Capítulo 7 – Arquitetura

1.Arquitetura de Software

A arquitetura de software é o planejamento estrutural de um sistema, definindo a organização dos seus componentes e a interação entre eles. Uma boa arquitetura facilita a manutenção, escalabilidade e evolução do software. Empresas que desenvolvem sistemas complexos, como bancos e plataformas de e-commerce, precisam de arquiteturas bem definidas para suportar alto volume de acessos e transações.

Aplicação no Mercado

Um exemplo prático é a construção de um sistema de vendas online. A utilização de uma Arquitetura em Camadas permitiria separar a interface do usuário, as regras de negócio e o banco de dados, garantindo maior flexibilidade e segurança. Já para empresas que lidam com múltiplos serviços, como Netflix e Amazon, a Arquitetura de Microsserviços permite escalar diferentes módulos separadamente, evitando que um erro em um serviço afete todo o sistema.

2. Debate Tanenbaum-Torvalds

Em 1992, Andrew **Tanenbaum** e Linus **Torvalds** debateram sobre a melhor arquitetura para sistemas operacionais.

- **Tanenbaum defendia microkernels**, onde o núcleo do sistema operacional gerencia apenas funções essenciais.
- **Torvalds optou por um kernel monolítico**, como no Linux, onde todas as funções são integradas em um único núcleo.
  Esse debate evidenciou como decisões arquiteturais impactam a evolução dos sistemas.

3. Padrões Arquiteturais

3.1 Arquitetura em Camadas

Organiza o sistema em **camadas hierárquicas**, onde cada camada só pode se comunicar com a imediatamente inferior. A **Arquitetura em Três Camadas** é um exemplo clássico:

1. **Interface com o Usuário** – apresentação e interação com o usuário.
2. **Lógica de Negócio** – regras e operações do sistema.
3. **Banco de Dados** – persistência das informações.
   Esse modelo facilita a **manutenção** e **escalabilidade** do software.

Exemplo no mercado: Em um sistema bancário, essa arquitetura pode ser aplicada para garantir a separação das operações financeiras (cálculo de juros, verificação de saldo) da interface do usuário. Isso possibilita que mudanças na interface não afetem a lógica de negócios.

3.2 Arquitetura MVC (Model-View-Controller)

Divide a aplicação em três partes:

- **Modelo**: gerencia os dados e regras de negócio.
- **Visão**: representa a interface do usuário.
- **Controlador**: gerencia as interações entre Modelo e Visão.
  Esse padrão melhora a **separação de responsabilidades**, facilita a **testabilidade** e permite **múltiplas interfaces**

para os mesmos dados.

Exemplo no mercado: Aplicado em plataformas de comércio eletrônico, como a Shopify. O **modelo** gerencia o estoque e pedidos, a **visão** exibe os produtos ao usuário e o **controlador** coordena as interações entre ambos.

3.3 Arquitetura baseada em Microsserviços

Divide o sistema em **módulos independentes** (serviços autônomos), que se comunicam por APIs.
Vantagens:

- Escalabilidade granular.
- Releases independentes.
- Uso de tecnologias variadas para cada serviço.
  Desafios:
- Complexidade na comunicação distribuída.
- Latência na interação entre serviços.
- Dificuldade na gestão de transações distribuídas.

Exemplo no mercado: No setor de streaming, como a Netflix, cada serviço (recomendação de filmes, controle de assinaturas, gestão de catálogo) opera de forma independente, garantindo estabilidade e escalabilidade.

3.4 Arquitetura Orientada a Mensagens

Utiliza **filas de mensagens** para comunicação assíncrona entre sistemas.
Benefícios:

- Desacoplamento temporal e espacial entre componentes.
- Maior **resiliência** e **escalabilidade**.

Exemplo no mercado: Em um sistema de delivery como o iFood, pedidos podem ser processados de forma assíncrona, evitando que falhas momentâneas impeçam a conclusão de transações.

3.5 Arquitetura Publish/Subscribe

Baseada na troca de eventos entre **publicadores** e **assinantes**, permitindo que múltiplos sistemas sejam notificados simultaneamente sobre um evento específico.
Diferente das Filas de Mensagens:

- Eventos são distribuídos para **múltiplos assinantes**.
- Assinantes recebem notificações assíncronas.

Exemplo no mercado: Um sistema de reservas de passagens aéreas pode notificar diferentes serviços (milhagens, contabilidade, envio de e-mails) assim que uma passagem for comprada.

3.6 Outros Padrões Arquiteturais

- **Pipes & Filtros**: fluxo de dados processado em etapas independentes (exemplo: comandos Unix).
- **Cliente/Servidor**: arquitetura clássica onde clientes solicitam serviços a um servidor remoto.
- **Peer-to-Peer (P2P)**: sistemas descentralizados onde cada nó pode atuar como cliente e servidor (exemplo: torrents).

Exemplo no mercado: Aplicações de blockchain utilizam **Peer-to-Peer** para garantir a descentralização e evitar pontos únicos de falha.

4. Anti-Padrão Arquitetural: "Big Ball of Mud"

O anti-padrão **Big Ball of Mud** descreve sistemas sem estrutura arquitetural clara, onde qualquer módulo pode se comunicar diretamente com outro, gerando um código difícil de manter e modificar.
Exemplo real: Um sistema bancário que cresceu de 2,5 milhões para 25 milhões de linhas de código, tornando-se impossível de gerenciar.