Software Architecture and Design

CSC224

Lecture 5

Learning Objectives:

- Understand the purpose and importance of software architecture in system design.
- Learn about architectural design principles and patterns.
- Explore common software architecture styles and their applications.
- Understand how to evaluate and document software architectures.

- **1. What is Software Architecture?**
- **Definition:**
  Software architecture refers to the high-level structure of a software system, describing its components, their interactions, and the principles guiding its design and evolution.
- **Purpose:**
- **Communication:** Acts as a blueprint for stakeholders.
- **Design Decisions:** Guides implementation and development.
- **System Qualities:** Ensures scalability, maintainability, performance, and security.

2. Key Concepts in Software Architecture

- **A. Architectural Design Decisions**
- The architecture of a system is influenced by:
  - **Functional Requirements:** What the system should do.
  - **Non-Functional Requirements:** Constraints like performance, security, and reliability.
  - **Business Goals:** Cost, time-to-market, and long-term maintainability.
  - **Technological Environment:** Tools, platforms, and programming languages.
- **B. Software Architecture vs. Design**
- **Architecture:** Focuses on the high-level structure and components of the system.
- **Design:** Involves more detailed decisions about components, algorithms, and data structures.

3. Architectural Patterns and Styles

- Architectural patterns are reusable solutions to common design problems. Below are some common patterns:
- **A. Layered Architecture**
- **Structure:** System is divided into layers, each providing services to the layer above it.
- **Examples:**
  - Presentation Layer (UI).
  - Business Logic Layer.
  - Data Access Layer.
- **Advantages:**
- Promotes separation of concerns.
- Easier to maintain and test.
- **Disadvantages:**
- Performance can be affected by the added overhead of layers.

- **B. Client-Server Architecture**
- **Structure:** Divides the system into clients (requesters of services) and servers (providers of services).
- **Examples:** Web applications, database systems.
- **Advantages:**
- Centralized control and management.
- Easier to update and secure.
- **Disadvantages:**
- Server dependency can lead to bottlenecks or single points of failure.

- **C. Microservices Architecture**
- **Structure:** System is composed of small, independently deployable services.
- **Examples:** Amazon, Netflix.
- **Advantages:**
- Enables scalability and flexibility.
- Easier to update and test individual services.
- **Disadvantages:**
- Increased complexity in communication and management.

- **D. Event-Driven Architecture**
- **Structure:** Components interact by producing and consuming events.
- **Examples:** Real-time analytics systems, IoT applications.
- **Advantages:**
- High scalability and responsiveness.
- Loose coupling between components.
- **Disadvantages:**
- Debugging and tracing can be challenging.

- **E. Pipe-and-Filter Architecture**
- **Structure:** Data flows through a series of components (filters), each transforming it.
- **Examples:** Data processing pipelines.
- **Advantages:**
- Supports reuse and concurrency.
- Easy to modify and scale.
- **Disadvantages:**
- Can be inefficient if data transformation is computationally expensive.

- **4. Principles of Good Architectural Design**
- **Modularity:** Decompose the system into independent components.
- **Scalability:** Ensure the system can handle increased workloads.
- **Security:** Design with data protection and secure access in mind.
- **Performance:** Optimize response times and resource utilization.
- **Maintainability:** Make the system easy to update and extend.

5. Documenting Software Architecture

- **Why Document Architecture?**
- Provides a reference for implementation and maintenance.
- Facilitates communication among stakeholders.
- **Common Documentation Approaches:**
- **Views:** Different perspectives on the architecture.
  - Logical View: Focuses on functionality.
  - Development View: Focuses on software components.
  - Physical View: Focuses on deployment.
  - Process View: Focuses on runtime interactions.
- **Diagrams:** Use visual tools like UML to represent components and interactions.
  - Component Diagrams: Show system components and their relationships.
  - Deployment Diagrams: Show the physical deployment of software components.

6. Evaluating Software Architectures

- **Why Evaluate?**
- Ensure the architecture meets requirements before implementation.
- Identify potential risks and weaknesses.
- **Techniques:**
- **Architecture Tradeoff Analysis Method (ATAM):**
  - Identify quality attributes (e.g., performance, security).
  - Evaluate tradeoffs between competing attributes.
- **Scenario-Based Evaluation:**
  - Define use-case scenarios and analyze how the architecture handles them.
- **Prototyping:**
  - Build small-scale prototypes to validate critical decisions.

7. Examples of Architectural Choices

- **A. E-Commerce System**
- **Architecture Style:** Microservices.
- **Components:**
  - User Authentication Service.
  - Product Catalog Service.
  - Order Management Service.
  - Payment Gateway.
- **B. Library Management System**
- **Architecture Style:** Layered Architecture.
- **Layers:**
  - Presentation Layer: User Interface.
  - Business Logic Layer: Issue/Return Book Logic.
  - Data Access Layer: Database Operations.

8. Key Takeaways

- Software architecture is the foundation of a system’s design, influencing its success and longevity.
- Architectural patterns provide proven solutions to design challenges.
- Good architecture balances functionality, quality attributes, and business goals.
- Documentation and evaluation are crucial for ensuring effective and maintainable designs.

Discussion Questions

- Compare the advantages and disadvantages of layered and microservices architectures.
- How do non-functional requirements influence architectural decisions?
- Why is architectural evaluation important in large-scale systems?

Practical Activity

- **Objective:** Practice architectural design and documentation.
- **Task:**
  - Choose a system (e.g., a social media platform or inventory management system).
  - Identify its functional and non-functional requirements.
  - Propose an appropriate architectural pattern and justify your choice.
  - Create a component diagram to represent the system’s architecture.