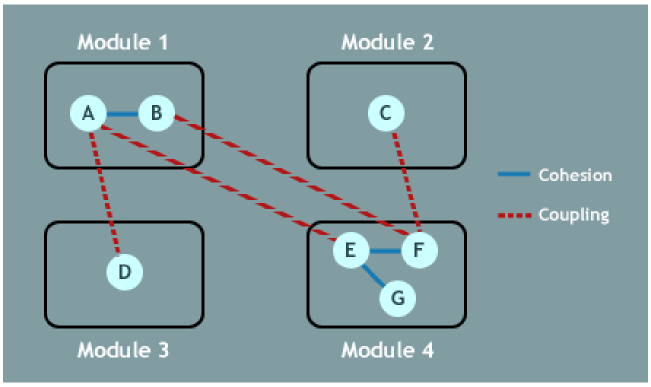
Slide 07 : Software Design Strategies

Made by Md. Mehedi Hasan Rafy

Software Design

Software design means planning how to turn requirements into working software by finding the best solution.

Structured Design



- Breaks the problem into smaller parts (modules).
- Based on divide and conquer.
- Helps designers focus better and solve each small problem clearly.
- Modules are organized in hierarchy and communicate with each other.

Good structured design has:

- **High Cohesion** (modules do one specific task)
- **Low Coupling** (less dependency between modules)

Function-Oriented Design

- System is divided into functions (small sub-systems) that perform specific tasks.
- Follows structured design and divide and conquer approach.
- Focus is on functions and data flow, not system state.

Key Points:

- Functions pass data among each other or use global data.
- Function call may change program state, which can be risky.
- Best suited when system state is not important.

Design Process

1. Break system into functions based on their roles.
2. Use **DFD (Data Flow Diagram)** to show how data moves and changes.
3. Describe each function in detail.

Object-Oriented Design (OOD)

Focuses on objects (real-world entities) and their attributes and behaviors, not just functions.

Key Concepts:

- ***Object****:* Entity with data (attributes) and actions (methods). Example: Person, Bank.
- ***Class****:* Blueprint of an object. Object = instance of a class.

Main Features:

1. ***Encapsulation****:* Bundles data and methods together; hides internal details (information hiding).
2. ***Inheritance****:* Subclasses reuse features of parent classes.
3. ***Polymorphism****:* Same method name works differently based on input.

Design Process:

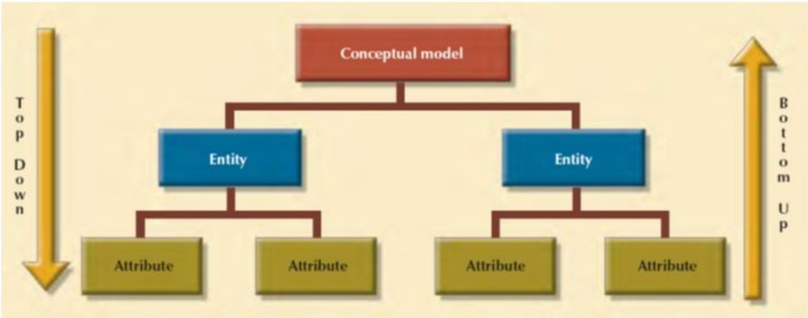
1. Create design from requirements or sequence diagrams.
2. Identify objects and group them into classes.
3. Define class hierarchy and relationships.
4. Build application framework.

Software Design Approaches

Software systems are made of subsystems and components arranged hierarchically.

1. ***Top-Down Design:***

- Start with the whole system as one piece.
- Break it down into smaller subsystems or components step by step.



Top-down vs. bottom-up design sequencing

- Continue until smallest parts are defined.
- Best when designing from scratch and details are unknown.

