GENERAL

architectural design in software engineering examples

Published: September 19, 2026 | Index ID: #-

Architectural Design in Software Engineering: A Comprehensive Guide with Real World Examples

In today's fast moving technology landscape, software systems must be robust, scalable, and adaptable. Achieving these qualities begins long before code is written, in the early stages of **architectural design**. This discipline shapes how components interact, how data flows, and how the system evolves over time. Whether you're a seasoned architect, a developer stepping into a new project, or a product manager curious about the underlying structure, understanding architectural design—and seeing it in action through concrete examples—can dramatically improve the quality of your software.

What Is Architectural Design in Software Engineering?

Architectural design is the process of defining a system's high level structure, its key components, and the relationships between them. It answers fundamental questions such as:

- Which modules or services are required?
- How will data be stored, accessed, and synchronized?
- What communication patterns will be used?
- How will the system handle growth, failure, and change?

Unlike detailed coding or testing, architectural design focuses on the big picture. It balances functional requirements with non functional constraints—performance, security, maintainability, and cost. A well thought out architecture can prevent costly redesigns, reduce technical debt, and accelerate time to market.

Why Architectural Design Matters

Without a clear architectural blueprint, a project can spiral into a tangled web of interdependent components. Here are some key benefits of investing time in architectural design:

- Scalability – A good architecture anticipates growth, allowing the system to expand horizontally or vertically without major rework.
- Maintainability – Clear boundaries and well defined interfaces make it easier for teams to modify or replace parts of the system.
- Performance – By identifying bottlenecks early, architects can design for optimal data flow and resource usage.
- Reliability – Fault tolerant designs, such as redundancy and graceful degradation, increase uptime.
- Security – Architecture can enforce isolation, least privilege access, and secure communication channels.

Core Architectural Patterns

Architectural patterns are reusable solutions to common design problems. They provide a shared vocabulary for teams and a roadmap for structuring systems. Below are some of the most widely used patterns, each illustrated with examples.

Layered Architecture

The layered approach divides a system into distinct layers, typically:

- Presentation – UI, API endpoints, or client interfaces.
- Business Logic – Domain rules, workflows, and services.
- Data Access – Persistence, repositories, and database interactions.
- Infrastructure – Messaging, logging, and external integrations.

Example: A classic e commerce platform uses a layered architecture where the front end React application talks to a Spring Boot API. The API layer invokes business services that enforce cart and order rules, while repositories interact with a PostgreSQL database. This separation simplifies unit testing and allows the front end to evolve independently.

Microservices Architecture

Microservices split a monolithic application into small, independently deployable services, each responsible for a specific domain capability. Communication typically occurs over lightweight protocols (HTTP/REST, gRPC, or message queues).

Example: **Netflix**employs a microservices model for its streaming platform. Services like recommendation, billing, and content delivery run in isolated containers. This design enables rapid iteration, continuous deployment, and independent scaling based on demand.

Event Driven Architecture (EDA)

EDA centers around the production, detection, and reaction to events. Components emit events to a message broker; subscribers consume events asynchronously.

Example: **Uber**uses an event driven design for its ride matching service. When a rider requests a trip, an event is published. Separate services, such as driver availability and surge pricing, consume the event, update their state, and emit new events that eventually result in a ride assignment. This decouples services and improves responsiveness.

Model View Controller (MVC)

MVC is a software design pattern that separates concerns into:

- Model – Data and business rules.
- View – User interface.
- Controller – Input handling and orchestration.

Example: The Ruby on Rails framework follows MVC. A Rails application's controllers handle HTTP requests, models encapsulate data logic, and views render HTML. This separation enables developers to work on UI and backend logic concurrently.

Service Oriented Architecture (SOA)

SOA focuses on reusable services that expose well defined interfaces. Unlike microservices, SOA services are often larger and may share a common enterprise service bus (ESB).

Example: **Banking systems**frequently adopt SOA. A core banking service handles account management, while separate services manage loans, payments, and compliance. These services communicate through an ESB, ensuring consistent security and transaction management across the enterprise.

Real World Architectural Design Examples

Below are detailed case studies that illustrate how architectural patterns are applied in diverse domains.

E Commerce Platform: Layered + Microservices

Large online retailers need to serve millions of users while maintaining flexibility. One approach combines a traditional layered architecture for the front end with microservices for backend capabilities.

- Front End – A React SPA consumes RESTful APIs.
- API Gateway – Routes requests to appropriate services, handles authentication, and rate limits traffic.
- Microservices – Separate services for product catalog, inventory, cart, checkout, and payment processing.
- Data Stores – NoSQL for product catalog, relational DB for orders, and caching layers (Redis) for session data.
- Event Bus – Kafka streams inventory updates, order events, and promotional changes.

