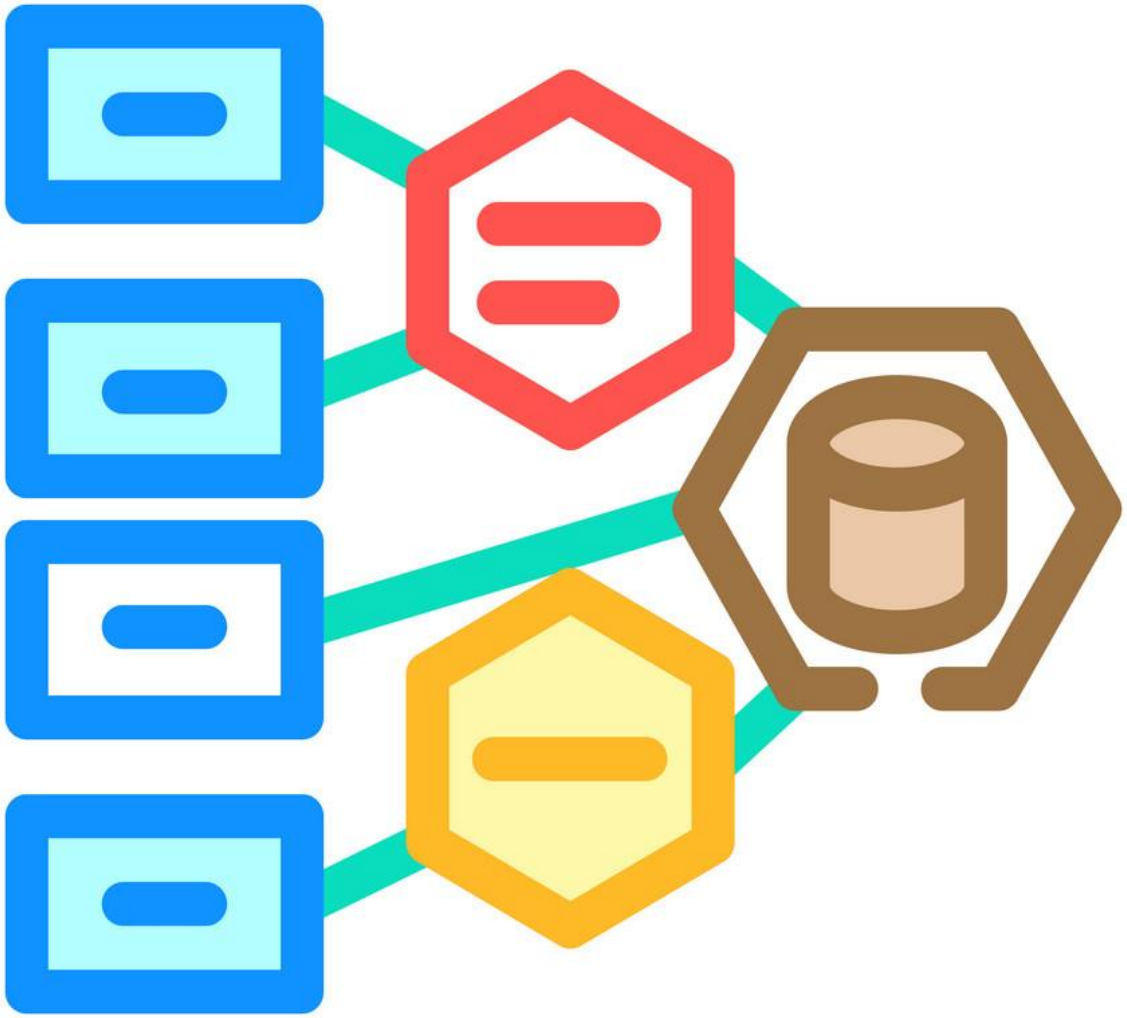
Software Design and architecture

Fall 2024

Dr. Natalia Chaudhry



Outline

- Architecture Documentation in Software Design and Architecture

- Architecture documentation is the process of describing the software architecture of a system.
- It provides a blueprint for developers, stakeholders, and future maintainers, detailing the system’s structure, decisions, and rationale.

