

Introduction to Software Architecture and Design

Introduction to Software Architecture and Design

(A concise, easy-to-read guide for students and junior developers)

1. What Is Software Architecture?

Aspect	Description
Definition	The high-level structure of a software system, describing its components, their responsibilities, and how they interact.
Goal	Provide a blueprint that guides implementation, ensures quality attributes (e.g., performance, security), and facilitates evolution.
Scope	Concerned with <i>system-wide</i> decisions, not individual line-of-code details.
Stakeholders	Architects, developers, testers, project managers, operations, customers, and sometimes regulatory bodies.

Key Characteristics

- Abstraction** – hides low-level details, focusing on “what” and “why” rather than “how”.
- Decision-making** – each architectural choice is a trade-off among quality attributes.
- Documentation** – models, diagrams, and written rationale serve as a shared reference.

2. Quality Attributes (Non-Functional Requirements)

Category	Example Attributes	Typical Architectural Concerns
Performance	Response time, throughput	Caching, load balancing, async processing
Scalability	Horizontal/vertical scaling	Stateless services, partitioning, micro-services
Reliability	Availability, fault tolerance	Redundancy, graceful degradation, circuit breakers
Security	Authentication, authorization, data protection	Encryption, secure communication, zero-trust

Introduction to Software Architecture and Design

Category	Example Attributes	Typical Architectural Concerns
Maintainability	Modifiability, testability	Layered design, separation of concerns, CI/CD
Usability	User experience, accessibility	Front?end frameworks, responsive design
Portability	Platform independence	Containerization, platform?agnostic APIs

Tip: Prioritize attributes early. Use the **ATAM (Architecture Trade?off Analysis Method)** or **QoS trees** to rank importance.

3. Architectural Styles & Patterns

Style / Pattern	Core Idea	Typical Use Cases
Layered (n?tier)	Organize code into horizontal layers (presentation, business, data).	Enterprise apps, classic web systems.
Client?Server	Separate concerns between requestor (client) and provider (server).	Web applications, mobile back?ends.
Microservices	Decompose a system into independently deployable services.	Large, evolving domains; need for continuous delivery.
Service?Oriented Architecture (SOA)	Reusable services communicating via a bus (often SOAP).	Enterprise integration, legacy system wrap?around.
Event?Driven	Components react to events via pub/sub or message queues.	Real?time analytics, IoT, decoupled pipelines.
Domain?Driven Design (DDD) – Hexagonal / Clean Architecture	Center the domain model, isolate external concerns.	Complex business logic, high maintainability.
Model?View?Controller (MVC)	Separate input (controller), UI (view), and data (model).	Web frameworks (Spring MVC, ASP.NET MVC).
Pipe?and?Filter	Data flows through a series of processing stages (filters).	Compilers, data transformation pipelines.

Introduction to Software Architecture and Design

Style / Pattern	Core Idea	Typical Use Cases
Broker	Mediates communication between components (e.g., message broker).	Distributed systems, middleware.
Peer-to-Peer	Nodes act as both client and server.	File sharing, blockchain.