Architectural Design Patterns

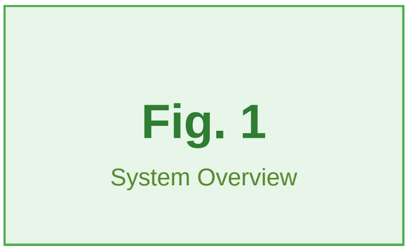

Figure 1: High-level system architecture diagram

The layered architecture pattern separates concerns into distinct tiers, each responsible for specific functionality. This approach improves maintainability and allows teams to work on different layers independently. The presentation layer handles user interactions, the business logic layer contains core application rules, and the data layer manages persistence and retrieval.

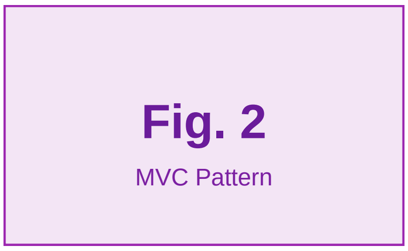

Figure 2: Model-View-Controller pattern flow

The Model-View-Controller (MVC) pattern divides applications into three interconnected components. The Model represents data and business logic, the View displays information to the user, and the Controller handles user input and coordinates between Model and View. This separation enables developers to modify one component without affecting others, promoting code reusability and testability.

Implementing MVC requires careful consideration of communication between components. Controllers should remain thin, delegating complex operations to the Model. Views should be stateless and focused solely on presentation. This discipline ensures that your codebase remains maintainable as it grows and evolves over time.

The benefits of MVC become increasingly apparent in large projects with multiple developers. Each team member can focus on their assigned layer without creating conflicts or dependencies with other areas. Testing becomes simpler because components have clear responsibilities and minimal coupling.

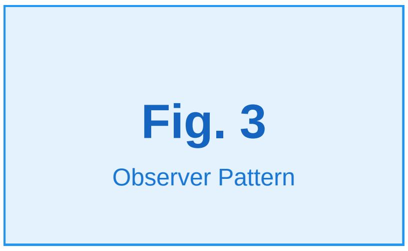

Figure 3: Event-driven Observer pattern

The Observer pattern facilitates loose coupling between objects by establishing a one-to-many relationship. When one object changes state, all registered observers are notified automatically. This pattern is fundamental to event-driven architectures and is extensively used in graphical user interfaces and reactive programming frameworks.

Observer pattern implementation requires defining a subject that maintains a list of observers and methods to register, unregister, and notify them. When the subject's state changes, it iterates through all observers and calls their update method. This approach scales well and makes it easy to add new observers without modifying the subject class.

Understanding and applying these design patterns helps architects create robust, scalable, and maintainable systems. Pattern knowledge accelerates development because proven solutions exist for common problems, reducing the need for reinventing solutions and allowing teams to focus on domain-specific logic rather than infrastructure concerns.