Software Architecture

Building the Blueprint for Software Systems

Why Do We Need Software Architecture?

Imagine building a skyscraper without blueprints...

- **No plan** → Workers don't know what to build
- **No coordination** → Electrical, plumbing, structure conflict

- **No standards** → Different floors built differently
- **Chaos** → Project fails, building collapses

Same with software! Without architecture, code becomes a mess.

What is Software Architecture (SA)?

1. Software Architecture is about (Three Software Architecture Components)

- Elements
- Interactions of Elements
- Constraints on Elements and interactions

2. Software Architecture provides

- a framework to satisfy requirements
- a basis for software design

Real-World Analogy: City Planning

City Planning	Software Architecture
Buildings (houses, offices, shops)	**Components** (modules, classes, services)
Roads & Infrastructure	**Interfaces & Communication**
Zoning Laws (residential, commercial)	**Constraints & Design Patterns**
Master Plan	**Architectural Framework of Requirements**
City Requirements (population, traffic)	**System Requirements** (performance, scalability)

Key Components of SA

1. **Elements**
    - Modules, components, services, classes
    - Like buildings in a city
2. **Interactions**
    - How components communicate
    - Like roads and utilities connecting buildings
3. **Constraints**
    - Rules and limitations
    - Like building codes and zoning laws

Common SA Patterns

- MVC
- MVVM
- Server/Client
- Other Important Architectures

MVC (Model-View-Controller)

- MVC: Web example
  - Model: data (HTML)
  - View: how the model is presented (CSS)
  - Controller: handles input and updates model/view (JavaScript)
- **Benefits:** Separation of concerns, easier testing, maintainability

- MVC: Restaurant Analogy
  - Model (Kitchen) - Prepares food, manages ingredients
  - View (Dining Room) - What customers see and interact with
  - Controller (Waiter) - Takes orders, coordinates between kitchen and customers

MVVM (Model-View-ViewModel)

- MVVM: Web example
  - **Model:** data (JavaScript objects, API/data layer)
  - **View:** UI display (HTML/CSS)
  - **ViewModel:** connects view and model, handles UI logic and data binding (JavaScript frameworks like Vue, Knockout, Angular)
- **Benefits:** Separation of concerns, easier testing, two-way data binding, maintainability

- **Model (Kitchen, mainly food):**
  - Prepares and manages the food
- **View (Dining Room, only view):**
  - What customers see only
- **ViewModel (Manager):**
  - Middleman: takes orders from the dining room, communicates with the kitchen, and updates the dining room when food is ready
  - Handles logic and coordination

Core Concepts

Aspect	MVC (Model-View-Controller)	MVVM (Model-View-ViewModel)
Components	Model, View, Controller	Model, View, ViewModel
Main Link	Controller connects Model & View	ViewModel connects Model & View
Data Binding	Manual, not built-in	Built-in, supports two-way binding
User Input	Handled by Controller	Handled by View, passed to ViewModel

Structure & Responsibilities

- **MVC**
  - Controller manages user input and updates Model/View.
  - View is aware of Model.
  - Simpler, best for small to medium projects
- More tight coupling among M,V, and C

- **MVVM**
  - ViewModel handles UI logic and data binding (UI component to UI variables).
  - View is *not* directly aware of Model.
  - Suited for complex UIs and larger projects
- Less tight coupling among M,V, and VM

Testability & Maintenance

Feature	MVC	MVVM
Testability	Moderate; Controller tightly coupled with View	High; ViewModel easily tested, loosely coupled
Maintainability	Good for simple apps	Excellent for complex apps
Code Reuse	Limited	High, especially ViewModels

When to Use

- **MVC:**
  - Simple, smaller projects
  - When direct data binding is not needed
- **MVVM:**
  - Large, complex, or data-driven UIs
  - When maintainability, testability, and two-way data binding are priorities

Client-Server Architecture

- **Divides software into two roles:**
  - **Client:** Requests services or data
  - **Server:** Provides services or data
- **How it works:**
  - Client sends a request over a network
  - Server processes the request and sends back a response

Key Features

- **Clear separation of roles**
- **Multiple clients** can connect to a single server
- **Standard protocols** (e.g., HTTP, TCP/IP) enable communication
- **Scalable and flexible** for many types of applications

Real-World Analogy

- Like a coffee shop:
  - **Client:** Customer ordering coffee
  - **Server:** Barista making and serving the coffee

Other Important Architectures

- Microservices
- Layered Architecture
- Event-Driven

Microservices

- Food Court analogy: Each restaurant (service) specializes in one thing
    - Pizza Service
    - Noodle Service
    - Burger Service

In a real world: Netflix

- Thousands of microservices: user profiles, recommendations, video streaming, billing.

Layered Architecture

- Office Building analogy: Each floor has a specific purpose
  - 4F: Presentation
  - 3F: Business Logic
  - 2F: Data Access
  - 1F: Database

In a real world: TCP/IP Stack (Internet)

- Four layers of TCP/IP: Link (hardware), Internet (routing), Transport (delivery), Application (web, email, apps).

Event-Driven

- Emergency Response analogy: When the fire alarm rings, everyone responds
    - Event → Fire Dept
    - and Ambulance
    - and Police

In a real world: Amazon Purchase

- A single purchase event triggers inventory updates, payment processing, shipping service, and customer notifications.

How to Choose the Right Architecture?

Ask These Questions:

1. **Scale:** How many users? (Small app vs. Facebook)
2. **Complexity:** Simple calculator vs. Banking system
3. **Team Size:** Solo developer vs. 100-person team
4. **Performance:** Real-time game vs. Blog website
5. **Maintenance:** Will it change frequently?
6. **Technology:** Web, mobile, desktop, embedded?

Architecture Decision Example

Building a University Course Management System:

Requirement	Architectural Choice	Why?
Web + Mobile access	**Client-Server**	Multiple client types
Complex business rules	**Layered Architecture**	Separation of concerns
Real-time notifications	**Event-Driven**	Instant updates
Different user roles	**MVC/MVVM**	Clean UI separation

Key Takeaways

Why Architecture Matters:

- Prevents chaos and technical debt
- Enables team collaboration
- Supports scalability and maintenance

What Architecture Provides:

- Blueprint for development
- Framework for decisions
- Constraints that guide design

How to Apply:

- Understand requirements first
- Choose patterns that fit
- Document decisions clearly