Main Design Concepts:

1. **Abstraction**:
   - **Explanation**: Abstraction is the process of simplifying complex systems by focusing only on the essential details and hiding unnecessary ones.
   - **Example**: Think of a car. You don't need to know the working of the engine to drive it. Similarly, in software, only the necessary details are shown to the user.
2. **Modularity**:
   - **Explanation**: Modularity involves breaking down a system into smaller, independent parts or modules that can be developed and tested separately.
   - **Example**: In a music app, the module for playing songs is separate from the module for managing the playlist.
3. **Encapsulation**:
   - **Explanation**: Encapsulation means hiding the internal workings of a module and exposing only what is necessary. It protects the data within a module.
   - **Example**: In a login system, the process of verifying user details is hidden from the rest of the system, ensuring security.
4. **Cohesion**:
   - **Explanation**: Cohesion measures how closely related and focused the functions of a module are. High cohesion means a module has a single, well-defined purpose.
   - **Example**: A "User Authentication" module should only deal with user login/logout processes, not other unrelated tasks.
5. **Coupling**:
   - **Explanation**: Coupling refers to how much one module depends on another. Low coupling is desirable because it means modules can function independently.
   - **Example**: A "Payment" module should not heavily depend on a "User Profile" module. If they are loosely coupled, changes to one won't break the other.
6. **Hierarchy**:
   - **Explanation**: Hierarchy is the organization of modules or components in levels, where higher levels control or depend on lower ones.
   - **Example**: A "Dashboard" might be at a higher level and display data from lower-level components like "User Info" and "Reports."
7. **Separation of Concerns**:
   - **Explanation**: This principle states that different aspects of a system (like design, data management, and functionality) should be handled separately.
   - **Example**: In a web application, the user interface (UI) design should be separate from the business logic (how the app works).

Why These Concepts Matter:

- **Better Maintenance**: When software is well-organized, it’s easier to find and fix bugs or add new features.
- **Reusability**: Modules can be reused in different projects.
- **Improved Efficiency**: Well-structured software runs more smoothly and is easier to understand.