This hybrid design allows the retailer to scale the checkout service during peak sales, update the catalog independently, and roll out new features without downtime.

Healthcare Information System: Layered + SOA

Healthcare applications must comply with strict regulatory standards (HIPAA, GDPR). A layered architecture ensures separation of concerns, while SOA enables integration with legacy systems.

- Presentation Layer – Web portal for patients and clinicians.
- Business Layer – Clinical decision support, appointment scheduling, and billing.
- Integration Layer (ESB) – Mediates communication between the new system and older laboratory or pharmacy systems.
- Data Layer – Secure relational databases with encryption at rest.

By exposing services such as GetPatientRecord and UpdateLabResult, the system can interoperate with external devices and comply with audit requirements.

Social Media Platform: Microservices + Event Driven

Platforms like **Twitter** need real time data pipelines for feeds, notifications, and analytics.

- Microservices – Separate services for posting, retweeting, user profiles, and trending topics.
- Event Bus – Kafka streams user actions; services consume relevant events to update feeds.
- Streaming Analytics – Real time dashboards for engagement metrics.
- Data Stores – Distributed NoSQL for user timelines; graph databases for follower relationships.

Event driven design ensures that a new tweet propagates through the system quickly, while microservices enable independent scaling of high traffic components like the feed generator.

Financial Trading System: Layered + Event Driven + Low Latency

High frequency trading platforms demand sub millisecond response times. Architecture must minimize overhead and guarantee deterministic behavior.

- Presentation Layer – Trading clients (desktop or web) send orders via a low latency protocol.
- Order Matching Service – Core engine written in C++ for speed, operating in a single thread.
- Event Bus – In memory queues deliver market data updates to matching engine and risk modules.
- Data Layer – In memory data structures for order books; persistent logs for audit.

By avoiding network hops and using shared memory for critical paths, the system achieves the necessary

performance while still maintaining modularity.

Internet of Things (IoT) Platform: Edge + Microservices + Event Driven

IoT ecosystems involve millions of devices generating data streams. A robust architecture separates edge processing from cloud analytics.

- Edge Devices – Collect sensor data, perform local filtering, and publish events via MQTT.
- Edge Gateway – Aggregates data, provides initial security checks, and forwards events to the cloud.
- Cloud Microservices – Ingest, transform, and store data; run machine learning models for anomaly detection.
- Event Bus – Kafka or Azure Event Hubs for scalable ingestion.
- Data Stores – Time series databases (InfluxDB) for sensor data; NoSQL for device metadata.

This architecture supports real time monitoring, offline operation, and rapid deployment of new device firmware.

Architectural Design Tools and Methodologies

Choosing the right tools can accelerate design, improve documentation, and facilitate communication across teams.

Unified Modeling Language (UML)

UML provides a standardized set of diagrams—class, sequence, component, and deployment diagrams—to visualize architecture. Many teams use tools like **Enterprise Architect** or **Lucidchart**to create and share UML models.

ArchiMate

ArchiMate is an open standard for enterprise architecture modeling. It extends UML with additional concepts for business, application, and technology layers, making it useful for aligning IT with business strategy.

Domain Driven Design (DDD)

DDD emphasizes modeling the business domain in code. It introduces bounded contexts, entities, value objects, and aggregates to guide architectural decisions. When combined with microservices, each service can map to a bounded context, ensuring clear ownership.

Model Driven Architecture (MDA)

MDA separates platform independent models (PIM) from platform specific models (PSM). Code generators can transform high level models into deployable artifacts, reducing manual coding errors.

Architecture Decision Records (ADRs)

Baca Juga (Artikel Terkait)

- **advanced engineering electromagnetics solution manual edition hardcover by balanis constantine a published by wiley**
  URL: http://www.denniswildberger.com/advanced-engineering-electromagnetics-solution-manual-edition-hardcover-by-balanis-constantine-a-published-by-wiley.pdf
- **advanced engineering electromagnetics solutions manual**
  URL: http://www.denniswildberger.com/advanced-engineering-electromagnetics-solutions-manual.pdf
- **advanced engineering electromagnetics wiley 1989 grading**
  URL: http://www.denniswildberger.com/advanced-engineering-electromagnetics-wiley-1989-grading.pdf
- **advanced engineering fluid mechanics by biswas**
  URL: http://www.denniswildberger.com/advanced-engineering-fluid-mechanics-by-biswas.pdf

Tags: #architectural #design #software