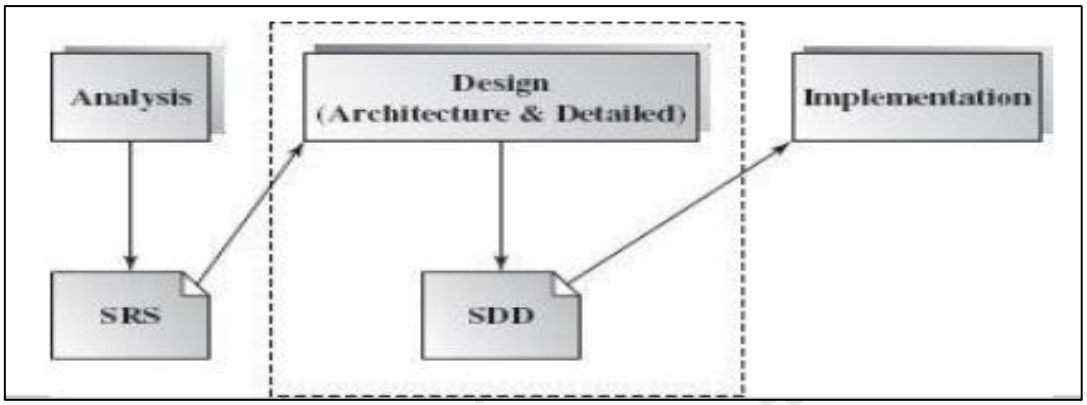
Software Design and its Goal?

Software Design: is the process of defining the architecture, components, and interfaces of a software system to meet specific requirements.

Goal: The goal of software design is to create a model that fulfills all customer requirements and ensures a smooth and successful implementation.



Breakdown:

Analysis Phase:

- **Purpose:** This phase involves gathering and documenting system requirements, leading to the creation of the Software Requirements Specification (SRS).
- **SRS Document:** The SRS outlines all functional and non-functional requirements, serving as the foundation for the subsequent design and development phases.
- **Flow:** The design process begins once the SRS is finalized.

Design Phase:

- **Architectural Design:**
  - **Purpose:** Develop a high-level structure of the system by identifying the main components and how they interact.
  - **Goal:** To ensure that the system architecture supports all the requirements specified in the SRS.

- **Detailed Design:**
  - **Purpose:** Define the specific details of each component, including algorithms, data structures, and interfaces.
  - **Goal:** Break down the architecture into smaller, implementable parts that can be directly used for coding.

- **Output:** The Software Design Document (SDD), which includes detailed design specifications and diagrams.
- **Flow:** The implementation phase follows once the design is documented in the SDD.

Implementation Phase:

- **Purpose:** The system is built based on the SDD, with developers writing, integrating, and testing the code.
- **Flow:** The SDD serves as a blueprint for the implementation process.

Why Software Design is important?

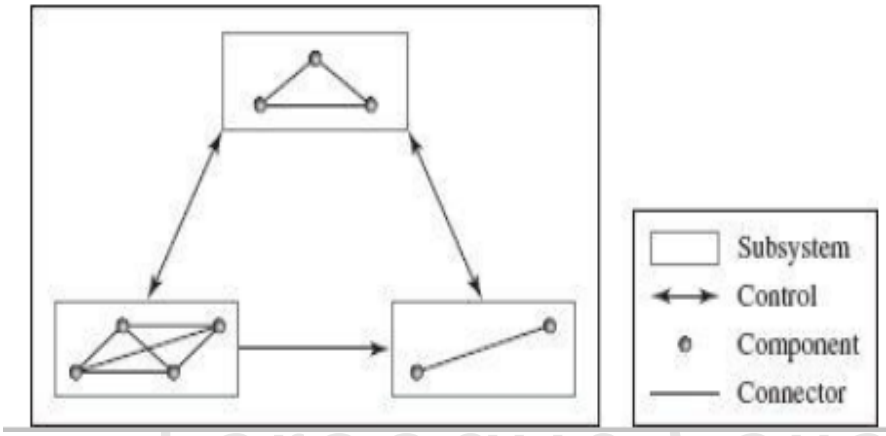
- As software systems continue to grow in scale, complexity, and distribution, their proper design becomes extremely important in software production.
- Any software, regardless of its application domain, should have an overall architecture design that guides its construction and development.

