**Variant 2 — Answers**

**1) OOP – Full Definition**

Object-Oriented Programming (OOP) is a paradigm based on the concept of "objects" that encapsulate data (fields/attributes) and behavior (methods). Key principles include:

* **Encapsulation**: combining data and methods with controlled access (private/protected/public)
* **Inheritance**: creating new classes based on existing ones
* **Polymorphism**: enabling objects of different classes to be processed through a common interface
* **Abstraction**: hiding complex implementation behind simple interfaces

**2) Procedural – Structural Decomposition**

Structural decomposition is the process of breaking down a complex task into a hierarchy of simpler subtasks. The main goal is to simplify planning, management, control, and resource allocation by dividing a large objective into understandable and achievable stages.

**3) Software Architecture – Definition and Examples**

Software architecture is the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles guiding its design and evolution. Examples:

* **Monolith**: a traditional application with a single codebase
* **Microservices**: Uber split into services for geolocation, payments, notifications
* **Event-driven**: trading platforms processing market events
* **Layered**: web applications with presentation/business/data access layers

**4) Software Entropy – At Least 3 Examples**

Software entropy is a measure of chaos and disorder in the codebase, leading to reduced maintainability. Examples:

* **Increasing coupling**: the User class starts depending on EmailService, PaymentProcessor, AnalyticsTracker — violating the Single Responsibility Principle (SRP)
* **Code duplication**: the same 200 lines of validation logic repeated in 10 different parts of the system
* **Inherited technical debt**: legacy code with outdated patterns that is "too expensive" to rewrite
* **Bloated interfaces**: a god-class Utils with 50+ unrelated methods

**5) Sedov’s Law of Hierarchical Compensation in IT**

Sedov’s Law in IT describes the balance between diversity and manageability in complex systems: the more freedom at the lower level, the more constraints are required at the upper level — and vice versa. It helps analyze architecture, technologies, and organizational strategies. Examples:

* **Caching**: speeds up reading, but complicates invalidation and data consistency
* **Microservices**: improve scalability, but add network latency and orchestration complexity
* **NoSQL**: enables horizontal scaling, but sacrifices ACID transactions
* **Containerization**: simplifies deployment, but adds overhead and cluster management complexity
* **Low-code platforms**: accelerate development, but limit flexibility and create vendor lock-in
* **Agile/Scrum**: increases adaptability, but introduces meeting and documentation overhead
* **Automated testing**: improves quality, but slows development and requires test maintenance