SOFTWARE ENGINEERING

UNIT - 3

TOPIC – 2

DESIGN CONCEPTS AND DESIGN MODEL

1. Key Design Concepts

Several important ideas are used in the design process. These concepts help us plan and build software better:

a. Abstraction

Abstraction means **simplifying** complex things by breaking them into smaller parts. For example, instead of focusing on how a car's engine works, you can just think of the car as something that gets you from place A to B.

There are three types:

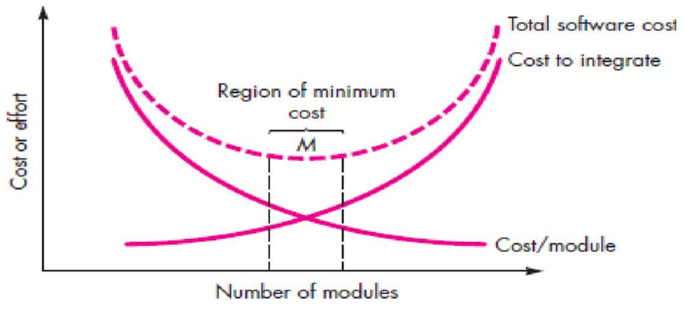
- **Procedural Abstraction**: Describes a process or task without getting into details.
- **Data Abstraction**: Groups related data together, like having all the details of a "Student" in one place.
- **Control Abstraction**: Simplifies the control flow (like making it easier to manage how different parts of a program interact).

b. Refinement

Refinement is about starting with a big idea and **breaking it into smaller, more detailed steps**. For example, instead of just saying "manage student records," we can break that down into "add a student," "update student details," and "delete a student."

c. Modularity

Modularity means splitting the software into small, independent parts called **modules**. Each module handles one task. For example, in a school system, there could be a module for managing students and another module for managing courses.



d. Architecture

The **architecture** is like the **blueprint** of the software. It shows how the **big parts** of the software (like different modules) will work together and how they will communicate. A good architecture makes the software easier to build and maintain.

e. Information Hiding

This concept is about **keeping details hidden** inside each module, only showing what's necessary. For example, a module that handles student grades might keep its internal calculations hidden and only give out the final grades. This reduces the risk of errors spreading through the system when changes are made.

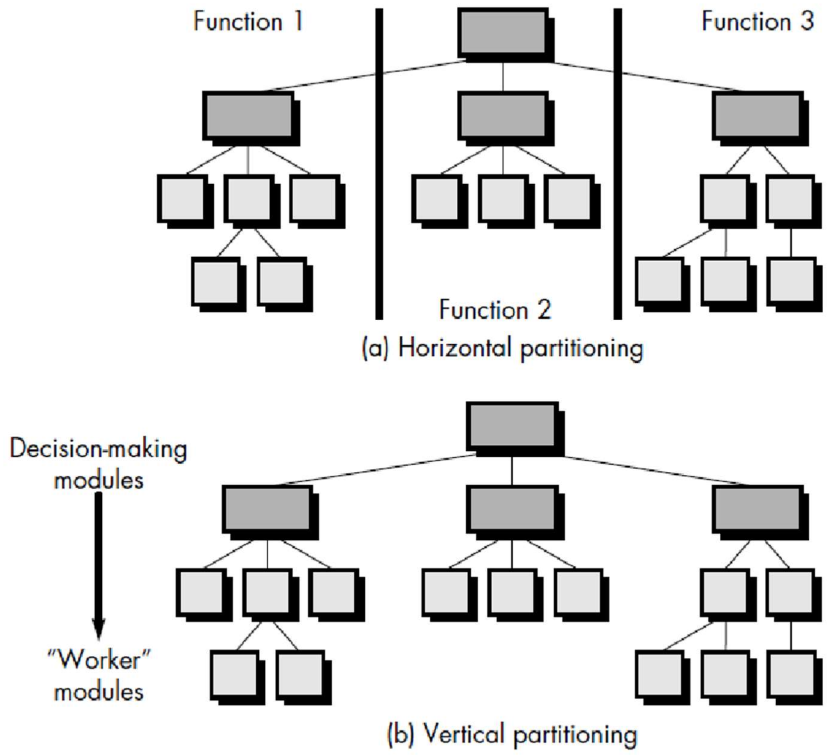
f. Control Hierarchy

This defines how **control flows** through the system. Some parts of the system control others. For example, in a school system, the **admin module** might control the **student registration module**. There's a hierarchy, just like how a manager controls employees in a company.