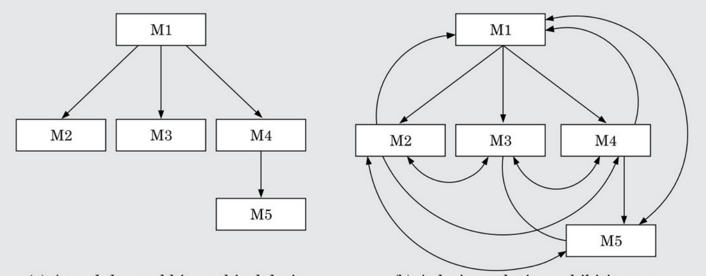
2. ***Bottom-Up Design:***

- Start with small basic components.
- Combine them step by step to form higher-level components.
- Continue until the full system is formed.
- Best when building on existing components or systems.

Analysis vs. Design

- Analysis focuses on understanding what the system should do; it is generic and platform-independent.
- Design focuses on how to implement the system; it considers specific implementation details.
- Analysis uses models like:
  - ***Function-oriented:*** Data Flow Diagrams (DFDs)
  - ***Object-oriented:*** UML diagrams
- Design models come from transforming analysis models step-by-step.
- Analysis models are usually hard to implement directly in code.

Comparison of Modularity



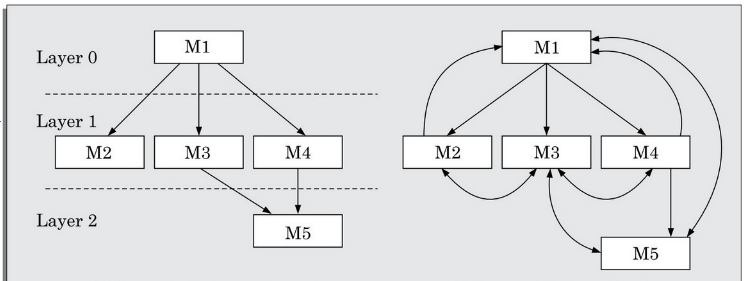
(a) A modular and hierarchical design (b) A design solution exhibiting poor modularity and hierarchy

- Modularity compares how well a design breaks a system into modules.
- Intuitively, a design with fewer interactions between modules is easier to understand.
- There is no direct quantitative metric to measure modularity yet but modularity can be judged by cohesion and coupling:
  - ***High cohesion*** (modules focused on a single task)
  - ***Low coupling*** (modules with minimal dependencies)
- Designs with high cohesion and low coupling make development easier and reduce complexity.

Layered Design

- Modules are arranged in layers forming a tree-like structure.
- A module can only call modules in the layer immediately below it.
- Higher layers act like managers controlling lower layers.
- It supports control abstraction: lower layers don’t know about higher layers.
- Debugging is easier because errors in a module usually come from modules below it.

Layered Arrangement of Modules



(a) Layered design with good control abstraction
(b) Layered design showing poor control abstraction

- Control hierarchy shows how modules call each other, usually in a tree diagram (structure chart).
- In layered design, modules are arranged in layers based on call relations.
- A module can only call modules in lower layers, never in the same or higher layers.
- Top layer modules act as managers, calling lower layers to get work done.
- Middle layers both perform tasks and call lower layers.
- Bottom layers are worker modules that do all work themselves and call no one.
- If all modules call each other freely (no layers), design is not layered.

Layered Design Terminologies

- ***Superordinate module:*** Controls another module (higher level).
- ***Subordinate module:*** Controlled by another module (lower level).
- ***Visibility:*** Module B is visible to A if A directly calls B (only immediate lower layer modules are visible).
- ***Control Abstraction:*** Higher layers are hidden from lower layers; modules only call the layer directly below.
- ***Depth:*** Number of layers in the design.
- ***Width:*** Number of modules at a layer (span of control).
- ***Fan-out:*** Number of modules a module controls (calls). In (a) Fan-out of M1 is 3.
    - High fan-out (>7) means poor cohesion (module does too many tasks).
- ***Fan-in:*** Number of modules calling a module. In (a) Fan-in of M5 is 2.
    - High fan-in means good code reuse.

Functional Independence

A module is functionally independent if it does one task and has minimal interaction with others.

- ***Error Isolation:*** Errors stay within the module, making bugs easier to find and fix.
- ***Reuse:*** Independent modules with clear interfaces can be reused in other programs easily.
- ***Understandability:*** Independent modules reduce complexity, making the system easier to understand.