Example: A documentation package for a social media platform may include architectural diagrams, descriptions of components like user profiles, notifications, posts, databases, and how they communicate via APIs or messaging queues.

Types of Documentation in Software Architecture

- **High-Level Architecture Documentation**: Focuses on the major components of the system and how they interact. Often presented in a system diagram or component diagram.

A cloud-based video streaming service like Netflix might provide a high-level diagram showing user devices, content delivery networks (CDN), and backend services (e.g., video processing, recommendation engine).

- **Low-Level Architecture Documentation**: Provides detailed descriptions of individual components and their inner workings.

The recommendation engine suggests content to users based on their preferences, viewing history, and trends. It aims to enhance the user experience and increase engagement.

It uses machine learning algorithms to recommend movies, shows, and content categories, which are displayed on the Netflix homepage.

Key Components

- **Data Ingestion Service**: Gathers raw data from various sources such as user behavior (views, likes, ratings), device type, time of day, etc.
  - **Technologies**: Uses Apache Kafka for real-time data ingestion.
  - **Input and Output**: Ingests events (clicks, views, ratings) and feeds them into a data pipeline that prepares them for analysis.
- **Data Processing Pipeline**: Processes raw data to derive features needed for machine learning models, such as recent viewing patterns, genre preferences, and social data.
  - **Technologies**: Uses Apache Spark for distributed data processing.
  - **Steps**:
    - **Data Cleaning**: Removes duplicates and cleans incomplete data.
    - **Feature Engineering**: Extracts useful features (e.g., preferred genres, recent interactions).
    - **Data Aggregation**: Aggregates session-level data (e.g., number of views by genre, duration watched).

- **Model Training Component**: Trains machine learning models to predict and rank content for each user.
  - **Algorithm**: Collaborative filtering and deep learning methods, including matrix factorization, neural networks, and reinforcement learning.
  - **Infrastructure**: Uses GPUs and distributed training on a cloud platform.
  - **Input**: Feature set from the Data Processing Pipeline.
  - **Output**: A recommendation model optimized for fast inference.
- **Recommendation Service API**: Provides an API that receives a user's ID and returns a list of recommended content.
  - **Technologies**: RESTful API powered by a microservices architecture.
  - **Endpoints**:
    - **Get Recommendations**: GET /recommendations/{user_id}
    - **Feedback Capture**: POST /feedback to capture post-recommendation user interactions.

Further following aspects may also be outlined…

Data Flow and Interaction
Deployment Diagram
Database Details

Contd.. *Types of Documentation in Software Architecture*

- **Architectural Decision Records (ADR)**: A structured way of documenting significant architectural decisions, their trade-offs, and reasoning.

This documentation helps ensure that the rationale behind significant design decisions is preserved for future reference, supporting effective communication, continuity, and evolution of the system.

Key Components of an ADR

Each ADR typically contains the following sections:

1. **Title**: A brief description of the decision.
2. **Context**: Background information explaining why this decision was needed, including any relevant requirements, constraints, or challenges.
3. **Decision**: The choice that was made, clearly defined.
4. **Consequences**: The impact of the decision on the system, including trade-offs, limitations, and benefits.
5. **Status**: The current status of the decision (e.g., proposed, accepted, deprecated).
6. **Alternatives Considered**: Other options that were evaluated and why they were not chosen.

ADRs are particularly helpful in agile environments where the architecture is iterative and decisions are frequently revisited.

In the context of Netflix, ADRs would document architectural decisions made to support the streaming platform's vast and complex architecture, especially given the demands for scalability, reliability, and user personalization.

Here's an example ADR that Netflix might have used when deciding on the adoption of microservices architecture.

Title: Migration from Monolithic to Microservices Architecture

Context:
Netflix originally operated on a monolithic architecture, but as the platform grew, this design created scalability and reliability challenges. With millions of global users streaming content simultaneously, it became difficult to scale parts of the system independently, and failures in one component affected the entire application.

Decision:

Adopt a microservices architecture, decomposing the monolithic application into individual, loosely coupled services.

