации
2. Для критических систем используйте формальные методы
3. Автоматизируйте процессы проверки
4. Учитывайте ограничения каждого подхода
5. Комбинируйте разные методы для комплексной проверки

**Важно:** Ни один метод не дает 100% гарантии корректности, поэтому рекомендуется использовать комбинацию подходов в зависимости от важности и сложности системы.