**Ответы на вопросы.**

**Вариант 2**

**1) ООП - полное определение**

Объектно-ориентированное программирование — это парадигма, основанная на концепции "объектов", которые содержат данные (поля/атрибуты) и поведение (методы). Ключевые принципы:

* **Инкапсуляция**: объединение данных и методов с контролем доступа (private/protected/public)
* **Наследование**: создание новых классов на основе существующих
* **Полиморфизм**: возможность объектов разных классов обрабатываться через общий интерфейс
* **Абстракция**: сокрытие сложной реализации за простыми интерфейсами

**2) Процедурная - Структурная декомпозиция**

Структурная декомпозиция — разбиение сложной задачи на иерархию более простых подзадач. Основная цель — упростить планирование, управление, контроль и распределение ресурсов, разделив большую цель на понятные и достижимые этапы.

**3) Архитектура ПО - определение, примеры**

Архитектура ПО — фундаментальная организация системы, воплощенная в ее компонентах, их отношениях друг к другу и среде, и принципах, управляющих ее проектированием и развитием.

* **Монолит**: традиционное приложение с единой кодовой базой
* **Микросервисы**: Uber с разделением на сервисы геолокации, платежей, уведомлений
* **Событийно-ориентированная**: торговые платформы с обработкой рыночных событий
* **Слоеная**: веб-приложения с presentation/business/data access слоями

**4) Энтропия ПО, не менее 3 примеров**

Энтропия ПО — мера хаоса и беспорядка в кодовой базе, ведущая к снижению поддерживаемости.

* **Растущая связанность**: класс User начинает зависеть от EmailService, PaymentProcessor, AnalyticsTracker — нарушение SRP
* **Дублирование кода**: одинаковые 200 строк валидации в 10 разных местах системы
* **Унаследованный технический долг**: легаси-код с устаревшими паттернами, который "слишком дорого" переписывать
* **Расплывающиеся интерфейсы**: класс-бог Utils с 50+ несвязанными методами

**5) Закон Иерархических Компенсаций Седова в контексте АйТи**

Закон Иерархических Компенсаций Седова в IT описывает баланс между разнообразием и управляемостью в сложных системах: чем больше свободы на нижнем уровне, тем больше ограничений требуется на верхнем, и наоборот. Это помогает анализировать архитектуру, технологии и организационные подходы.

* **Кэширование**: ускоряет чтение, но усложняет инвалидацию и согласованность данных
* **Микросервисы**: улучшают масштабируемость, но добавляют сетевые задержки и сложность оркестрации
* **NoSQL**: дает горизонтальное масштабирование, но жертвует ACID-транзакциями
* **Контейнеризация**: упрощает деплой, но добавляет overhead и сложность управления кластером
* **Low-code платформы**: ускоряют разработку, но ограничивают гибкость и создают vendor lock-in
* **Agile/Scrum**: увеличивает адаптивность, но добавляет бюрократию митингов и документации
* **Автоматическое тестирование**: улучшает качество, но замедляет разработку и требует поддержки тестов