SOFTWARE ENGINEERING

UNIT - 3

TOPIC – 1

DESIGN PROCESS AND DESIGN QUALITY

1. Definition of Design Engineering

Design Engineering is the step where we take **what the software should do** (from the requirements) and figure out **how it will do it**. Think of it like building a house: you know how many rooms you want (requirements), but now you need to plan how to build the house (design). This design becomes the blueprint that guides the entire development.

The main parts of the design include:

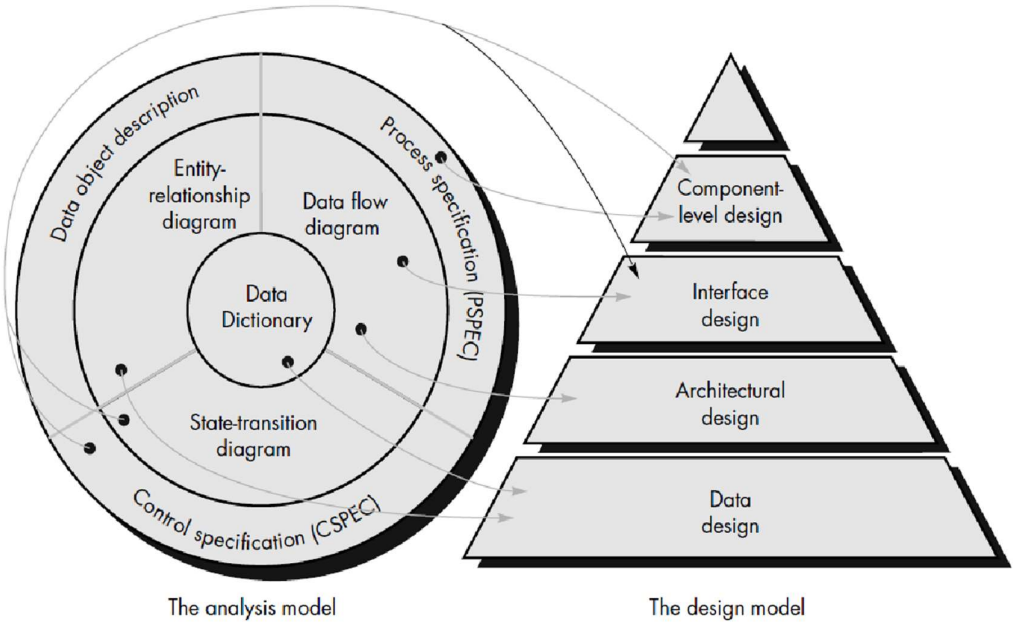
- **Data/Class Design**: Organizing the information the software will use.
- **Architecture**: How the big pieces of the software fit together.
- **Interface Design**: How the software interacts with users or other systems.
- **Component Design**: The smaller parts (or components) of the system that each do a specific job.

2. Turning Analysis into Design

We start with the **requirements model** (which describes what the software should do) and turn it into the **design model** (which shows how the software will do it).

This is done in steps:

1. **Data/Class Design**: This step is about defining how data will be structured. Imagine in a school system you need to store data about students, teachers, and courses. This design will tell you how to organize that data into **classes**.



2. **Architecture**: Here, we decide how the **big parts** of the system are connected. For example, how the **student data module** will connect to the **course management module**.
3. **Interface Design**: This step is about planning how the software will communicate with the outside world. It defines how users will interact with the system and how different parts of the software talk to each other.
4. **Component-Level Design**: Finally, we look at the smaller building blocks of the software. Each **component** (like the login feature or registration feature) is designed in detail.

3. The Design Process

The design process doesn't happen in one go; it's done in steps and refined over time. With each iteration, the design becomes more detailed and clearer.

Throughout this process, the design is checked for **quality** to ensure:

- The design meets **all the requirements**.
- It is **understandable** so that others can build, test, and maintain it easily.
- The design provides a complete picture of how the software will work.

4. Quality Attributes (FURPS)

To ensure the design results in a high-quality product, we follow the **FURPS** model, which stands for:

- **Functionality:** Evaluates if the system has all the required features and works as expected.
    - **Example**: An online banking system should allow users to transfer money, check their balance, and pay bills securely.
- **Usability:** Measures how easy it is for users to interact with the system.
    - **Example**: A weather app that clearly shows the forecast and is easy to navigate.
- **Reliability:** Assesses if the system performs consistently over time without errors.
    - **Example**: A payment gateway should process transactions without failure.
- **Performance:** Refers to how quickly the system operates.
    - **Example**: An e-commerce website should load pages quickly and handle a large number of users during sales.
- **Supportability:** Refers to how easy it is to maintain, upgrade, and test the system.
    - **Example**: A mobile app that can easily be updated to add new features

These attributes guide the design process to ensure the final software product is robust and meets user expectations.

5. Design Guidelines for Good Software

To make sure the design is good, we follow these rules:

1. **Familiar Structure**: Use well-known design patterns or styles and create parts that are designed well and can grow over time.

2. **Modular Design**: Split the system into smaller, manageable parts (modules) that can work independently.
3. **Clear Representation**: The design should clearly show how data, architecture, interfaces, and components are organized.
4. **Good Class Structure**: Make sure the design leads to proper class structures that will be used in the software.
5. **Independent Parts**: Each part should be able to function on its own without needing other parts too much.
6. **Simple Connections**: Keep the connections between parts of the system simple, especially how the system interacts with the outside world.
7. **Based on Requirements**: The design should come from the information gathered during the requirements phase.
8. **Clear Communication**: Use simple diagrams or explanations to show how the design works.