SNS COLLEGE OF ENGINEERING

Kurumbapalayam (po), Coimbatore – 641 107

Accredited by NAAC-UGC with 'A' Grade

Approved by AICTE & Affiliated to Anna University, Chennai

DEPARTMENT OF INFORMATION TECHNOLOGY

Unit 3

Architectural Styles in Object-Oriented Software Engineering

Architectural styles refer to common patterns or strategies used in designing software systems. They provide a blueprint or general approach for organizing the system's components and managing their interactions. The key architectural styles in OOSE include **Layered Architecture**, **Client-Server Architecture**, **Tiered Architecture**, **Pipe and Filter Architecture**, and **Model-View-Controller (MVC)**.

1. Layered Architecture

Layered architecture is a type of architectural style where the system is divided into distinct layers, each responsible for a specific aspect of the system's functionality. Layers are organized in such a way that each layer communicates only with the layer directly above or below it, ensuring separation of concerns.

Characteristics:

- **Separation of concerns**: Each layer handles a specific responsibility (e.g., presentation, logic, data).
- **Abstraction**: Layers hide their internal implementation details from other layers.
- **Loose coupling**: Each layer interacts only with adjacent layers, leading to reduced interdependence between components.

Common Layers:

- **Presentation Layer**: Handles the user interface and user interaction.

- **Business Logic Layer**: Contains the core application logic and rules.
- **Data Layer**: Manages data access, database interactions, and persistence.

Example:

- **Web Application**:
  - **Presentation Layer**: The frontend, which could be a web page rendered using HTML/CSS/JavaScript.
  - **Business Logic Layer**: The backend that handles processing, such as calculations or data manipulation.
  - **Data Layer**: A database (e.g., MySQL) that stores and retrieves data.

Example Use Case: In an e-commerce website, the user interface allows customers to browse products (presentation), the server processes orders and calculates prices (business logic), and the data layer stores product information, customer details, and order history.

2. Client-Server Architecture

Client-server architecture is a model where two components, a **client** and a **server**, interact over a network. The client sends requests, and the server processes those requests and responds accordingly.

Characteristics:

- **Centralized server**: The server holds and manages resources, while the client accesses and uses those resources.
- **Request-response mechanism**: The client sends requests, and the server processes them and sends a response.
- **Scalability**: The system can be scaled by adding more clients or more servers.

Example:

- **Web Browsing**:
  - **Client**: The user's web browser (e.g., Google Chrome) that sends requests for web pages.

- **Server**: The web server (e.g., Apache, Nginx) that handles the requests, processes them, and returns HTML content to the browser.

Example Use Case: A **web application** like Facebook or Google: The user (client) requests data like a status update or search result, and the web server (server) processes the request and sends back a response.

3. Tiered Architecture

Tiered architecture extends the concept of client-server by dividing the system into multiple physical or logical tiers. Typically, the system is split into three tiers: presentation, business logic, and data. Each tier can be deployed on a different machine or system, making the system distributed.

Characteristics:

- **Multiple tiers**: At least three tiers are commonly used: presentation, business logic, and data.
- **Distributed system**: Each tier can reside on different physical machines or processes.
- **Communication over a network**: Tiers typically communicate over a network.

Example:

- **Three-Tier Architecture**:
  - **Presentation Tier**: The user interface, such as a web browser or mobile app.
  - **Business Logic Tier**: The application server that contains the business rules and logic.
  - **Data Tier**: The database server that handles data storage and retrieval.

Example Use Case: An enterprise application with a **three-tier architecture**:

- The **client tier** (web or mobile interface) sends requests to the application server (business logic tier).

- The **business logic tier** processes the request and retrieves data from the **data tier** (database).
- Data is then sent back to the client tier.

4. Pipe and Filter Architecture

Pipe and filter is an architectural style where the system is composed of a sequence of processing steps (filters), where each filter performs a specific operation and the data flows between filters via pipes. This architecture is especially useful for data processing and transformation systems.

Characteristics:

- **Filters**: Each filter is a modular unit of processing that performs a well-defined task.
- **Pipes**: Data flows from one filter to another through pipes.
- **Loose coupling**: Filters are independent, and the entire system is easily extensible or modifiable.

Example:

- **Text Processing**:
  - **Filter 1**: Tokenizes a string of text into words.
  - **Filter 2**: Removes stop words (like "and," "the," etc.).
  - **Filter 3**: Analyzes sentiment by evaluating the text's positivity or negativity.

Example Use Case: A **log processing system** that reads logs from a file, processes the data in steps (parsing, filtering, and aggregating), and outputs the final processed data to another system or a report.

5. Model-View-Controller (MVC) Architecture

MVC is a widely used architectural pattern that separates an application into three interconnected components: **Model**, **View**, and **Controller**. This separation helps manage complexity, especially in user interfaces.

Characteristics:

- **Model**: Represents the data and the business logic of the application.
- **View**: Displays the data to the user and updates the display when the data changes.
- **Controller**: Acts as an intermediary between the Model and View. It processes user input and updates the Model or View accordingly.

Example:

- **Web Application** (e.g., a task manager):
  - **Model**: Manages task data (e.g., creating, updating, and deleting tasks).
  - **View**: Displays the task list to the user.
  - **Controller**: Handles user actions (e.g., adding a new task, deleting a task) and updates the model.

Example Use Case: An online shopping cart:

- **Model**: Contains data like product details, customer information, and order history.
- **View**: Displays product listings, shopping cart, and checkout page.
- **Controller**: Handles user actions like adding items to the cart or completing a purchase.