SOFTWARE ENGINEERING

UNIT - 3

TOPIC – 3

ARCHITECTURAL DESIGN

Architectural design in software engineering is the process of creating a **plan** or **blueprint** for how a software system will work. This plan defines the structure of the system, how the different parts of the software will interact, and how to make sure the system can be easily updated, scaled (grow), and maintained. It's similar to designing a house before building it, ensuring all parts like rooms, walls, and utilities work together.

Purpose of Architectural Design

The purpose of architectural design is to ensure that:

1. **The system is well-organized** and all parts work smoothly together.
2. **It is easy to maintain** and update when needed.
3. **It can handle growth**—more users, more data, or additional features.
4. **Everyone involved** (developers, managers, clients) understands how the system is supposed to work.

Key Elements of Architectural Design

1. Software Architecture

Software architecture is the **big-picture structure** of the system. It shows how the main parts of the software will be organized and how they will interact. The goal here is to make sure that all the components are in the right place and can work together effectively.

Example: In a **food delivery app**:

- **User Interface**: Where users browse and order food.
- **Payment System**: Where users pay for their orders.

- **Delivery Tracking**: Where users track their delivery status.

These components must connect and work together to provide a smooth user experience.

Purpose: To define the overall structure and ensure that all parts work together seamlessly.

2. Architectural Styles

Architectural styles are **different ways to structure** a software system. Each style has its own way of organizing the system's components to achieve the best performance and meet specific needs.

Types of Architectural Styles with Examples:

- **Layered Architecture**: The system is divided into layers, with each layer handling a specific task.

  Example: In a **banking app**:

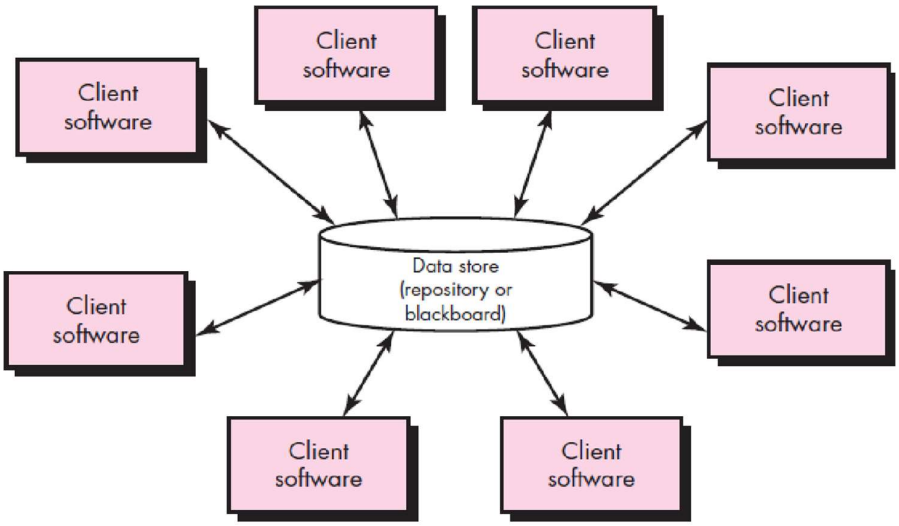
  - The **User Interface** shows account balances.
  - The **Business Logic** layer processes payments.
  - The **Data Layer** stores transaction data.

  Purpose: To organize tasks in layers so they can work separately but still communicate with each other.

- **Data-Centered Architecture**: The system revolves around a central **database**.

  Example: In a **library system**, all book and user information is stored in a central database.