g. Structural Partitioning

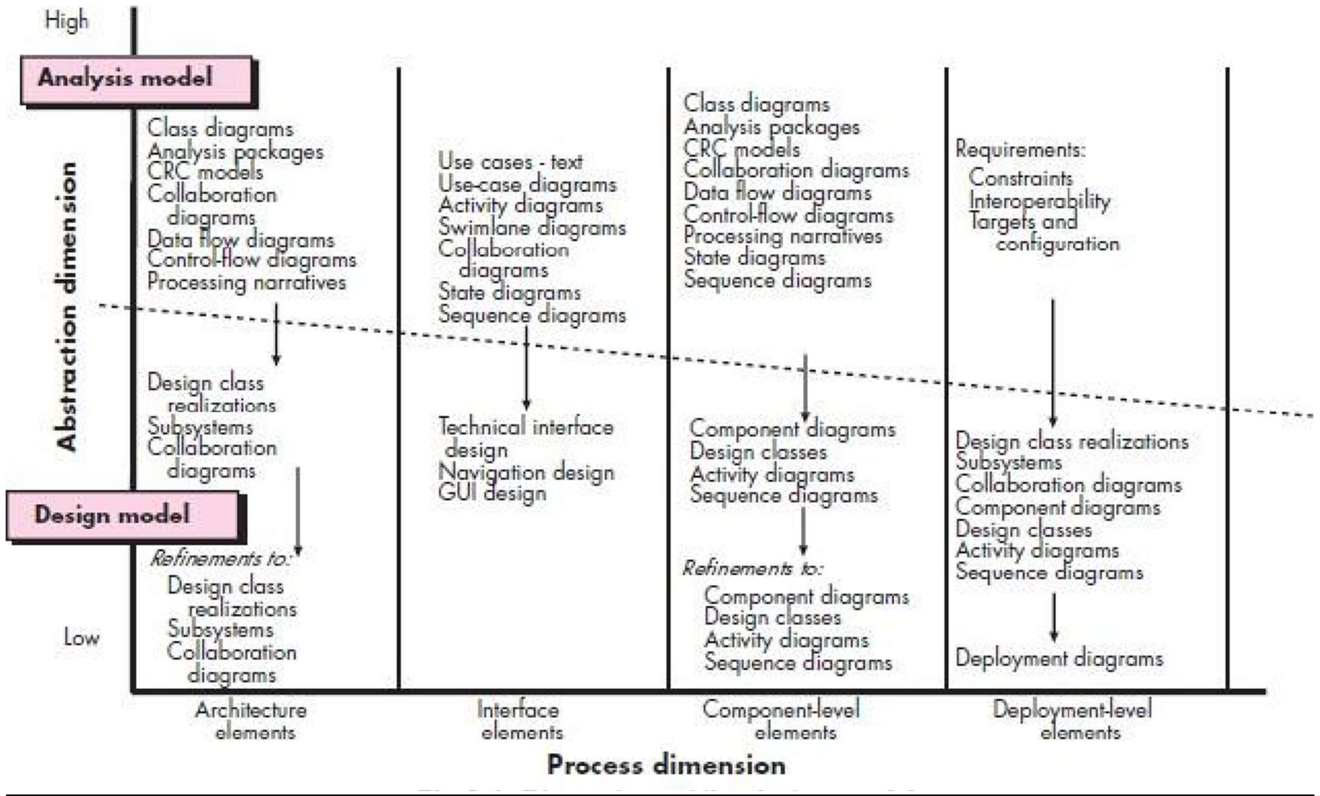
We divide the system into smaller parts, based on the way the system works:

- **Horizontal Partitioning**: Divides the system into layers, like input, processing, and output.
- **Vertical Partitioning**: Organizes the system so that control flows from the top (higher-level decisions) down to the bottom (detailed tasks).



2. The Design Model

The design model is a detailed plan of how the software will be built, and it evolves over time. The design model uses **UML diagrams** to represent different aspects of the system.



Here are the main parts of the design model:

Data Design Elements

This part of the design focuses on how **data** will be organized and structured in the system. For example, a school system would need to store student data, teacher data, and course data.

Architectural Design Elements

These describe the **big structure** of the software. Think of it as a **floor plan** of a house showing how different parts are connected. It gives an overall view of how the software will function.

Interface Design Elements

This part describes how the software will interact with:

- **Users**: through a user interface (UI) like buttons, menus, and forms.
- **Other systems**: through external connections like APIs.
- **Internal components**: showing how different parts of the software will communicate with each other.

Component-Level Design Elements

This describes the **small parts** of the software in detail. For example, a "Login" component might handle checking usernames and passwords. Each component works independently but connects to others to form the full system.

Deployment-Level Design Elements

This shows how the software will be **set up in the real world**. It describes where each part of the software will run (for example, on a server or a user's computer).

7. UML Diagrams

UML diagrams are visual tools to help explain the design. Common types include:

- **Class Diagram**: Shows the structure of the system in terms of classes (e.g., a "Student" class with data like name and grades).
- **Use Case Diagram**: Shows how users will interact with the system (e.g., a student registering for a course).
- **Sequence Diagram**: Shows the sequence of steps between system parts (e.g., when a student logs in).
- **Activity Diagram**: Looks like a flowchart and shows steps in a process (e.g., registering a student).
- **State Diagram**: Shows different states an object can be in (e.g., a student registration could be "pending" or "approved").
- **Component Diagram**: Shows how the system is divided into parts that do specific jobs.
- **Deployment Diagram**: Shows how the software will be set up, like what parts will run on a server.