PUC Minas

Resenha

Projeto de Software

Gustavo Delfino Guimarães

1489062@sga.pucminas.br

Matrícula: 836079

Belo Horizonte, MG – Fevereiro de 2025

Engenharia de Software Moderna

Capítulo 7: Arquitetura

O capítulo 7 trata sobre a arquitetura de software e sua importância para a implementação de um sistema. O texto traz uma introdução ao tema explicando duas diferentes visões de dar significado ao termo “Arquitetura de Software”, na primeira visão o termo é classificado com um “projeto de mais alto nível” onde é tratado a organização do todo e não como visto em capítulos anteriores na abstração de um problema em uma parte menor, seria a forma de organizar essa abstrações e suas conexões para que elas tenham o maior sentido do todo. A segunda forma de se caracterizar “Arquitetura de Software” é trazida pelo autor Ralph Johnson, que na arquitetura de software é onde se toma as decisões mais importantes de um projeto para a implementação de um sistema.

O capítulo também menciona o Debate Tanenbaum-Torvalds, ocorrido em 1992, sobre arquiteturas monolíticas e microkernels em sistemas operacionais. Tanenbaum defendia que microkernels eram a abordagem ideal, enquanto Torvalds argumentava a favor do modelo monolítico usado no Linux. O debate ilustra como decisões arquiteturais podem gerar impactos de longo prazo e serem difíceis de reverter.

A primeira arquitetura a ser explicada é a **Arquitetura em Camadas.** A arquitetura em camadas é um dos padrões mais utilizados no desenvolvimento de software desde as décadas de 1960 e 1970. Seu principal objetivo é organizar o sistema em módulos hierárquicos, onde cada camada tem responsabilidades bem definidas e só pode se comunicar com a camada imediatamente inferior. Essa abordagem facilita a manutenção, evolução e reutilização dos componentes do sistema. O autor utilizou o exemplo dos protocolos de rede para explicar a arquitetura em camadas onde cada protocolo utiliza outro protocolo.

- HTTP ➡️ TCP ➡️ IP ➡️ Ethernet

Além de simplificar a complexidade do software, a arquitetura em camadas controla as dependências entre os componentes, garantindo uma separação clara de responsabilidades. Essa estrutura facilita a substituição de tecnologias específicas (por exemplo, trocar TCP por UDP) e o reaproveitamento de camadas já existentes.

A Arquitetura em Três Camadas surgiu como uma evolução do sistema de camadas nos anos de 1980. Com o avanço das redes e do hardware, foi possível distribuir as aplicações em diferentes camadas, tornando os sistemas mais flexíveis e escaláveis.

Essa arquitetura é composta por três camadas principais:

1. Interface com o Usuário (Camada de Apresentação): Responsável por interagir com o usuário, exibindo informações e processando entradas, como cliques e formulários.

2. Lógica de Negócio (Camada de Aplicação): Implementa as regras de negócio do sistema.

3. Banco de Dados: Armazena todas as informações do sistema.

É importante frisar que esta é uma arquitetura distribuída onde geralmente a interface é executada na máquina do cliente, a camada de negócio é executada em um servidor de aplicação juntamente com o banco de dados.

Além disso, existe a possibilidade de utilizar arquiteturas em duas camadas, onde a interface e a lógica de negócio ficam na mesma camada, executando no cliente. No entanto, esse modelo sobrecarrega as máquinas dos usuários, exigindo maior poder de processamento.

A Arquitetura MVC(Model-View-Controller) é uma das arquiteturas mais divulgadas e utilizadas por desenvolvedores. Antes mesmo de termos um aprofundamento sobre arquitetura na faculdade, o “padrão” MVC já havia sido citado por diversos professores. Segundo o livro o padrão foi proposto na década de 80 para a utilização da linguagem smalltalk-80(pioneira no conceito de interface e orientação a objeto que conhecemos hoje) para facilitar o desenvolvimento de interfaces gráficas.

O padrão define que as classes do MVC devem se dispor da seguinte forma:

- Model: Responsável por armazenar e gerenciar os dados e regras de negócio, sem qualquer dependência das camadas visuais ou de controle.
- View: Define a interface gráfica do sistema, exibindo informações e interagindo com o usuário.
- Controller: Interpreta os eventos de entrada (como cliques e digitação) e coordena a atualização do modelo e da visão.

A utilização deste modelo força mais ainda aos princípios de alta coesão e baixo acoplamento. Graças ao MVC foi permitido a especialização do trabalho no desenvolvimento de sistemas, hoje podemos ter desenvolvedores apenas de front-end(view) e de back-end(controller e model). Esta arquitetura também permite que classes de model sejam reutilizadas para diferentes tipos de view.

Uma discussão mais recente com a forte implementação de frameworks Web é a diferenciação entre o MVC e a arquitetura três camadas, embora ambas compartilham semelhanças, elas possuem objetivos distintos. Enquanto o MVC foca na organização da interface do usuário, a arquitetura três camadas (apresentação, lógica de negócio e banco de dados) define a estrutura geral de sistemas distribuídos. É importante deixar claro também que o conceito de interface mudou com a utilização de views, enquanto nas três camadas algumas interfaces eram grandes matrizes utilizando linha de comando e textos.