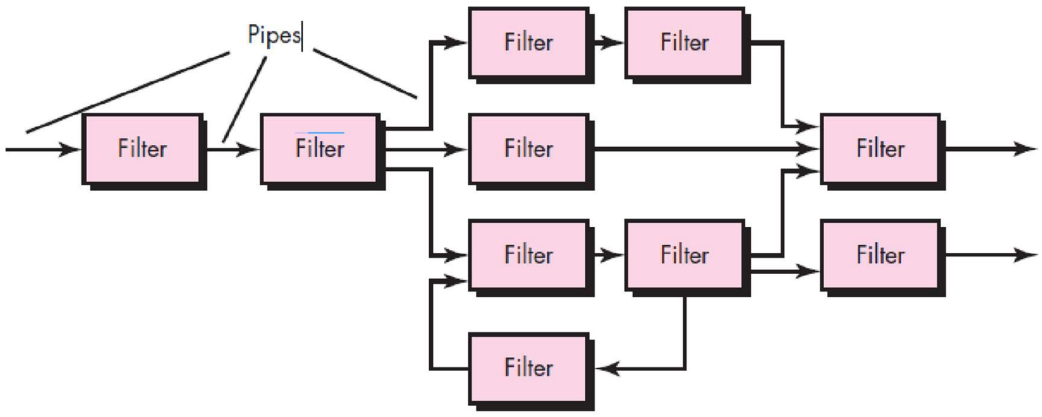
  Purpose: To centralize data storage and make sure all components use the same source of truth.



- **Data-Flow Architecture**: Data flows through different components and gets processed along the way.

  Example: In a **photo editing app**, an image passes through filters like brightness and contrast adjustments before being saved.

  Purpose: To process data step-by-step, allowing easy control of how data changes.

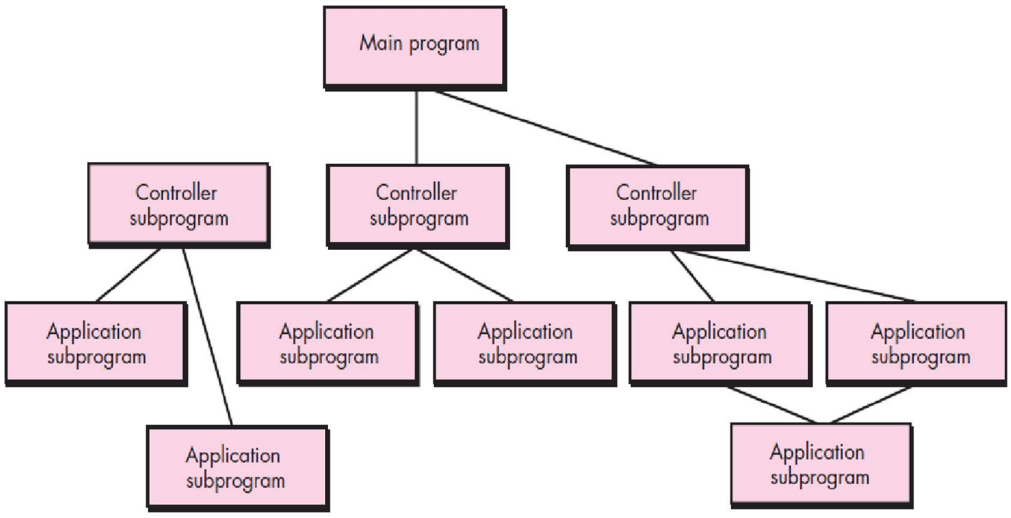


Pipes and filters

- **Call and Return Architecture**: Components call each other to perform actions.

  Example: In a **calculator app**, when the user clicks the "add" button, the system calls the addition function to provide a result.