- **Scale:** refers to the size or extent of the system, such as the number of users it can support, the amount of data it can handle, or the number of components involved.
- **Complexity:** in software architecture refers to how difficult it is to design, understand, or maintain a system. As software systems grow in size and functionality, they often involve more components, layers, and interactions. This increases the overall complexity, making the system harder to manage.
- **Distribution:** in software architecture refers to how different parts of a system are spread out across different locations or devices.

What is Software Architecture?

- Software Architecture is the description of elements from which system is built, interactions among those elements, patterns that guide their composition, and constraints on the patterns.
- Considers system as a collection of components and their interactions.
- Components are such things as clients and servers, databases, layers, etc.
- Interactions among components can be procedure calls, shared variable access, etc. At the architectural level, we also consider system-level issues such as capacity, consistency, performance, etc.

- Subsystems are the building blocks of a software system, each handling a specific domain of functionality.
- Components within these subsystems encapsulate discrete functionalities, making the system modular and maintainable.
- Control logic composes the interactions between these components, ensuring that the system operates correctly and efficiently.
- Connectors are crucial for facilitating communication, allowing different parts of the system to work together harmoniously.



Software Architecture from an Industry Perspective(Acc to Grady Booch 1991):

Software architecture refers to the logical and physical structure of a system, shaped by decisions made during development. According to Booch (1991), it encompasses the organization of a software system, including its components, their relationships, and the principles guiding its design and evolution.

Logical Structure: This involves the abstract organization of classes, modules, or components and their interactions to meet system requirements. It's often represented with design models like class or component diagrams.

Physical Structure: This pertains to the deployment of software in the real world, including component distribution across hardware, network topology, and data storage. Deployment diagrams are commonly used to visualize this.

Strategic Decisions: High-level choices made early in the project that affect the overall direction and structure of the system. For example, choosing between microservices and monolithic architectures impacts scalability and deployment.

Tactical Decisions: Specific, lower-level choices made during development, such as selecting a database technology, based on prior strategic decisions.

Software Architecture from an Industry Perspective (According to IEEE Standard 1471–2000)

Software architecture is the fundamental organization of a system, covering its components, their relationships, and the principles guiding its design and evolution.

Fundamental Organization of a System

This refers to the basic structure of the system, including major components and their roles. For example, an e-commerce platform might have subsystems like user management, product catalog, order processing, and payment handling.

Components

Components are encapsulated units of functionality, such as classes, modules, or services. In an e-commerce system, components might include a "Shopping Cart" service or a "User Authentication" component, each with specific functions.

Their Relationships to Each Other

This describes how components interact and depend on one another. For instance, "User Authentication" may work with "User Management" to validate credentials before accessing the "Shopping Cart."

Their Relationships to the Environment

The environment includes external systems, users, and hardware. In an e-commerce platform, this could involve external payment gateways, network infrastructure, and third-party APIs.

Principles Guiding Its Design and Evolution

Principles like modularity, scalability, and security ensure the architecture remains robust and adaptable over time.

Software Architecture [BASS, CLEMENTS, KAZMAN]

Software architecture is the structure of a system, including its elements, their properties, and their relationships.

Who is responsible for developing the architecture design?

- ✓ Software architects and designers are involved in this process.
- ✓ They translate (map) the software system requirements into architecture design.
- ✓ During the translation process, they apply various design strategies to divide and conquer the complexities of an application domain and resolve the software architecture.

Farooque Sajjad