**1. Корректность программного обеспечения**

**Определение корректности программы:**  
Программа считается корректной, если её поведение полностью соответствует формальным требованиям и спецификациям. Это означает, что:

* Для всех допустимых входных данных
* При соблюдении всех предусловий
* Программа выдаёт ожидаемые результаты
* И удовлетворяет всем заявленным свойствам

**Отличие корректности от надежности:**  
Корректность — это соответствие спецификации, тогда как надежность — способность работать без отказов в течение времени.

Пример:

* *Надежная, но некорректная программа*: Система обработки платежей, которая всегда завершает операции (надежность), но иногда начисляет неверные суммы (некорректность).
* *Корректная, но ненадежная программа*: Алгоритм сортировки, который всегда дает правильный результат (корректность), но иногда аварийно завершается при больших объемах данных (ненадежность).

**2. Спецификации корректности**

**Спецификация программы** — это формальное описание:

* Ожидаемого поведения
* Допустимых входных данных
* Требуемых результатов
* Условий выполнения

Без спецификации невозможно определить корректность, так как нет эталона для сравнения.

**Виды спецификаций:**

1. **Функциональные требования** — что система должна делать
2. **Предусловия** — условия, которые должны быть истинны перед выполнением
3. **Постусловия** — гарантии после выполнения при соблюдении предусловий
4. **Инварианты** — условия, сохраняющиеся на протяжении всей работы

**Пример для функции calculate\_positive\_average(numbers):**

* *Предусловие*:

python

Copy

Download

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

Массив не пуст и содержит только положительные числа

* *Постусловие*:

python

Copy

Download

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

Возвращает среднее значение с точностью до 0.001

**3. Методы проверки корректности**

**Тестирование:**

* *Суть*: Выполнение программы на контрольных примерах
* *Сильные стороны*:
  + Простота реализации
  + Не требует глубоких математических знаний
* *Слабые стороны*:
  + Проверяет только конкретные случаи
  + Не гарантирует полной корректности
* *Тест-кейс*: Конкретный сценарий (входные данные + ожидаемый результат)

**Статический анализ:**

* *Суть*: Проверка кода без его выполнения
* *Выявляет*:
  + Синтаксические ошибки
  + Нарушения стиля
  + Потенциальные уязвимости
* *Пример*: SonarQube обнаруживает утечки памяти

**Формальные методы:**

* *Идея*: Математическое доказательство соответствия кода спецификации
* *Инвариант цикла*: Условие, истинное перед каждой итерацией и после неё
* *Назначение*: Доказательство правильности алгоритмов

**Сравнение методов:**

* *Для небольшой функции*: Комбинация тестирования (покрытие всех граничных случаев) и статического анализа
* *Для большой системы*: Формальные методы для критических компонентов + тестирование + статический анализ

**Рекомендации:**

1. Всегда начинайте с четкой спецификации
2. Для важных систем используйте формальные методы
3. Автоматизируйте тестирование и статический анализ
4. Помните: ни один метод не дает 100% гарантии корректности