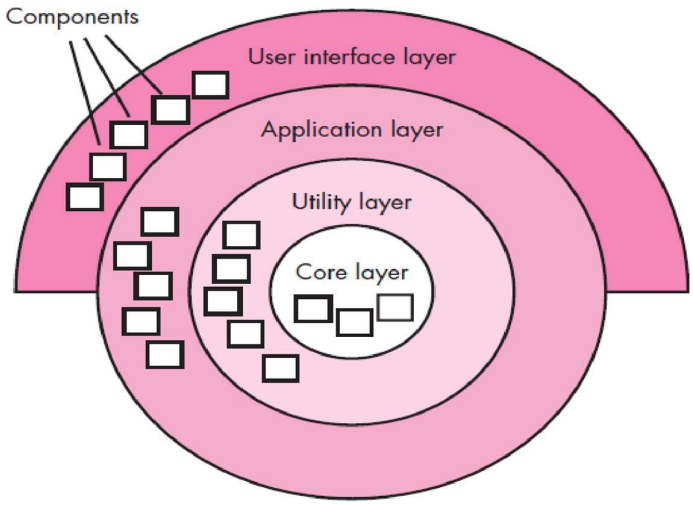
  Purpose: To allow components to communicate and perform tasks when requested.



- **Object-Oriented Architecture**: The system is based on **objects** that represent real-world things.

  Example: In a **car rental system**, there are objects like **Car**, **Customer**, and **Rental Agreement**, each with its own data and functions.

  Purpose: To model real-world objects in the system, making it easier to understand and manage.



3. Architectural Patterns

Architectural patterns are **solutions to common problems** in software design. These patterns provide proven ways to structure a system effectively.

Types of Architectural Patterns with Examples:

- **Model-View-Controller (MVC):** This pattern splits the system into three parts:
  - **Model**: Manages the data.
  - **View**: Displays the data.
  - **Controller**: Handles user input and updates the model and view.

  Example: In an **online store**:

  - The **Model** holds the product information.
  - The **View** shows the products to the users.
  - The **Controller** manages adding/removing items from the cart.

  Purpose: To separate the system into distinct parts, making it easier to maintain and update.

- **Microservices**: The system is divided into **small, independent services**, each handling a specific function. **Example**: In an **e-commerce site**, one service handles payments, another manages product inventory, and another manages user accounts.

  Purpose: To allow each part of the system to function independently, making it easier to scale and update specific parts.

- **Monolithic Architecture**: The system is built as a **single large unit** where all parts are tightly connected.

  Example: A **basic blog platform** where the user interface, content management, and database are all part of one system.

  Purpose: To build a simple system where everything is bundled together.

- **Event-Driven Architecture**: The system responds to **events** in real-time, like user actions or changes in the system.

  Example: In a **social media app**, a notification is sent when a new message is received.

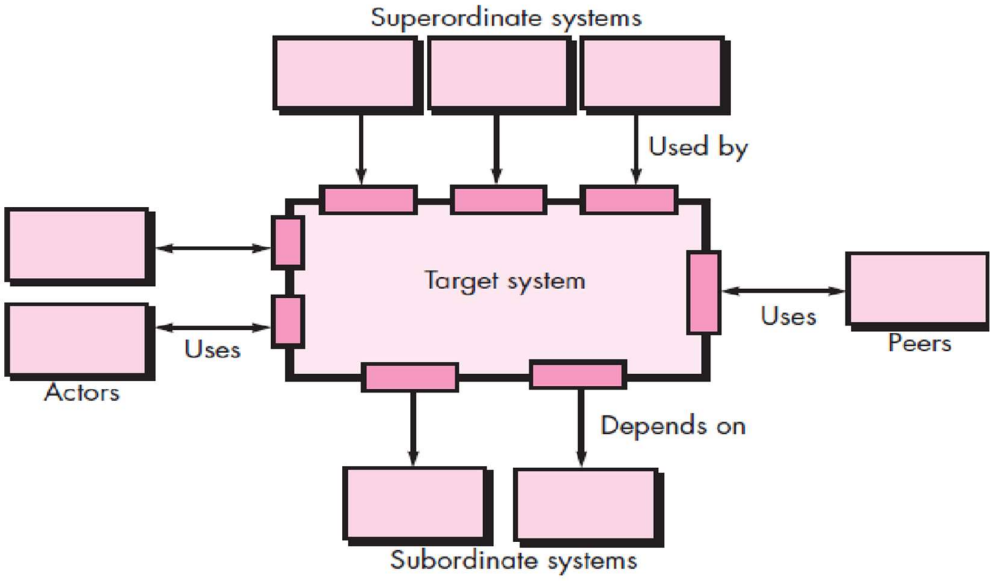
  Purpose: To handle real-time updates or changes in the system efficiently.

