

Software Architecture: Design Principles

FA-0808 · 2 days

Software Development

Introduction

Explore software architecture styles, modularity and quality concerns through discussion and case analysis. Eight sessions connect design choices to scalability, performance, security, resilience, integration and change.

This two-day foundation develops analysis and design reasoning. It does not guarantee robust, secure production systems or independent implementation of every pattern discussed.

Learning outcomes

- Explain architecture responsibilities and compare major styles and their trade-offs.
- Distinguish architectural styles from object-oriented design patterns.
- Propose modular interfaces and manage dependencies and coupling.
- Evaluate scalability, performance, security and resilience needs.
- Compare messaging, APIs and event-driven integration choices.
- Critique designs and propose maintainable evolutionary changes.

Prerequisites

Basic understanding of software development concepts and practices

Familiarity with at least one programming language

Some exposure to software design and development projects

Training outline

1. Architecture foundations

- Understanding the importance of software architecture
- Roles and responsibilities of a software architect
- Key principles and goals of software architecture
- Types of software architecture (monolithic, client-server, microservices, etc.)

2. Styles and implementation patterns

- Overview of commonly used architectural styles (layered, event-driven, MVC, etc.)
- Relating object-oriented design patterns (singleton, factory, observer) to implementation; distinguish them from architectural styles
- Selecting the appropriate architectural style for a given scenario
- Case analysis of architectural styles and patterns

3. Modularity and components

- Principles of modularity in software architecture
- Benefits and challenges of component-based design
- Designing modular systems using interfaces and components
- Managing dependencies and achieving loose coupling

4. Scalability and performance

- Understanding scalability and performance requirements
- Strategies for scaling software systems (horizontal vs. vertical scaling)
- Techniques for optimizing performance (caching, indexing, load balancing, etc.)
- Examples and trade-offs in scalable and performance-oriented architectures

5. Security in architecture

- Importance of security in software systems
- Common security vulnerabilities and mitigation strategies
- Secure design principles and practices (authentication, authorization, encryption)
- Evaluating secure architectural patterns and implementation considerations

6. Resilience and fault tolerance

- Designing for resilience and fault tolerance
- Handling failures and mitigating their impact
- Redundancy and failover mechanisms: benefits, costs and limits
- Case analysis of resilience and failure handling

7. Integration and communication

- Strategies for integrating software components and systems
- Message-based integration (message queues, publish-subscribe)
- API design and RESTful principles
- Event-driven architecture and event sourcing as distinct design options

8. Evolutionary design

- Understanding the need for adaptable and evolutionary architectures
- Designing for change and maintainability
- Refactoring and architectural refactoring techniques
- Strategies for introducing architectural changes in legacy systems