Nos anos 2000, com a popularização da Web, frameworks MVC (como Spring, Django e Ruby on Rails) foram criados para organizar aplicações Web, adaptando o conceito original. Nesses frameworks, o modelo representa a camada de persistência de dados, enquanto a visão e o controlador compõem a interface e a lógica de apresentação. Assim, os sistemas

Web MVC frequentemente se assemelham a uma arquitetura de três camadas, gerando certa confusão entre os termos.

A Single Page Applications (SPAs), é uma arquitetura que revolucionou aplicações web, onde todo o código necessário (HTML, CSS e JavaScript) é carregado no navegador logo ao iniciar a aplicação. Assim, ao interagir com a aplicação, o usuário não percebe recargas de página, pois a atualização do conteúdo acontece de forma dinâmica. Um exemplo clássico de SPA é o Gmail, onde a lista de e-mails se atualiza sem a necessidade de recarregar a página inteira. Isso é possível graças à comunicação assíncrona com o servidor.

As SPAs melhoram a experiência do usuário ao eliminar recargas de página, proporcionando maior velocidade e fluidez. Essa abordagem é amplamente utilizada em aplicações modernas e pode ser implementada com frameworks como Vue.js, React e Angular, que simplificam o desenvolvimento e melhoram a performance da aplicação.

A Arquitetura de Microserviços surge com a alta de metodologias ágeis, que apesar do desenvolvimento ágil a manutenção não entrava nessa metodologia pois o sistema acabava um monolito dividido em sub blocos.

Na arquitetura Monolítica, mesmo que um sistema seja modular no desenvolvimento, ele é executado como um único processo, o que pode gerar dependências problemáticas entre os módulos. Pequenas alterações podem causar falhas em todo o sistema, tornando o processo de liberação lento e burocrático.

Já na arquitetura de microsserviços em vez de um único bloco, o sistema é dividido em múltiplos serviços independentes, cada um rodando em seu próprio processo. Isso reduz os efeitos colaterais das mudanças e permite maior autonomia dos times de desenvolvimento.

Os principais Benefícios dos Microsserviços são:

Atualizações independentes – Cada serviço pode ter seu próprio ciclo de vida e ser atualizado sem impactar o restante do sistema.

Escalabilidade – Serviços podem ser escalados separadamente, permitindo alocar mais recursos apenas para os módulos que demandam mais desempenho.

Flexibilidade tecnológica – Diferentes serviços podem ser desenvolvidos com tecnologias distintas (ex.: Java para cadastro, Python para recomendações).

Maior resiliência – Se um serviço falha, o restante do sistema continua funcionando, reduzindo indisponibilidades completas.

A popularização da computação em nuvem tornou os microsserviços ainda mais viáveis, pois permite escalar sistemas facilmente sem necessidade de manter infraestrutura própria.

A Lei de Conway sugere que a arquitetura de software reflete a estrutura organizacional das empresas. Assim, empresas de tecnologia com times distribuídos adotam microsserviços para alinhar software e organização, tornando a inovação mais ágil e eficiente.

A adoção de microsserviços tornou-se um diferencial competitivo para empresas que precisam de flexibilidade, escalabilidade e inovação contínua no desenvolvimento de software.

Na Arquitetura Orientada a Mensagens a comunicação entre clientes servidores ocorre por meio de um serviço intermediário, a fila de mensagens. O cliente envia mensagem para a fila enquanto os servidores retiram as mensagens e as processam, utilizando da estrutura FIFO para consumir as mensagens.

ALgumas das vantagens são:

- O cliente pode continuar o processamento após enviar uma mensagem;
- As mensagens não são perdidas caso o servidor falhe;
- Clientes e servidores não precisam conhecer uns aos outros;
- Não precisam estar online simultaneamente cliente e servidor.
- Vários servidores podem consumir mensagens de uma mesma fila.

Arquiteturas Publish/Subscribe (Pub/Sub): Nesse tipo de arquitetura as mensagens são eventos, existem os publicadores(publishers) e assinantes(subscribers). Os publicadores geram eventos e os enviam para o broker de mensagens, enquanto os assinantes recebem notificações dos eventos que o assinaram.

Neste conceito de arquitetura vários assinantes podem receber a mesma mensagem e um assinante recebe uma “notificação” quando acionado.

O capítulo ainda trata de alguns padrões como pipes e filtros, cliente/servidor e P2P. Ao final ele trata de “Anti-padrões arquiteturais” que são maneiras de não se desenvolver um sistema como o big ball of mund, onde ocorre um grande número de acoplamentos, baixa coesão e vai contra os princípios SOLID. Poderia citar aqui também o “Go Horse” tratado em sala pelo professor Aramuni, muito difundido e pouco coeso.