Difference between architectural styles and architectural patterns:

Feature	Architectural Styles	Architectural Patterns
Definition	General ways to organize and structure a system.	Specific, reusable solutions to common design problems.
Purpose	Provides an overall structure or layout for the system.	Solves specific design challenges within a system.
Scope	Broad and high-level, guiding the system's architecture.	Focused on solving particular issues or patterns of use.
Example	Layered architecture (dividing into layers like UI, logic, and database).	MVC (Model-View-Controller), separating data, display, and interaction.
Analogy	Like choosing a layout for a house (e.g., 2 floors).	Like using a specific building technique for a house.
When to Use	When deciding the overall organization of the system.	When solving a known problem in system design.

Context Models

A **context model** shows how a software system interacts with the **outside world**, like users, other systems, or devices. It is a simple diagram that illustrates what the system talks to and what information flows in and out.



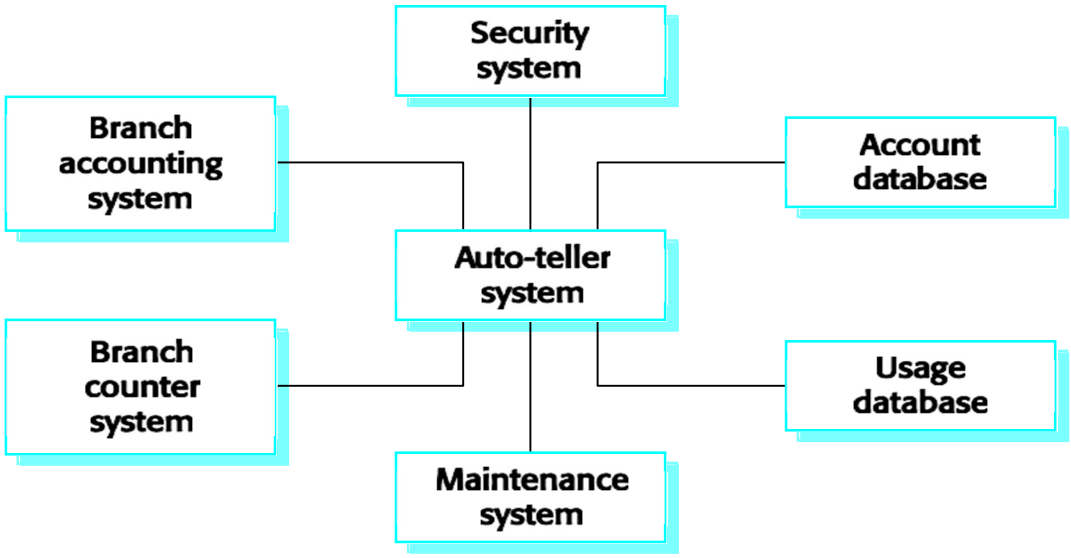
Example: In an **ATM system**:

- **User**: The person withdrawing or depositing money.
- **Bank Database**: Where the ATM checks account balances.
- **Card Reader**: The part that reads the user's card.
- **Cash Dispenser**: The part that gives the user money after a withdrawal.

Purpose of a Context Model

The purpose of a context model is to:

1. **Clarify how the system fits into a larger environment**.
2. **Show how the system interacts** with users, other systems, or devices.
3. **Identify the flow of information** between the system and external entities.



Conclusion

The **purpose of architectural design** is to create a clear plan for how the software system will be built. It helps ensure that the system is well-organized, easy to maintain, and can handle growth. By using the right **architectural styles** and **patterns**, software engineers can build systems that are efficient, scalable, and easy to manage.