Each service would be responsible for a specific function, such as user authentication, content recommendation, and video streaming.

Consequences:

•**Positive**:

- **Scalability**: Services can scale independently based on demand, optimizing resource usage.
- **Reliability**: A failure in one service (e.g., recommendations) does not bring down other parts of the application.
- **Flexibility**: Teams can develop, deploy, and update services independently, enabling faster releases and experimentation.

•**Negative**:

- **Complexity**: Managing communication between multiple services introduces complexity, requiring a robust API gateway, load balancing, and monitoring.
- **Data Consistency**: Different services may require consistent views of user data, so mechanisms like eventual consistency and synchronization need to be implemented.
- **Increased Operational Overhead**: Requires extensive infrastructure for container orchestration, service discovery, and fault tolerance.

Status:
Accepted and implemented in phases. Initial services were carved out from the monolithic system, with critical components like user authentication and content catalog management migrated first.

Alternatives Considered:
- **Remain Monolithic**: Keep the monolithic structure but optimize components for performance. This was rejected due to inherent scaling limitations.
- **Adopt a Service-Oriented Architecture (SOA)**: Considered moving to SOA, but it was deemed too tightly coupled compared to microservices, reducing scalability potential.

In this example, Netflix’s decision to adopt microservices would be a key architectural decision, documented for future teams to understand the reasoning, benefits, trade-offs, and impact on the system. This ADR would help guide maintenance and any future architectural adjustments as the application scales or introduces new features.

Contd.. *Types of Documentation in Software Architecture*

- **Design and Interface Documentation**: Describes the design principles (e.g. SOLID, encapsulation etc), component interfaces, and interactions in detail.

A **component interface** defines the ways in which different components in a software system interact with each other. It specifies the inputs, outputs, and interaction protocols for each component, enabling components to communicate while maintaining encapsulation and modularity.

The **User Profile Service** manages user-specific data, such as personal information, preferences, and watch history. It provides methods for retrieving and updating user profile information, which other components (e.g., Recommendation Engine, Content Service) can access.

The **User Profile Service** interface exposes a set of endpoints (or methods) that allow other components to interact with it. In a REST API-based system, these endpoints would be HTTP-based; in a microservices setup, they might use gRPC or messaging protocols.

Components of Architecture Documentation

A summary of the system, its goals, and key constraints.

- **Component Diagrams**: High-level representations showing how major components interact within the system.
- **Data Flow Diagrams**: Describe how data moves through the system.
- **Deployment Diagrams**: Show the physical distribution of software components across servers and devices.
- **Behavioral Models**: Include sequence diagrams, activity diagrams, or state machines.
- **System Architecture Patterns**: Examples include layered, client-server, microservices, event-driven, etc.

Documenting Architectural Decisions

- **ADR Template**: Include sections like the problem description, alternatives considered, chosen solution, and rationale.
- **Trade-offs**: Discuss the trade-offs made during decision-making, such as performance vs. maintainability.
- **Justification**: Provide reasons behind architectural decisions, including alignment with business goals, cost, or specific technical requirements.

Quality Attributes in Architecture Documentation

- **Scalability**: Will the system handle increasing load or demand?
- **Performance**: How well does the system perform under various conditions?
- **Security**: How is the system protected against unauthorized access and data breaches?
- **Maintainability**: Can the system be easily updated or extended?
- **Fault Tolerance**: How does the system handle failures or errors?

Documenting Non-Functional Requirements (NFRs)

- NFRs are constraints or qualities that the architecture must meet, such as:
  - Performance requirements
  - Availability and reliability
  - Usability
  - Security
- It is essential to address NFRs early in the architecture documentation process.

Tools for Architecture Documentation

- **UML (Unified Modeling Language)**: Use diagrams such as component, deployment, sequence, and class diagrams.
- **C4 Model**: A modern approach to document software architecture using Context, Containers, Components, and Code diagrams.
- **Archimate**: A modeling language for describing enterprise architectures.
- **Documenting with Text**: Describing architecture using clear, concise, and structured textual formats.

That’s it